



Stephan Karrer
st.karrer@arcor.de

Diese PDF ist für Sie persönlich codiert. Sie steht Ihnen für Lern- und Ausbildungszwecke zur Verfügung. Jeglicher Inhalt, einschließlich der Layouts und Anordnungen, unterliegt dem Schutz des Urheberrechts sowie weiterer Schutzrechte.

Ohne schriftliche Genehmigung des HERDT-Verlags für Bildungsmedien sind Reproduktion und Weitergabe der PDF – auch in Teilen – ausdrücklich verboten und ziehen zivil- und strafrechtliche Konsequenzen nach sich.

XML 1.1

Grundlagen

Elmar Fuchs, Heiko Schröder

5. Ausgabe, Januar 2015

ISBN: 978-3-86249-407-1

XML11



1 Informationen zu diesem Buch	4	7 XML Schema	64
1.1 Voraussetzungen und Ziele	4	7.1 Der Unterschied zwischen Schema und DTD.....	64
1.2 Aufbau und Konventionen	5	7.2 Grundlagen zu XML Schema.....	66
2 Einführung in XML	8	7.3 Schema-Grundgerüst.....	68
2.1 Auszeichnungssprachen	8	7.4 Einfache Typen.....	70
2.2 XML.....	8	7.5 Datentypen.....	76
2.3 Die Verwendung von XML	11	8 Komplexe Elemente in Schema.....	82
3 Aufbau eines XML-Dokuments	14	8.1 Was ist ein komplexes Element?.....	82
3.1 Die grundlegende XML-Syntax	14	8.2 Definition eines komplexen Elements	82
3.2 Prolog als Definition eines XML-Dokuments.....	16	8.3 Indikatoren.....	86
3.3 Anlegen von XML-Elementen	18	8.4 Schema erweitern	90
3.4 Attribute eines Elements.....	21	8.5 Schema 1.1.....	91
3.5 Kommentare hinzufügen	22	8.6 Übungen.....	95
3.6 Wohlgeformtheit eines XML-Dokuments.....	23	9 Formatierungssprachen	96
3.7 Ein XML-Dokument erstellen	23	9.1 Übersicht der Sprachen	96
3.8 XML-Editor Editix.....	25	9.2 Grundlagen von XSL.....	96
3.9 Schnellübersicht.....	26	9.3 Einbinden von CSS.....	98
3.10 Übungen	27	9.4 Übung.....	101
4 Elemente der DTD	28	10 XPath	102
4.1 Dokumenttyp-Definition	28	10.1 XPath-Grundlagen	102
4.2 Definition einer internen DTD	29	10.2 XML-Prinzipien.....	104
4.3 Deklarieren der Elementtypen.....	30	10.3 XPath 2.0	109
4.4 Angabe der Elemente.....	30	10.4 Übung.....	112
4.5 Externe Teilmenge der DTD	34	11 XSL und XSLT.....	114
4.6 Gültiges Dokument.....	37	11.1 Einführung in XSL	114
4.7 Schnellübersicht.....	38	11.2 Einbinden einer XSL-Datei	116
4.8 Übungen	39	11.3 Templates	116
5 DTD – Attribute von Elementen.....	40	11.4 Selektion mit Filter in XPath.....	119
5.1 Attributlisten-Definition.....	40	11.5 Inhalte der Elemente ausgeben.....	120
5.2 Attributtypen.....	45	11.6 Reihenfolge der Template-Aufrufe	122
5.3 Referenz auf Entitäten	47	11.7 Übungen.....	124
5.4 Datentyp Notation	51	12 XSLT-Elemente	126
5.5 Schnellübersicht.....	54	12.1 Schleifen und Fallunterscheidungen	126
5.6 Übung	55	12.2 Schleifenbildung	126
6 Namensräume	56	12.3 Elemente sortieren	129
6.1 Grundlagen zu Namensräumen	56	12.4 Einfache Fallunterscheidung.....	130
6.2 Deklaration von Namensräumen	57	12.5 Komplexe Fallunterscheidung	132
6.3 Externe DTD und eigener Namensraum	60	12.6 Übungen.....	135
6.4 Übungen	62		

13 Links in XML.....	136	15.4	SAX.....	168	
13.1	Einführung in XLink.....	136	15.5	XML-Datenblöcke in HTML5.....	169
13.2	XLink.....	136	15.6	Übungen.....	172
13.3	Einfache Links.....	138			
13.4	Erweiterte und multidirektionale Links.....	140			
13.5	XBase.....	144			
14 XQuery.....	146	16 XHTML 1.1.....	174		
14.1	XQuery-Grundlagen.....	146	16.1	Überblick zu XHTML.....	174
14.2	XPath basierte XQuery-Abfragen.....	148	16.2	XHTML-Dokument deklarieren.....	175
14.3	Abfragen mit FLWOR-Ausdrücken.....	150	16.3	Elemente in XHTML.....	177
14.4	Gestaltung der Ausgabe der Auswertungsergebnisse.....	152	16.4	Attribute und Werte in XHTML.....	178
14.5	Auswertung verbundener Dokumente.....	153	16.5	JavaScript und Stylesheets in XHTML.....	179
14.6	Übungen.....	155	16.6	W3C MarkUp Validation Service.....	180
			16.7	Übung.....	181
15 DOM und SAX.....	156	17 SVG.....	182		
15.1	DOM.....	156	17.1	SVG-Grundlagen.....	182
15.2	Erzeugen eines XML-DOM.....	157	17.2	Erzeugung von SVG-Grafiken.....	182
15.3	Ansprechen der Knotenelemente.....	158	17.3	Einbindung von SVG in HTML5.....	185
			Stichwortverzeichnis.....	186	

1 Informationen zu diesem Buch

In diesem Kapitel erfahren Sie

- ✓ an wen sich dieses Buch richtet
- ✓ welche Vorkenntnisse Sie benötigen
- ✓ welche Software Sie für die Arbeit mit diesem Buch benötigen
- ✓ wie dieses Buch aufgebaut ist und welche Konventionen verwendet werden

1.1 Voraussetzungen und Ziele

Zielgruppe

Zielgruppe dieses Buches sind alle, die Daten per XML verwalten, austauschen und verarbeiten möchten.

Empfohlene Vorkenntnisse

Für die generelle Thematik XML sind die genannten Vorkenntnisse nicht zwingend erforderlich, erleichtern aber das Verständnis. Einige Kapitel setzen jedoch entsprechende Kenntnisse voraus.

- ✓ Kenntnisse in HTML
- ✓ Kenntnisse in CSS (Kapitel 9)
- ✓ Kenntnisse in JavaScript (Kapitel 15)

Lernziele

In diesem Buch lernen Sie, wie die XML-Daten angegeben werden müssen, welche Möglichkeiten Ihnen zur Verfügung stehen, die Struktur der Daten zu verifizieren und wie Sie die verschiedenen XML-Elemente anhand von Vorgaben selektieren können. Außerdem erlernen Sie, wie Sie mithilfe von JavaScript XML-Daten auswerten und formatiert im Browser anzeigen.

Hinweise zu Software

Sofern nicht anders vermerkt, wird im Folgenden davon ausgegangen, dass Sie die unten genannten Browser oder neuere Versionen verwenden.

Name	Version	Erhältlich unter der Internetadresse:
Internet Explorer	11	http://windows.microsoft.com/de-de/internet-explorer/download-ie
Mozilla Firefox	33	http://www.mozilla-europe.org/de/firefox/
Opera	25	http://de.opera.com/
Apple Safari	7	http://www.apple.com/de/safari/
Google Chrome	38	http://www.google.com/chrome/

Verwendete Software

Betriebssystem	Programm
Windows, Unix/Linux, Mac OS X	Free XML Editor Editix (http://www.freexmleditorsite.com/)
Windows	Altova XMLSpy 2015 Professional Edition (http://www.altova.com/)

In diesem Buch werden XML-Tools für das Betriebssystem Windows benutzt. Für das Betriebssystem Linux gibt es ähnliche Programme. Eine Auflistung von weiteren XML-Tools für Linux, Windows, Solaris bzw. Macintosh finden Sie unter anderem auf der englischsprachigen Webseite <http://www.garshol.priv.no/download/xmltools/>.

Free XML Editor Editix

Die Verwendung der Software Free XML Editor Editix 2008 ist für den nicht kommerziellen Einsatz kostenlos. Da diese Software auf Java basiert, kann sie unter den Betriebssystemen Windows, Unix/Linux und Mac OS X eingesetzt werden. Neben der freien existiert eine kostenpflichtige Variante mit einem erweiterten Leistungsspektrum.

Die umfangreiche Software unterstützt Sie beim Erstellen eines XML-Dokuments. Sie erlaubt Ihnen über eine grafische Oberfläche die komfortable Bearbeitung einer XML-Datei und bietet neben der Ansicht des Quelltexts auch die übersichtliche Baumansicht der Struktur des XML-Dokuments. Zudem können Sie mit der Software DTD/Schema und XSLT/XQuery bearbeiten und die Dokumente diesbezüglich validieren.

- ▶ Laden Sie die Software von der Adresse <http://www.freexmleditorsite.com/download.html> herunter.
- ▶ Starten Sie die Installation mit einem Doppelklick auf die ausführbare Datei.

Altova XMLSpy

Altova bietet ein ganzes Paket von Programmen für die Arbeit im Zusammenhang mit XML Dokumenten an. Das Programm XMLSpy gibt es als Professional und als Enterprise Edition. Bereits die Professional Edition unterstützt alle im Buch behandelten Themen. Für Testzwecke wird eine voll funktionsfähige 30-Tage-Testversion angeboten.

- ▶ Laden Sie die Software von der Webseite aus dem Downloadbereich der Webseite <http://www.altova.com/de/xmlspy.html> herunter.
- ▶ Starten Sie die Installation mit einem Doppelklick auf die ausführbare Datei.

Besonderheit der Browser Google Chrome und Opera

Aus Sicherheitsgründen laden Chrome und Opera aus lokalen XML-Dateien keine XSL-Dateien, sodass die im Buch erwähnten Beispiele ab Kapitel 10 lokal nicht ausgeführt werden können. Stattdessen wird nur eine leere Seite angezeigt. Wenn die Dateien auf einen Webserver übertragen und von dort aufgerufen werden, dann werden die XML-Dateien korrekt wiedergegeben.

Um auch lokal die Ergebnisse zu erhalten, müssen Sie die Browser mit der zusätzlichen Option `--allow-file-access-from-files` starten. Damit erhalten Sie die Möglichkeit, die Daten aus XML-Dateien auch lokal per XPath und XSL zu formatieren.

1.2 Aufbau und Konventionen

Aufbau und inhaltliche Konventionen des Buchs

- ✓ Am Anfang jedes Kapitels finden Sie die Lernziele.
- ✓ Die einzelnen Kapitel enthalten Übungen, mit deren Hilfe Sie die erlernten Kapitelinhalte wiederholen können.
- ✓ In Kapitel 2 - 8 lernen Sie die Grundlagen sowie die Strukturdefinition von XML-Dokumenten mittels DTD und XML-Schema kennen.
- ✓ Die Kapitel 9 - 14 zeigen die Darstellung von XML-Dokumenten und die Selektion einzelner Dokumentbestandteile. Es werden die Themen Einsatz von CSS, XSL, XSLT, XPath, XLink und XQuery behandelt.
- ✓ Das Kapitel 15 beschäftigt sich mit dem programmatischen Zugriff auf XML-Dokumente.
- ✓ Die Kapitel 16 und 17 stellen mit XHTML und SVG zwei Anwendungen von XML vor.

Typografische Konventionen

Damit Sie bestimmte Elemente auf einen Blick erkennen und zuordnen können, werden diese im Text durch eine besondere Schreibweise hervorgehoben. So werden beispielsweise wichtige Begriffe **fett** hervorgehoben.

<i>Kursivschrift</i>	kennzeichnet Datei- und Ordnernamen, Hyperlinks und Bezeichnungen für Menüs bzw. Menüpunkte, sowie Programmelemente wie Register oder Schaltflächen.
<code>Courier New</code>	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
<i>Courier New Kursiv</i>	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder ins Programm eingegeben werden.
[]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
/	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen Schrägstrich voneinander getrennt.
↵	Zu lange Programmzeilen oder Internetadressen (die eigentlich nicht getrennt werden dürfen) werden auf der nächsten Zeile fortgesetzt.

Was bedeuten die Symbole im Buch?



Hilfreiche Zusatzinformation



Warnhinweis



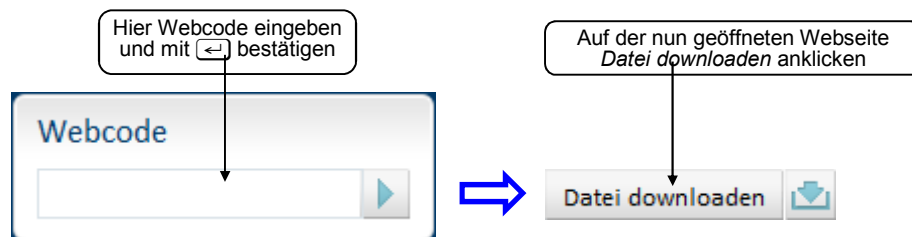
Praxistipp



Download-Adresse

Downloaddateien

Mit dem Kauf dieses Buches haben Sie die Berechtigung erworben, die Beispiel-, Übungs- und Ergebnisdateien von der Website des HERDT-Verlages unter www.herdt.com herunterzuladen. Nutzen Sie hierzu den Webcode, der sich rechts oben auf der Titelseite dieses Buches befindet:



Weitere Medien von HERDT nutzen

Hat Ihnen das vorliegende Buch gefallen, empfehlen wir Ihnen weitere HERDT-Bücher wie z. B.

- ✓ JavaScript - Grundlagen
- ✓ SQL - Grundlagen und Datenbankdesign

Diese und weitere Lehr- und Lernmaterialien können Sie direkt auf unserer Website www.herdt.com bestellen.

Wir wünschen Ihnen viel Spaß und Erfolg mit diesem Buch.

Ihr Redaktionsteam des HERDT-Verlags

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

2 Einführung in XML

In diesem Kapitel erfahren Sie

- ✓ etwas zur Geschichte der Auszeichnungssprachen
- ✓ den Zusammenhang zwischen SGML, HTML und XML
- ✓ welche Vorteile XML gegenüber HTML besitzt

2.1 Auszeichnungssprachen

Auszeichnungssprachen sind dafür konzipiert, neben den Textdaten, die den Inhalt eines Dokuments darstellen, auch spezielle Informationen hinzuzufügen. Auszeichnungssprachen gibt es schon sehr lange. Früher benutzten beispielsweise die Druckereien ihre eigene Sprache, um Anmerkungen an Textpassagen zu setzen. Somit wusste der Drucker, der das Dokument vom Autor erhielt, worauf er bei bestimmten Stellen im Text achten sollte.

Mit Verbreitung der Computer wurde es möglich, Texte digital zu verarbeiten. Es entstand eine Vielzahl von Programmen, einige davon als Speziallösung für große Verlage und Druckereien. Eines davon ist das Satzprogramm **T_X** von Donald E. Knuth, das bereits Mitte der 70er-Jahre entwickelt wurde. **T_X** ist eine Software zur Formatierung von Dokumenten, mit der nahezu alle Aufgaben einer Druckerei gelöst werden können, u. a. das Setzen komplexer Formeln und Tabellen. Der Umfang der Beschreibungssprache **T_X** ist festgelegt, wobei besonderer Wert auf das einfache Erstellen von technisch-wissenschaftlichen Dokumenten gelegt wurde.

1985 entstand die Meta-Auszeichnungssprache **SGML** (**S**tandard **G**eneralized **M**arkup **L**anguage) für die Inhaltsstruktur von Dokumenten, die in der Norm ISO 8879 als Standard festgelegt wurde. Diese Sprache beinhaltet eine Vielzahl von Befehlen, deren Dokumentation mit 500 Seiten sehr umfangreich ist. Außerdem wurde die **D**ocument **T**ype **D**efinition, kurz **DTD**, eingeführt, mit der sich eigene Auszeichnungssprachen beschreiben lassen.

Darauf beruhend wurde die Auszeichnungssprache **HTML** (**H**yper**T**ext **M**arkup **L**anguage) als „Sprache des Internets“ entwickelt. Festgelegt wurde die Spezifikation der Sprache vom W3-Konsortium (kurz W3C). Das ist eine internationale Organisation, die an der Entwicklung neuer Protokollspezifikationen und Architekturen für das World Wide Web arbeitet. Mitglieder sind z. B. Firmen wie Adobe, Microsoft und Sun Microsystems. Lange Zeit basierten alle verfügbaren Webseiten auf HTML.

2.2 XML

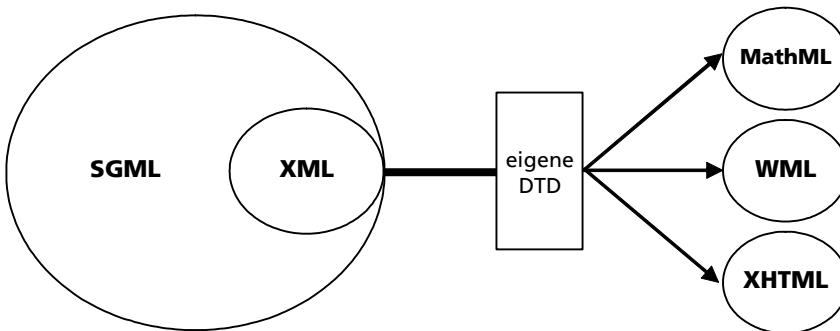
Entstehung

Die Meta-Auszeichnungssprache SGML und die Auszeichnungssprache HTML haben einige Nachteile. SGML ist ein sehr komplexer Standard mit unzähligen Möglichkeiten und somit zum Design von Webseiten nicht geeignet.

HTML, hat den plattform- und systemunabhängigen Informationsaustausch in den Datennetzen ermöglicht und somit den Internet-Boom ausgelöst. HTML wurde speziell für die Anzeige von Daten entworfen und ist zu unflexibel, um bestimmte Erweiterungen, beispielsweise die direkte Unterstützung von Videos, zu unterstützen. HTML besteht aus einer fest definierten Menge von Auszeichnungen wie beispielsweise Überschriften, Absätze, Tabellen. Aus dem Bestreben, HTML um zusätzliche über den Standard hinausgehende Möglichkeiten zu erweitern, entwickelten sich die Browser durch das nicht abgestimmte Handeln der einzelnen Hersteller unterschiedlich. Im Ergebnis wird nicht jede Seite von jedem Browser einheitlich dargestellt. Ein weiterer Nachteil von HTML besteht darin, dass in einer HTML-Seite im Wesentlichen alles Text ist. Für die gezielte Suche nach Informationen sind die Browser deshalb auf die Volltextsuche angewiesen.

Als Alternative verabschiedete das W3-Konsortium im Februar 1998 den **XML-Standard**, der eine stark vereinfachte Teilmenge von SGML ist. Die Idee des W3-Konsortiums war es, eine Metasprache zu schaffen, welche die grundlegenden Vorzüge von SGML besitzt, auf die Nachteile von HTML verzichtet und für jeden Autor leicht zu erlernen ist. Derzeit umfasst die Standardisierungsdokumentation von XML ca. 35 Seiten, also nur einen Bruchteil des Umfangs der SGML-Beschreibung.

Eine Version der ersten XML Empfehlung von 1998 mit Kommentaren über deren Entstehungsgeschichte von Tim Bray, der selbst Mitglied der Entwicklungskommission war, finden Sie unter <http://www.xml.com/axml/axml.html>.



XML ist eine Teilmenge von SGML

Aufbau von XML

XML (**eX**tensible **M**arkup **L**anguage) ist eine Teilmenge von SGML. In dem Begriff „extensible“ wird schon angedeutet, worum es bei dieser Sprache hauptsächlich geht: Sie ist so konzipiert, dass sie sich erweitern lässt. Dies ist möglich, da XML genau wie SGML keinen vordefinierten Aufbau in Form einer DTD (Document Type Definition) besitzt, sondern lediglich die Vorschriften zum Erstellen einer eigenen Typdefinition bereithält. Somit sind Sie in der Lage, mit einer eigenen DTD neue Sprachelemente zu entwickeln.

In XML wird die Struktur der Auszeichnungssprache definiert. Die definierten Elemente haben keinerlei Bedeutung für das Aussehen. Dies wird beispielsweise erst über Cascading Stylesheets (CSS) definiert. Die definierten Elemente beschreiben vielmehr die logische Struktur eines Dokuments und ermöglichen es, beliebige Strukturen zu bezeichnen.

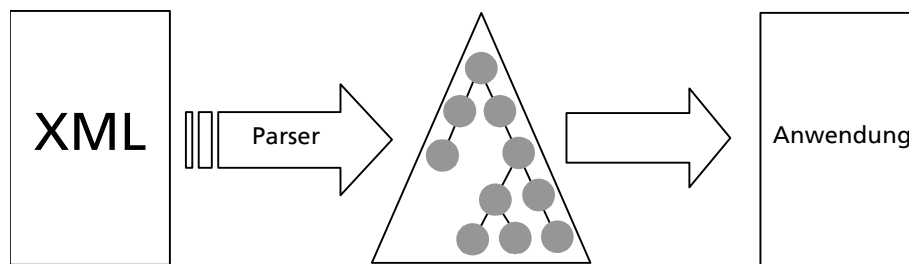
Die Metasprache XML ist **nicht** der Nachfolger der Auszeichnungssprache HTML. Während XML das Instrumentarium zur Bezeichnung von Dokumentstrukturen liefert, stellt HTML einen beschränkten Satz von Elementen zur Verfügung, mit dem Dokumente logisch ausgezeichnet werden können.

Beispiel

<pre><html> <head> <title>Titel der Seite</title> </head> <body> <h1>Überschrift der 1.Ebene</h1> <p>Absatz mit Informationen</p> <p>Absatz mit Informationen</p> </body> </html></pre>	<pre><dokument> <kopf> <titel>Titel der Seite</titel> </kopf> <koerper> <ueber>Überschrift der 1. Ebene</ueber> <absatz>Absatz mit Informationen</absatz> <absatz>Absatz mit Informationen</absatz> </koerper> </dokument></pre>
---	--

Gegenüberstellung der Syntax von HTML und XML

Der Inhalt eines XML-Dokuments ist eine Baumstruktur mit genau einem Wurzelement und stellt die interne Struktur des Dokuments dar. Dieser Baum kann durch einen sogenannten **XML-Parser** aufgebaut werden. Auf diese Weise kann für eine Anwendung, z. B. einen Browser, der Zugriff auf einzelne Baumknoten ermöglicht werden.



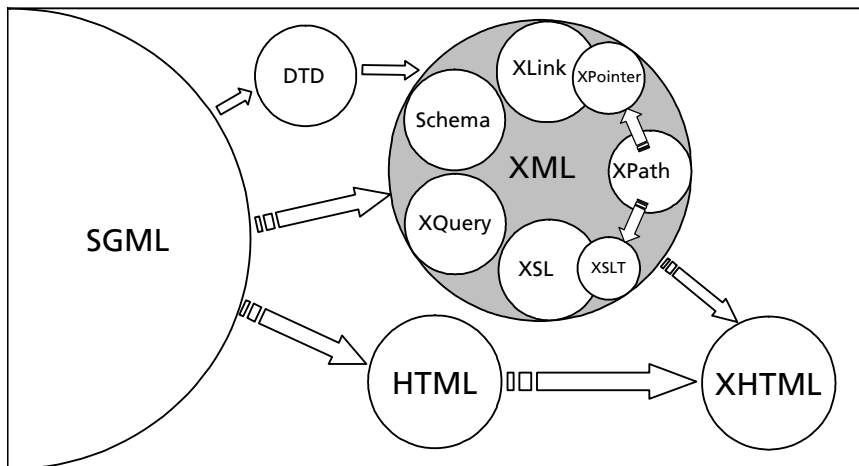
Aufbereitung der XML-Daten über eine Baumstruktur

Derzeit existieren spezielle Sprachen, die von der XML-Spezifikation abgeleitet wurden. Nachfolgend finden Sie eine kleine Auswahl.

Sprache	Erläuterung	Internetadressen
XHTML	Extensible Hypertext Markup Language Neuformulierung von HTML 4.0 auf Basis von XML (vgl. Kap. 16)	http://www.w3.org/TR/xhtml11/
MathML	Mathematical Markup Language Sprache zur Darstellung mathematischer Strukturen	http://www.w3.org/Math
SVG	Scalable Vector Graphics Sprache zur Definition von Grafikobjekten (vgl. Kap. 17)	http://www.w3.org/Graphics/SVG
CML	Chemical Markup Language Informationen über chemische Strukturen	http://cml.sourceforge.net

Weitere XML-verwandte Technologien

DTD	Document Type Definition ist eine Sprache zur Definition von möglichen Elementtypen, Attributen und Entities innerhalb eines XML-Dokuments. DTD benutzt eine SGML-verwandte Syntax.
Schema	Schema ist wie DTD eine Empfehlung zur Definition der Struktur eines XML-Dokuments. Das Erstellen eines Schemas erfolgt über die XML-Syntax und ist somit ebenfalls ein XML-Dokument.
XHTML	Extensible Hypertext Markup Language ist eine Auszeichnungssprache für Webseiten. Sie ist die Neuformulierung von HTML 4.0 in XML 1.0.
XSL	Extensible Stylesheet Language dient unter anderem zum Erstellen der Definition von Stylesheets. Sie beschreiben, wie die XML-Daten dem Benutzer angezeigt werden sollen. Zu XSL gehören XSL-FO (Formatting Objects), XSLT und damit indirekt auch XPath.
XSLT	Extensible Stylesheet Language for Transformations ist eine Sprache für die Umwandlung von XML-Dokumenten in andere Dokumenttypen, wie z. B. HTML oder auch wieder XML. XSLT beschreibt dabei, wie das Dokument transformiert werden soll.
XPath	Extensible Path Language ist eine Sprache zum Adressieren von Teilen eines XML-Dokuments, die sowohl von XSLT als auch von XPointer verwendet werden kann. XPath war früher Bestandteil von XSLT und ist jetzt ein eigener Standard.
XLink	Extensible Linking Language ist eine Spezifikation für Hyperlinks im XML-Dokument. Sie enthält den Adressierungsstandard XPointer.
XPointer	Extensible Pointer Language gibt an, wie Adressen in XLink-Ausdrücken zu verwenden sind. XPointer legt das Format des Teils eines Hyperlinks fest, der auf eine Sprungmarke (Anker) eines XML-Dokuments verweist.
XQuery	Extensible Query Language ist eine Abfragesprache, die XML als Datenmodell verwendet.



Zusammenhang von SGML, HTML, XML und verwandten Technologien

2.3 Die Verwendung von XML

Basis des Datenaustauschs

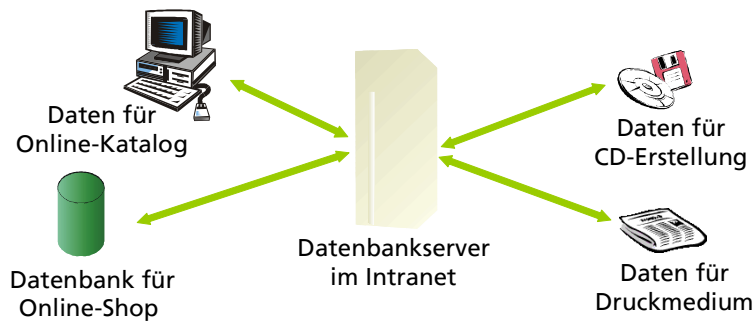
Obwohl ursprünglich als Form zur Darstellung von Dokumenten gedacht, hat sich XML mittlerweile als Sprache für den Datenaustausch zwischen Systemen etabliert. Diese Möglichkeit der Anwendung der Sprache wurde bereits frühzeitig erkannt. Ron Rappaport, ein Industrieanalyst der Firma Zona Research, drückte es 1998 folgendermaßen aus: „XML ist ein einfacherer, flexiblerer und kostengünstigerer Weg, um Daten zwischen verschiedenen Anwendungen auszutauschen, als alle heute vorhandenen Mittel. Es ist ein Schlüsselement der Business-zu-Business-Kommunikation (B2B). XML wird den Informationsaustausch in Firmen revolutionieren, vergleichbar mit der Entwicklung des Telefons, Fax und Fotokopierers. Diese vorgenannten Erfindungen erleichterten den Informationsaustausch erheblich, und XML ist dabei, diese Entwicklung im Internet zu vollziehen.“

Heute wird immer mehr XML verwendet, wenn es um die Darstellung und den Austausch von Informationen in Unternehmen geht. Mitverantwortlich für diese positive Entwicklung sind Softwarehersteller wie IBM, Adobe Systems, Microsoft und Sun Microsystems, die den Standard bereits seit Jahren in ihren Produkten einsetzen. Beispielsweise basiert das Format der Dokumente der Office Suite von Microsoft auf XML. Der Großteil der Softwareanwendungen besitzt heute eine Schnittstelle, um Daten im XML-Format zu importieren und zu exportieren und immer mehr relationale Datenbankmanagementsysteme unterstützen das Format XML.

Ein Beispiel für den möglichen Einsatz von XML als unabhängige Datenplattform ist die Speicherung von Produktdaten im elektronischen Handel (E-Commerce). Benutzt eine Vielzahl von Anbietern eine gemeinsam entwickelte Sprache zum Verwalten des Produktkatalogs, ist es für den Endnutzer auf einfache Weise möglich, nach einem bestimmten Produkt im Internet zu suchen, ohne jede einzelne Webseite des Anbieters aufrufen zu müssen. Er schickt sogenannte Software-Agenten auf die Suche nach dem preiswertesten Angebot. Diese durchsuchen das Internet nach den festgelegten Suchkriterien und liefern dem Kaufinteressenten nur das zurück, was er wissen will: Wie viel kostet ein Produkt bei den verschiedenen Anbietern und wo erhalte ich es am günstigsten?

XML im Unternehmen

Nicht nur im Internet verbreitet sich die Anwendung von XML. Auch die Verwaltung von Firmendaten vereinfacht sich durch die Anwendung des XML-Formats für die Datenspeicherung. Stellen Sie sich vor, Sie sind Händler von Computer-Hardware. In diesem Fall werden Sie eine Datenbank erstellen, welche die Produkte Ihres Unternehmens verwaltet. Ihren Kunden möchten Sie monatlich eine Informations-CD mit den neuesten Produkten zukommen lassen, wobei die Daten zusätzlich im Internet in Form eines Online-Shops zu finden sein sollen. Als Werbemittel versenden Sie einen gedruckten Katalog, den Sie ebenfalls als Download-Möglichkeit im Internet anbieten möchten.



Allein diese Art der Produktdarstellung erfordert bisher eine drei- bis vierfache Datenhaltung, da Sie die Daten für jedes unterschiedliche Medium in das entsprechende Format bringen müssen.

Medium	Datenformat
Datenbankserver	SQL-Abfragesprache
Online-Katalog	PDF (Portable Document Format)
Daten auf CD	HTML
Druckmedium	PostScript, zunehmend auch PDF

Mit XML stellen Sie einmalig die notwendigen Daten zur Verfügung und können diese über eine XSL-Transformation in das gewünschte Ausgabeformat bringen. Mit dem definierten Format von XSL-FO (Formatting Objects) beschreiben Sie, wie die Ausgabe der Daten aussehen soll. Ein Prozessor setzt die Beschreibung um und generiert das entsprechende Ausgabeformat, z. B. ein PDF oder RTF-Dokument.

XML und HTML

Während XML in der beschriebenen Form heute vielfach zum Einsatz kommt, konnte der ursprüngliche Ansatz HTML durch XML abzulösen, nicht realisiert werden. Die als Nachfolgesprache gedachte Empfehlung XHTML stieß bei den Entwicklern von Webseiten auf ein eher gedämpftes Echo. Eine Ursache dafür waren die Anforderungen an einen 100% exakten Code. Jeder Fehler in einer XHTML führt dazu, dass die Seite nicht dargestellt werden kann. Als Konsequenz auf die mangelnde Akzeptanz wurde mit HTML5 ein Weg zur Modernisierung von HTML gewählt. Dieser erlaubt sowohl die flexiblere Anwendung von HTML als auch, wenn der Entwickler es möchte, den Einsatz der strengeren Regeln von XHTML.



Mit XHTML als eine Form der Anwendung von XML beschäftigt sich Kapitel 16.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

3 Aufbau eines XML-Dokuments

In diesem Kapitel erfahren Sie

- ✓ wie Sie ein XML-Dokument erstellen
- ✓ was Sie bei der Angabe von Elementen beachten müssen
- ✓ wie Sie Daten in eine XML-Struktur bringen
- ✓ was unter einem wohlgeformten Dokument zu verstehen ist

Voraussetzungen

- ✓ Editor zum Erfassen des Quelltextes
- ✓ XML-fähiger Browser
- ✓ Grundkenntnisse in HTML (sind hilfreich)

3.1 Die grundlegende XML-Syntax

Abhängig vom gewählten Zeichensatz kann ein XML-Dokument im Text-Format dargestellt werden. Somit kann es in jedem Texteditor, beispielsweise dem Windows-Standard-Editor Notepad, erstellt werden. Das Dokument enthält, wie in HTML, die Elemente und die zu verarbeitenden Daten. Dabei wird das Dokument in zwei Bereiche eingeteilt, den **Prolog** mit den Angaben zum XML-Dokument, der Dokumenttyp-Definition sowie gegebenenfalls weiteren Angaben wie eine zu verwendende Stylesheet-Deklaration für die Darstellung der Datei und den **Bereich** mit der Auflistung der Elemente.

Beschreibung	XML
Prolog (u. a. DTD-Definition)	<code><?xml version="1.0"?></code> <code><!DOCTYPE ...></code>
Elemente	<code><BEREICH></code> <code></BEREICH></code>



Der XML-Standard 1.1 erlaubt die Prolog-Definition `<?xml version="1.1"?>`, jedoch unterstützen die beiden weitverbreiteten Browser Internet Explorer und Firefox diese Version nicht. Deshalb wird in den Beispielen dieses Buches weiterhin die Version 1.0 angegeben. Die XML-Version 1.1 können Sie angeben, wenn die von Ihnen verwendete Anwendung diese Version unterstützt.

Beispiel: *kap03\start.xml*

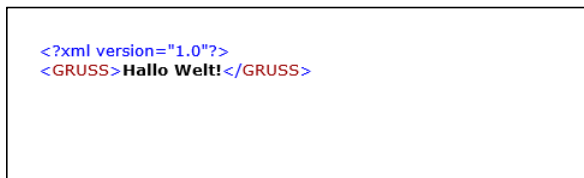
Im folgenden Beispiel erstellen Sie ein XML-Dokument und zeigen es in einem Browser an.

- ▶ Öffnen Sie einen Texteditor.
- ▶ Geben Sie den nachfolgenden XML-Code ein.

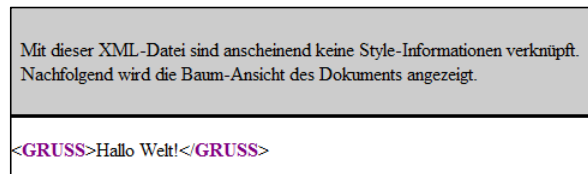
①	<code><?xml version="1.0"?></code>
②	<code><GRUSS>Hallo Welt!</GRUSS></code>

- ① Diese Zeile kennzeichnet das Dokument als XML-Dokument.
- ② Ähnlich wie in HTML wird das öffnende Tag `GRUSS` in spitzen Klammern angegeben. Zwischen dem öffnenden und schließenden Tag befinden sich die zu verarbeitenden Informationen.

- ▶ Speichern Sie den XML-Code unter dem Dateinamen *start.xml*.
- ▶ Starten Sie den Browser Internet Explorer oder Mozilla Firefox.
- ▶ Laden Sie die soeben gespeicherte Datei *start.xml* in den Browser.



„start.xml“ im Internet Explorer



„start.xml“ im Mozilla Firefox

Der Quelltext des XML-Dokuments wird in beiden Browsern angezeigt. Die Elemente und die Daten werden farblich voneinander getrennt dargestellt. Zusätzlich zeigt der Internet Explorer auch die Verarbeitungsvorschrift `<?xml version="1.0"?>` an.

Jede XML-Auszeichnung muss in spitze Klammern `<` `>` gesetzt werden.

Falsch	Richtig
<code>?xml version="1.0"?</code>	<code><?xml version="1.0"?></code>

Alle angegebenen Elemente müssen ein Anfangs- und Ende-Tag besitzen.

Falsch	Richtig
<code><ELEMENT1>Inhalt</code>	<code><ELEMENT1>Inhalt</ELEMENT1></code>

Die Bezeichnung des jeweiligen Elements ist an keine vorgegebene Richtlinie gebunden. Die Groß- und Kleinschreibung von XML-Elementen ist von Bedeutung, da XML case-sensitive ist, d. h. zwischen der Groß- und Kleinschreibung von Elementnamen unterscheidet. Schreiben Sie das Start-Tag groß, müssen Sie das schließende Ende-Tag ebenfalls großschreiben. Wenn Sie das Start-Tag kleinschreiben, müssen Sie auch das Ende-Tag kleinschreiben. Entscheiden Sie sich für eine einheitliche Schreibweise.

Falsch	Richtig
<code>Bild</code>	<code>Bild</code>

Den Namen eines Elements können Sie in XML selbst bestimmen. Er darf aus folgenden Zeichen bestehen:



- ✓ Buchstaben (auf Namen, die mit "xml" beginnen, ist zu verzichten)
- ✓ Ziffern, Unterstrich
- ✓ Umlaute und Sonderzeichen (je nach vereinbartem Zeichensatz)

Diese Zeichen können Sie beliebig miteinander kombinieren. Ein Elementname beginnt mit einem Buchstaben oder einem Unterstrich. Danach sind Buchstaben (je nach Zeichensatz auch Umlaute), Ziffern, Punkte, Unterstriche und Bindestriche zulässig. Doppelpunkte dürfen nur im Zusammenhang mit Namensräumen verwendet werden. Leerzeichen und Fragezeichen sind nicht zulässig.

Reservierte Zeichen

Einige Zeichen dürfen nicht im Text oder Elementnamen verwendet werden, weil sie zur Abgrenzung von Element und Inhalt sowie Attribut und Wert verwendet werden. Dies bedeutet, die Zeichen `<`, `>`, `"`, `'` und `&` dürfen nicht im Inhalt verwendet werden, da sie ein Teil der Auszeichnungssprache sind. Deshalb müssen sie als Entities angegeben werden.

Zeichen	Entity	Erklärung
<code><</code>	<code>&lt;</code>	lower than (kleiner als)
<code>></code>	<code>&gt;</code>	greater than (größer als)
<code>"</code>	<code>&quot;</code>	quotation (Anführungszeichen)
<code>'</code>	<code>&apos;</code>	apostrophe (Apostroph)
<code>&</code>	<code>&amp;</code>	ampersand (kaufmännisches Und)

CDATA-Blöcke

CDATA steht für Character Data. Als CDATA gekennzeichnete Blöcke werden nicht durch den XML-Parser überprüft. Teile eines XML-Dokuments, die reservierte Zeichen benötigen, können in einem CDATA-Block eingeschlossen werden und werden deshalb nicht überprüft. Die mehrfache und aufwendige Verwendung von Entities erübrigt sich damit.

```
<![CDATA[
  Hier können reservierte Zeichen verwendet werden
]]>
```

- ✓ Ein CDATA-Block beginnt mit der Zeichenkette `<![CDATA[` .
- ✓ Ein CDATA-Block endet mit der Zeichenkette `]]>` .
- ✓ Alle im Block enthaltenen Zeichen werden nicht vom XML-Parser als reservierte Zeichen gewertet.
- ✓ Die Zeichenkette zur Kennzeichnung des Endes des CDATA-Blockes darf selbst nicht innerhalb des Blockes vorkommen.

3.2 Prolog als Definition eines XML-Dokuments

Angabe der XML-Version

Jedes XML-Dokument wird mit der Processing Instruction (Verarbeitungsvorschrift)

```
<?xml ...?>
```

eingeleitet. Diese teilt dem Browser bzw. XML-Prozessor mit, dass es sich im Nachfolgenden um ein Dokument mit XML-Daten handelt.

Zusätzlich wird die Versionsnummer der verwendeten XML-Syntax angegeben.

```
<?xml version="1.0"?>
```

Eine Processing Instruction, kurz PI, beginnt mit `<?` und endet mit `?>` und beinhaltet eine spezielle Anweisung für den Prozessor.

Angabe des Zeichensatzes

XML-Dokumente sind Textdaten. Deshalb muss angegeben werden, welcher Zeichensatz in einem Dokument angewendet werden kann und wie die Zeichen in Byte-Form kodiert werden sollen. Fehlt die Angabe des Zeichensatzes, wird als Standard UTF-8 verwendet. In Abhängigkeit von der eingestellten Speicherform des Editors kann eine fehlende Angabe zu Problemen führen.

- Ändern Sie das bisherige Beispiel, um die deutschen Umlaute in das Dokument einzufügen.

Beispiel: *kap04|encoding-start.xml*

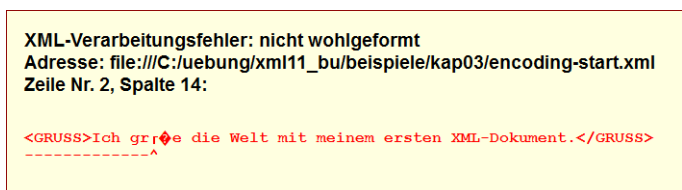
```
<?xml version="1.0"?>
<GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
```

- Öffnen Sie diese Datei in einem Browser.

Verwenden Sie den Windows Editor mit seiner Speichervoreinstellung (ANSI), führt dies zu folgender Ansicht:



Leere Ausgabe im Internet Explorer



Konkreter Fehlerhinweis im Mozilla Firefox

Im Gegensatz zu früher gibt der Internet Explorer keine Fehlermeldung aus, wenn ein ungültiges Zeichen im Text gefunden wurde. Die Anzeige bleibt leer. Ungültig sind in diesem Fall der Umlaut **Ü** sowie das Zeichen **ß**. Diese Zeichen kommen nicht in jedem Zeichensatz vor, da sie spezielle Zeichen der deutschen Sprache sind.

Es wurde eine Zeichensatzfamilie entwickelt, welche die verschiedenen Sprachen auch im Internet unterstützt. Diese Familie besteht aus verschiedenen Zeichensätzen, die eine Kombination der ersten 128 Zeichen des ASCII-Zeichensatzes und des sprachspezifischen Latin-Zeichensatzes darstellen.

ASCII-Zeichensatz 128 Zeichen Code 0 bis 127	Latin-Zeichensatz 128 Zeichen Code 128 bis 255
--	--

Das hat den Vorteil, dass die üblichen lateinischen Groß- und Kleinbuchstaben, die arabischen Ziffern und die Sonderzeichen, wie Satzzeichen oder kaufmännische Zeichen, in all diesen Zeichensätzen immer zur Verfügung stehen.

Wichtige Latin-Zeichensätze

Zeichensatz	für
Latin 1, ISO 8859-1	die westeuropäischen und amerikanischen Sprachen Deutsch, Amerikanisch, Englisch, Französisch
Latin 2, ISO 8859-2	die slawischen Sprachen Polnisch, Rumänisch, Slowakisch
Latin 3, ISO 8859-3	die Sprachen Esperanto, Galizisch, Maltesisch und teilweise Türkisch
Latin 4, ISO 8859-4 bis ISO 8859-8	die Sprachen Estnisch, Lettisch und Litauisch. Dieser Zeichensatz enthält noch weitere Unterteilungen für die Sprachen Bulgarisch, Russisch, Ukrainisch, Serbisch, Mazedonisch usw.
Latin 5, ISO 8859-9	die Sprache Türkisch
Latin 6, ISO 8859-10	die Sprachen Lappländisch und Grönländisch
Latin 9, ISO 8859-15	Überarbeitung der ISO 8859-1; beinhaltet u. a. auch das Euro-Symbol €

Zusätzlich zur ISO-Norm 8859 steht auch die ISO/IEC-Norm 10646 zur Verfügung. Diese beinhaltet die Unicode-Formate UTF-8 und UTF-16, die jeder XML-Prozessor lesen und umsetzen kann.

Im XML-Dokument geben Sie in der Processing Instruction den zu verwendenden Zeichensatz über das Schlüsselwort `encoding` an.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Sie werden überwiegend den Zeichensatz ISO 8859-1 benutzen. Dieser enthält beispielsweise die für die deutsche Sprache typischen Umlaute **Ä**, **Ö**, **Ü** sowie das **ß**.

Beispiel: *kap03|encoding.xml*

- Fügen Sie das Schlüsselwort `encoding` mit dem westeuropäischen Zeichensatz ISO-8859-1 ein.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GRUSS>Ich grüße die Welt mit meinem ersten XML-
Dokument.</GRUSS>
```

Ausgabe im Internet Explorer

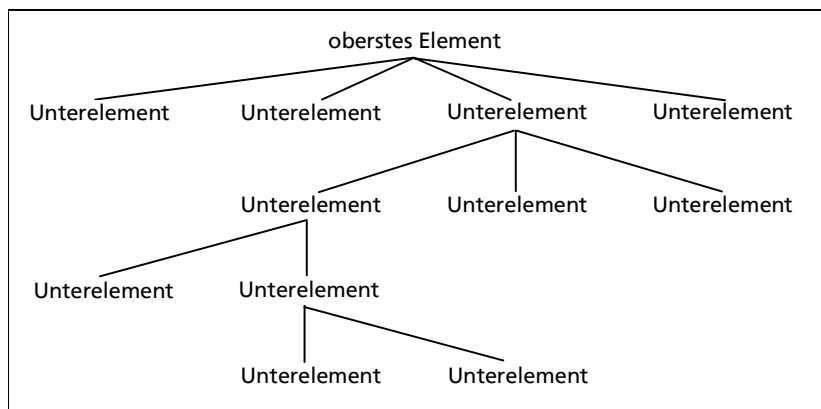
Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft.
Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```
<GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
```

Ausgabe im Mozilla Firefox

3.3 Anlegen von XML-Elementen

Die Elemente eines XML-Dokuments beschreiben die Struktur eines Dokuments. Sie sind nicht für die Ausgabe am Bildschirm zuständig.



Baumstruktur eines XML-Dokuments

Das oberste Element verzweigt sich in weitere Unterelemente. Diese können sich wiederum in mehrere Unterelemente unterteilen. Da diese Aufteilung der Struktur eines Baumes (Stamm, Ast, Zweig, Blatt) ähnelt, wird sie auch als **Baumstruktur** bezeichnet.

Elemente mit einem Inhalt

Alle Elemente bestehen aus einem Anfangs- und einem Ende-Tag. Das **Anfangs-Tag** enthält zwischen den spitzen Klammern den Namen des Elements und die optionalen Attribute. Das **Ende-Tag** besteht neben den spitzen Klammern aus dem Elementnamen, dem ein Schrägstrich `/` vorangestellt wird. Hat ein Element keinen Inhalt, können Anfangs- und Ende-Tag in Kurzschreibweise zusammengefasst werden.

Die Elementnamen dürfen Sonderzeichen enthalten, wenn der entsprechende Zeichensatz in der XML-Processing Instruction vereinbart ist.

Syntax

```
<xml-element>Inhalt des Elements</xml-element>
```

Beispiel: *kap03\elemente-start.xml*

In dem bisherigen XML-Beispiel haben Sie bereits ein Element in das Dokument eingefügt.

```
<GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
```

Der angegebene Text wird somit als **Inhalt** des Elements `GRUSS` gekennzeichnet und kann im weiteren Verlauf über diese Bezeichnung angesprochen und verarbeitet werden.

- Erweitern Sie das Beispiel um den Namen des Autors.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
<AUTOR>Heiko Schröder</AUTOR>
```

- Öffnen Sie die XML-Datei im Internet Explorer oder im Firefox.

Ich grüße die Welt mit meinem ersten XML-Dokument.

Fehlerhafte Ausgabe im Internet Explorer

XML-Verarbeitungsfehler: "Junk" nach Dokument-Element
Adresse: file:///C:/uebung/xml11_bu/beispiele/kap03/elemente-start.xml
Zeile Nr. 3, Spalte 1:

```
<AUTOR>Heiko Schröder</AUTOR>
```

Fehlermeldung im Mozilla Firefox

Die Browser zeigen jeweils einen Fehler an, da nur ein Element der obersten Ebene zugelassen ist.

XML verlangt in der Dokumentstruktur eine eindeutige Kennzeichnung des obersten Hauptelements. Erst dann können weitere Unterelemente eingefügt werden. Im Beispiel befinden sich die Elemente GRUSS und AUTOR in der gleichen Hierarchie. Die Lösung des Problems besteht darin, ein Hauptelement zu benennen, das alle anderen Elemente umschließt.

Beispiel: *kap03elemente.xml*

```

① <?xml version="1.0" encoding="ISO-8859-1"?>
② <DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  <AUTOR>Heiko Schröder</AUTOR>
③ </DOKUMENT>

```

- ① Das Hauptelement mit der Bezeichnung DOKUMENT wird festgelegt.
- ② Die Elemente GRUSS und AUTOR befinden sich innerhalb des Hauptelements DOKUMENT.
- ③ Das Hauptelement DOKUMENT wird geschlossen.

► Übernehmen Sie das Beispiel und öffnen Sie es im Browser.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
- <DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-
    Dokument.</GRUSS>
  <AUTOR>Heiko Schröder</AUTOR>
</DOKUMENT>

```

Anzeige im Internet Explorer

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft.
 Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```

- <DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  <AUTOR>Heiko Schröder</AUTOR>
</DOKUMENT>

```

Anzeige im Mozilla Firefox

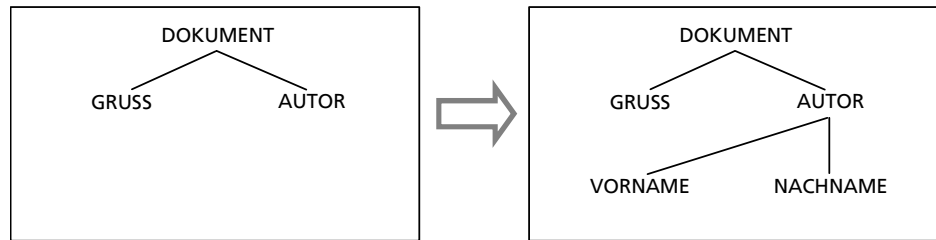
Weitere Unterelemente hinzufügen

Die Grundstruktur eines XML-Dokuments darf nur aus einem Hauptelement bestehen. Darunter können Sie dann mehrere Unterelemente einfügen. Auch diese können weitere Unterelemente enthalten, die wiederum auch Unterelemente enthalten können usw.

Es ist sinnvoll, XML-Daten bis zum kleinstmöglichen Element aufzuteilen. Diese Daten können später besser ermittelt und verarbeitet werden.



Die bisherige Struktur des Beispiels soll weiter untergliedert werden.



Beispiel: *kap03\subelemente.xml*

```

① <?xml version="1.0" encoding="ISO-8859-1"?>
  ② <DOKUMENT>
    ③ <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
    ④ <AUTOR>
      ⑤ <VORNAME>Heiko</VORNAME>
      <NACHNAME>Schröder</NACHNAME>
    </AUTOR>
  </DOKUMENT>
  
```

- ① Das Element `DOKUMENT` ist das Hauptelement und umschließt alle weiteren Unterelemente.
- ② Die bisherige Angabe des Autors wird weiter untergliedert. Die nachfolgenden Elemente sollen den Autor näher spezifizieren.
- ③ Der Vorname des Autors wird als Unterelement des Elements `AUTOR` definiert.
- ④ Der Nachname des Autors wird angegeben.
- ⑤ Das Element `AUTOR` wird mit dem Ende-Tag `</AUTOR>` abgeschlossen.

Achten Sie darauf, dass das schließende Tag des jeweiligen Elements entsprechend der Hierarchie gesetzt wird. Dies bedeutet, Sie dürfen die einzelnen Elemente nicht ineinander verschachteln, sondern müssen die jeweilige Elementstruktur beachten. Folgende Angabe wäre beispielsweise fehlerhaft:

```

<AUTOR>      <VORNAME>      <NACHNAME>      </VORNAME>      </NACHNAME>      </AUTOR>
  ①            ②            ③            ②            ③            ①
  
```

Richtig hingegen ist folgende Reihenfolge:

```

<AUTOR>      <VORNAME>      </VORNAME>      <NACHNAME>      </NACHNAME>      </AUTOR>
  ①            ②            ②            ③            ③            ①
  
```

Elemente ohne Inhalt

In HTML enthalten einige Elemente keinen Inhalt, beispielsweise das Tag `img` oder das Element `br`. Auch XML erlaubt die Angabe eines Elements ohne Inhalt.

Syntax

```
<xml-element></xml-element>
```

- Fügen Sie in Ihr Beispiel ein leeres Element ein und öffnen Sie die Datei in Ihrem Browser.

Beispiel: *kap03\ohneinhalt.xml*

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  <AUTOR>
    <VORNAME>Heiko</VORNAME>
    <NACHNAME>Schröder</NACHNAME>
  
```

①	<pre> <FOTO></FOTO> </AUTOR> </DOKUMENT> </pre>
---	---

```

<?xml version="1.0" encoding="ISO-8859-1"?>
- <DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-
    Dokument.</GRUSS>
  - <AUTOR>
    <VORNAME>Heiko</VORNAME>
    <NACHNAME>Schröder</NACHNAME>
    <FOTO/> ①
  </AUTOR>
</DOKUMENT>

```

Element-Kurzform im Internet Explorer

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft.
Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```

- <DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  - <AUTOR>
    <VORNAME>Heiko</VORNAME>
    <NACHNAME>Schröder</NACHNAME>
    <FOTO/> ①
  </AUTOR>
</DOKUMENT>

```

Element-Kurzform im Mozilla Firefox

Die Browser laden die XML-Datei, ohne eine Fehlermeldung auszugeben. Sie stellen die Elemente jedoch anders dar als bisher gewohnt ①. Sie fügen die beiden angegebenen Tags zu einem Element in Kurzform zusammen. Diese Darstellung ist völlig korrekt, da in der XML-Syntax die Angabe eines leeren Elements in Kurzform möglich ist. Die Browser optimieren die Ausgabe eines Elements ohne Inhalt automatisch.

Syntax

```
<xml-element />
```

Sie können die Kurzschreibweise für ein leeres Element verwenden. Dabei wird vor das Zeichen `>` ein Schrägstrich `/` geschrieben. Zusätzlich können Sie vor dem Schrägstrich ein Leerzeichen einfügen. Im Gegensatz zu XHTML ist das Leerzeichen in XML nicht zwingend erforderlich.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  <AUTOR>
    <VORNAME>Heiko</VORNAME>
    <NACHNAME>Schröder</NACHNAME>
    <FOTO />
  </AUTOR>
</DOKUMENT>

```

3.4 Attribute eines Elements

Mit einem Attribut haben Sie die Möglichkeit, ein Element genauer zu spezifizieren oder bestimmte Wertvorgaben zuzuweisen. In HTML können Sie beispielsweise einen Absatz über die Angabe von `<p align="center">` zentriert ausrichten.

Sie sollten dabei beachten, dass Attribute innerhalb des Elements nur einmal angegeben werden können, Unterelemente hingegen beliebig oft. Die Angabe der Reihenfolge der Attribute eines Elements ist nicht fest fixiert und kann flexibel gehandhabt werden. Attribute werden z. B. eingesetzt, wenn bestimmte Werte vorgegeben werden sollen und keine eigenen Werte angegeben werden dürfen.

Syntax

```
<xml-element AttributName1="Wert1" AttributName2="Wert2">Inhalt</xml-element>
```

Die Anzahl der festgelegten Attribute ist theoretisch unbegrenzt, jedoch darf ein bestimmtes Attribut innerhalb eines Elements nur **einmal** verwendet werden. Die Werte werden in einfachen oder doppelten Anführungszeichen angegeben.

Beispiel: *kap03\attribute.xml*

Dem XML-Element FOTO werden verschiedene Attribute zugeordnet.

① ②	<pre> <?xml version="1.0" encoding="ISO-8859-1"?> <DOKUMENT> <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS> <AUTOR> <VORNAME>Heiko</VORNAME> <NACHNAME>Schröder</NACHNAME> ① <FOTO quelle="bild.jpg" /> ② <FOTO quelle="bild.jpg" hoehe="200" breite="100" /> </AUTOR> </DOKUMENT> </pre>
--------	--

- ① Das Element FOTO besitzt ein Attribut, mit dem der Name der Grafikdatei bestimmt wird.
- ② Das Element FOTO besitzt drei verschiedene Attribute, das Attribut *quelle* zur Angabe der Grafikdatei sowie die Attribute *hoehe* und *breite*, mit denen die Höhe und die Breite der Grafik festgelegt werden.

Attributnamen

Die Attribute können Sie selbst definieren – wie die Elementnamen. Welche Attribute angegeben werden müssen und welchen Wert ein Attribut besitzen darf, legen Sie in der Dokumenttyp-Definition (DTD) bzw. dem XML-Schema fest, auf die in den nächsten Kapiteln näher eingegangen wird.

Neben den selbst zu definierenden Attributnamen sind in der XML-Spezifikation zwei Attribute festgelegt, welche für jedes Element genutzt werden können:

- ✓ `xml:lang` kennzeichnet die verwendete Sprache des Inhaltes eines Elements
- ✓ `xml:space` bestimmt den Umgang mit Leerraum im Element

Als Werte zur Festlegung der Sprache werden die in der ISO-Norm ISO 639 festgelegten zwei- bzw. dreistelligen Ländercodes verwendet, beispielsweise

```
<Element xml:lang="en"> .. Inhalt .. </Element>
```

Die Leerraumbehandlung steuert den Umgang des XML-Prozessors mit Leerzeichen, Tabulatoren und Leerzeilen im Inhalt eines Elements. Die Vorgaben werden jedoch nicht von allen Anwendungen vollständig beachtet. Die Wirkung des Attributs ist deshalb unterschiedlich. Mögliche Werte sind:

- ✓ `preserve` alle Leerräume sollen so, wie im Inhalt vorhanden verarbeitet werden
- ✓ `default` der Umgang mit den Leeräumen wird der jeweiligen Anwendung überlassen

3.5 Kommentare hinzufügen

In XML können Sie, wie in jeder Programmier- und Skriptsprache, Kommentare einfügen, um eigene Notizen im Quelltext zu hinterlegen. Quelltexte sollten Sie immer so kommentieren, dass Sie auch nach längerer Zeit den Quelltext nachvollziehen können und dass der Quelltext auch jederzeit von einer anderen Person verstanden und gepflegt werden kann. Kommentare werden vom verwendeten Programm/Parser/Browser nicht berücksichtigt.

<pre><!-- Kommentar --></pre>

Einen Kommentar leiten Sie mit der Zeichenkette `<!--` ein. Danach folgt die Erläuterung, die in das Dokument eingefügt werden soll. Einen Kommentar schließen Sie mit der Zeichenkette `-->` ab.

Beispiel: kap03\kommentar.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<DOKUMENT>
  <GRUSS>Ich grüße die Welt mit meinem ersten XML-Dokument.</GRUSS>
  <AUTOR>
    <VORNAME>Heiko</VORNAME>
    <NACHNAME>Schröder</NACHNAME>
    <!-- Nachfolgend erscheint ein Foto von mir -->
    <FOTO quelle="bild.jpg" hoehe="200" breite="100" />
  </AUTOR>
</DOKUMENT>
```

Bei Kommentaren ist zu beachten, dass

- ✓ sie nicht vor der Processing Instruction zur XML-Deklaration stehen dürfen,
- ✓ keine Kommentare innerhalb von Kommentaren erlaubt sind,
- ✓ sie nicht innerhalb von Tags oder Processing Instruction stehen dürfen.

3.6 Wohlgeformtheit eines XML-Dokuments

Der Begriff „Wohlgeformtheit“ wird Ihnen in Bezug auf XML des Öfteren begegnen. Ein wohlgeformtes Dokument, im Englischen „well formed document“, bedeutet:

- ✓ Ein XML-Dokument besteht aus dem Prolog und mindestens einem Element.
- ✓ Es gibt nur ein Hauptelement (Wurzelement oder auch root element).
- ✓ Alle Elemente sind richtig verschachtelt. Sie dürfen sich nicht überlappen.
- ✓ Alle Attributwerte stehen in einfachen oder doppelten Anführungszeichen.
- ✓ Ein Element darf nicht zwei Attribute mit demselben Namen besitzen.
- ✓ Kommentare dürfen nicht direkt in den Elementen eingefügt werden.
- ✓ Reservierte Zeichen wie `<` oder `>` sind in der speziellen Form `<` bzw. `>` anzugeben.

Die aktuellen Browser testen bei der Anzeige eines Dokuments dessen Wohlgeformtheit. Enthält das Dokument Fehler, zeigt Ihnen dies der Browser sofort an. Wird Ihr Dokument also im Browser ohne eine Fehlermeldung angezeigt, haben Sie für den Prozessor des Browsers ein wohlgeformtes XML-Dokument erstellt.



3.7 Ein XML-Dokument erstellen

Nachdem Sie die grundlegenden Regeln zum Erstellen eines einfachen XML-Dokuments erlernt haben, sind Sie in der Lage, die folgende Aufgabe in einem XML-Dokument zu strukturieren.

Sie besitzen zu Hause beispielsweise eine Reihe von Musik-CDs, Schallplatten und MP3-Dateien. Zur besseren Übersicht Ihrer Musiksammlung können Sie diverse Verwaltungsprogramme nutzen. Durch die Verwaltung der Musikalben über XML können Sie diese Daten flexibel nutzen.

Grobstruktur

Als Erstes muss das Hauptelement des XML-Dokuments benannt werden. Die Bezeichnung des Elements sollte so gewählt werden, dass sie auch Aufschluss über dessen Inhalt gibt. Beispielsweise könnte der Name des Hauptelements `MUSIKSAMMLUNG` lauten.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
  <ALBUM>
  </ALBUM>
</MUSIKSAMMLUNG>
```

Grobstruktur mit Attribut

Da für Musik die verschiedensten Tonträger und -formate existieren (Compact Discs, Schallplatten, MP3-Dateien etc.), sollten diese in der Katalogisierung berücksichtigt werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
  <ALBUM typ="CD">
  </ALBUM>
  <ALBUM typ="Vinyl">
  </ALBUM>
  <ALBUM typ="MP3">
  </ALBUM>
</MUSIKSAMMLUNG>
```

Feinstruktur

Als Nächstes folgt die weitere Unterteilung der verschiedenen Musikmedien. Ein Album lässt sich eindeutig über den Namen des Interpreten und den Titel des Albums identifizieren. Zusätzlich sollen die Lieder des jeweiligen Albums in der XML-Datei gespeichert werden. Demzufolge wird das Element `ALBUM` erweitert, indem die Unterelemente `INTERPRET`, `TITEL` und `LIED` eingefügt werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
  <ALBUM typ="CD">
    <INTERPRET></INTERPRET>
    <TITEL></TITEL>
    <LIED></LIED>
  </ALBUM>
  <ALBUM typ="Vinyl">
    <INTERPRET></INTERPRET>
    <TITEL></TITEL>
    <LIED></LIED>
  </ALBUM>
  <ALBUM typ="MP3">
    <INTERPRET></INTERPRET>
    <TITEL></TITEL>
    <LIED></LIED>
  </ALBUM>
</MUSIKSAMMLUNG>
```

Die Struktur für die Verwaltung Ihrer Musiksammlung ist somit abgeschlossen. Die Daten können nun in das XML-Dokument eingefügt werden, in dem sie jeweils in das entsprechende Element eingefügt werden.

Beispiel: *kap03\musiksammlung.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
  <ALBUM typ="CD">
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
    <LIED>Honey</LIED>
    <LIED>Find my baby</LIED>
    <LIED>Porcelain</LIED>
    <LIED>Why does my heart feel so bad?</LIED>
    <LIED>South side</LIED>
    <LIED>Rushing</LIED>
    <LIED>Bodyrock</LIED>
    <LIED>Natural Blues</LIED>
    <LIED>Machete</LIED>
  </ALBUM>
</MUSIKSAMMLUNG>
```



```

<LIED>...</LIED>
</ALBUM>
<ALBUM typ="Vinyl">
  <INTERPRET>a-ha</INTERPRET>
  <TITEL>Minor earth major sky</TITEL>
  <LIED>Minor earth major sky</LIED>
  <LIED>Little black</LIED>
  <LIED>...</LIED>
</ALBUM>
<ALBUM typ="MP3">
  <INTERPRET>Mesh</INTERPRET>
  <TITEL>Fragmente</TITEL>
  <LIED>Trust you</LIED>
  <LIED>My defender</LIED>
  <LIED>...</LIED>
</ALBUM>
</MUSIKSAMMLUNG>

```

Möchten Sie die XML-Struktur noch weiter verfeinern, fügen Sie der Hierarchie entsprechend in den bestehenden Elementen weitere Unterelemente ein. Beispielsweise könnten Sie spezielle Informationen zum Herausgeber, zum Erscheinungsjahr, zur Gesamtspielzeit, zur Spielzeit eines Titels und eigene Anmerkungen hinterlegen.

Beispiel: *kap03\musiksammlung_erw.xml*

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
  <ALBUM typ="CD">
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
    <GESAMTZEIT>63:03</GESAMTZEIT>
    <LABEL>Mute Records Limited</LABEL>
    <JAHR>1999</JAHR>
    <LIED>
      <LIEDTITEL>Honey</LIEDTITEL>
      <TITELZEIT>3:28</TITELZEIT>
    </LIED>
    <LIED>
      <LIEDTITEL>Find my baby</LIEDTITEL>
      <TITELZEIT>3:59</TITELZEIT>
    </LIED>
    <LIED>
      <LIEDTITEL>Porcelain</LIEDTITEL>
      <TITELZEIT>4:01</TITELZEIT>
    </LIED>
    <LIED>...</LIED>
  </ALBUM>
</MUSIKSAMMLUNG>

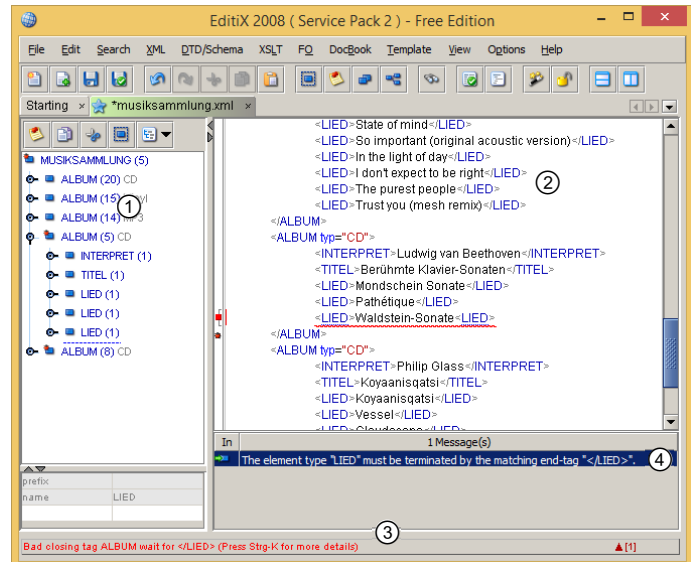
```

3.8 XML-Editor Editix

Mittlerweile gibt es, wie für jede Sprache, auch Programme, die Sie bei der Erstellung von XML-Dokumenten unterstützen. Ein für nicht kommerzielle Zwecke kostenlos verwendbares Programm zum Erfassen von XML-Elementen ist Editix 2008. Herunterladen können Sie dieses englischsprachige Programm über die Webadresse <http://free.editix.com>. Neben der freien gibt es eine lizenzierte Version, mit einem erweiterten Funktionsumfang.

Editix ist eine Anwendung, die das Erstellen eines XML-Dokuments visuell unterstützt. Es erlaubt Ihnen die Eingabe von Daten für den Aufbau einer XML-Struktur. Das Programm verwaltet die Standardelemente eines XML-Dokuments, wie Elemente und Attribute sowie die notwendigen Daten. Eine visuelle Unterstützung während der Dateneingabe bietet die farbige Hervorhebung von Elementen, Attributen und Werten. Ebenso kann die Baumansicht eingeblendet werden, die Ihnen die XML-Struktur mit allen Elementen übersichtlich anzeigt. Zusätzlich erhalten Sie Informationen über den korrekten Aufbau (Wohlgeformtheit).

- ① In dieser Baumansicht werden die einzelnen Elemente des Dokuments übersichtlich dargestellt. Bei der Auswahl eines Elements werden darunter die Attribute aufgelistet.
- ② Die Eingabe der XML-Elemente und Daten erfolgt in diesem Bereich. Die Syntax wird farblich hervorgehoben, dies verhilft zu einer besseren Übersicht.
- ③ Treten während der Eingabe Fehler auf, werden entsprechende Meldungen in der Statusleiste angezeigt.
- ④ Eventuell auftretende Fehler beim Validieren des XML-Dokuments werden hier aufgelistet. Mit einem Doppelklick kann sofort zu dem fehlerhaften Element gesprungen werden.



Zur näheren Erläuterung der Bedienung des Programms lesen Sie die englischsprachige Windows-Hilfe, die dem Programm beigelegt ist.

3.9 Schnellübersicht

Ein wohlgeformtes XML-Dokument beinhaltet ...

- ✓ einen Prolog mit der Angabe der Version und des verwendeten Zeichensatzes
- ✓ genau ein Hauptelement

Außerdem gilt:

- ✓ Alle angegebenen Elemente müssen ein Anfangs- und Ende-Tag besitzen, auch wenn es sich um ein leeres Element handelt.
- ✓ Das schließende Tag des Elements muss entsprechend der Hierarchie gesetzt werden.
- ✓ Der Name des Elements ist frei wählbar.
- ✓ Die Groß- und Kleinschreibung der Elemente ist zu beachten.
- ✓ Jedes einzelne Element kann weitere Unterelemente enthalten.
- ✓ Die Elementnamen dürfen auch Sonderzeichen enthalten, wenn der entsprechende Zeichensatz vereinbart ist.
- ✓ Angegebene Attributwerte müssen immer in Anführungszeichen gesetzt werden.

3.10 Übungen

Übung 1: Theoriefragen zum Aufbau eines XML-Dokuments

Übungsdatei: --

Ergebnisdatei: *kap03\uebung1-2.html*

- ① Beschreiben Sie, wie das Gerüst eines XML-Dokuments aufgebaut ist und welche Angaben für die Wohlgeformtheit notwendig sind.
- ② Erklären Sie, was unter Attributen zu verstehen ist und wie viele Attribute pro Element verwendet werden können.

Übung 2: Strukturieren von Informationen

Übungsdatei: --

Ergebnisdateien: *kap03\uebung3.html*,
kap03\uebung3.xml

- ① Sie absolvieren ein XML-Seminar mit mehreren Teilnehmern. Geben Sie die nachfolgenden Informationen in einer XML-Struktur an. Beachten Sie dabei, welche Informationen einmalig sind und welche mehrmals vorkommen.

Ziel des Seminars: Einführung in XML Datum: 02.04. - 06.04. in XML-Stadt Zeitplan: ab 09:00 Uhr Themen: Überblick XHTML XML-Struktur ab 13:00 Uhr Themen: DTD entwickeln Darstellung von XML mit XSL 17:00 Uhr - Ende	Personen: Seminarleiter Herr Hans Hauser Teilnehmer Herr Rolf Rabicht Frau Sigrun Sieghaar Frau Linda Landmann Herr Bruno Brahms
--	--

Übung 3: XML-Struktur entwickeln

Übungsdatei: --

Ergebnisdatei: *kap03\uebung4.xml*

- ① Entwickeln Sie eine XML-Struktur, mit der Sie die technischen Details von Kraftfahrzeugen verwalten können. Folgende Informationen sollen integriert werden:

	BMW	Mercedes	Audi
Modell	BMW Z8	SLK 32 AMG	1.8 T quattro
Leergewicht	1690 kg	1495 kg	1395 kg
Zylinder	8	6	4
Hubraum	4941 cm ³	3199 cm ³	1781 cm ³
Leistung	294 kW	260 kW	165 kW
Höchstgeschwindigkeit	250 km/h	250 km/h	243 km/h
von 0 auf 100 km/h	4,7 s	5,2 s	6,4 s

4 Elemente der DTD

In diesem Kapitel erfahren Sie

- ✓ wie Sie die Elemente und Inhalte der XML-Daten festlegen
- ✓ wie eine Dokumenttyp-Definition aufgebaut ist
- ✓ was unter einem gültigen Dokument zu verstehen ist
- ✓ wozu Parser notwendig sind

Voraussetzungen

- ✓ Kenntnisse im Aufbau eines XML-Dokuments

4.1 Dokumenttyp-Definition

Definition des Datenmodells einer XML-Datei

In einem XML-Dokument ist die Dokumenttyp-Definition kurz DTD ein zentraler Bestandteil. Die Dokumenttyp-Definition ist das Regelwerk, nach dem sich die Dokumente richten müssen. Sie enthält die folgenden Richtlinien:

- ✓ welche Elemente das Dokument enthalten darf,
- ✓ welchen Inhalt die Elemente haben dürfen,
- ✓ welche Attribute erlaubt sind,
- ✓ welche Werte die Attribute annehmen können,
- ✓ wie die Elemente und in welcher Reihenfolge die Elemente ineinander geschachtelt werden dürfen.

Eine Dokumenttyp-Definition selbst ist kein XML-Dokument. Für die Dokumenttyp-Definition wird eine SGML-verwandte Syntax verwendet. Durch die Verwendung einer formalen Sprache zur Beschreibung des Datenmodells einer XML-Datei ist gewährleistet, dass die Beschreibung selbst von Programmen gelesen und ausgewertet werden kann. Damit kann die sogenannte Validierung einer XML-Datei, ihre Überprüfung unter Berücksichtigung des für sie festgelegten Datenmodells, automatisch durch Programme erfolgen.

In Anlehnung an die objektorientierte Programmierung wird im Verhältnis von DTD und XML-Datei auch die Begrifflichkeit der DTD als Definition einer Klasse von Dokumenten und den einzelnen XML-Dateien als Dokumentinstanzen verwendet.



Anstelle einer DTD kann auch ein XML-Schema zur Definition des Datenmodells einer XML-Datei verwendet werden (vgl. Kapitel 7 und 8).

Angabe der DTD in HTML

HTML benötigt zur Validierung eines HTML-Dokuments und zur Spezifikation der möglichen Elemente und Attribute eine DTD. Deren Einbindung ist für ein HTML-Dokument jedoch nicht zwingend notwendig, da die Browser diese Definition bereits kennen und intern umsetzen können.

Folgendermaßen können Sie im HTML-Code angeben, welche HTML-Version Ihr Dokument unterstützt.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Diese Zeile verweist auf eine Dokumenttyp-Definition für den HTML-Standard 4.0. Die Anweisung `DOCTYPE HTML PUBLIC` bedeutet, dass sich der Inhalt einer Webseite auf die öffentlich verfügbare HTML-DTD bezieht. Die Angaben zwischen den Anführungszeichen geben Auskunft über den Herausgeber der DTD, das W3-Konsortium, den SGML-Dokumenttyp `HTML` in der Sprachversion 4.0 sowie die Sprache, in der die Elemente definiert wurden.

Ab HTML5 ist auch die entsprechende Kurzangabe möglich:

```
<!DOCTYPE html">
```

4.2 Definition einer internen DTD

Eine DTD wird innerhalb eines XML-Dokuments über die Syntax

```
<!DOCTYPE name [  
  <!-- Element-Definitionen -->  


```

direkt nach der Versionsangabe im XML-Prolog `<?xml version="1.0"?>` und vor dem eigentlichen Beginn des XML-Elements definiert. Eingeleitet wird die DTD mit dem Schlüsselwort `<!DOCTYPE ...>`. Die Bezeichnung `name` ist hierbei eine eindeutige Bezeichnung für die nachfolgende Dokumenttyp-Definition. Innerhalb der eckigen Klammern `[]` werden die möglichen Elemente definiert, die im XML-Dokument vorkommen dürfen.

Die Norm erwartet bei der Namensangabe den Namen des obersten Elements. Daher muss der Name der DTD so benannt werden wie das Hauptelement der XML-Datei.



Beispiel: *kap04\doctype.xml*

```
① <?xml version="1.0" encoding="ISO-8859-1"?>  
② <!DOCTYPE MUSIKSAMMLUNG [  
  <!-- Definitionen der einzelnen Elemente -->  
③ ]>  
④ <MUSIKSAMMLUNG>  
  <!-- Die einzelnen Elemente mit den Daten. -->  
  </MUSIKSAMMLUNG>
```

- ① Das Dokument beginnt mit der Versionsangabe im XML-Prolog.
- ② Die Dokumenttyp-Definition mit dem Namen `MUSIKSAMMLUNG` wird eingeleitet.
- ③ Die DTD wird geschlossen und beendet.
- ④ Es beginnt das Hauptelement der XML-Datei mit den verschiedenen Unterelementen.

Verwenden Sie eine interne Dokumenttyp-Definition, sollten Sie die einleitende Verarbeitungsvorschrift `<?xml ...?>` um das Schlüsselwort `standalone` erweitern. Diese Option kennzeichnet, ob der Parser eine externe DTD einlesen muss oder ob die DTD innerhalb des XML-Dokuments definiert ist.



```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
```

Die Angabe von `standalone="yes"` teilt dem Parser mit, dass die Dokumenttyp-Definition innerhalb des geladenen XML-Dokuments zu finden ist. Das XML-Dokument benötigt keine weiteren externen Dateien. Die automatische Vorgabe `standalone="no"` teilt dem XML-Prozessor mit, dass noch weitere externe Dateien einzubinden sind.

4.3 Deklarieren der Elementtypen

Elementtyp-Deklarationen legen die Namen der Unterelemente fest, die innerhalb der übergeordneten Elemente möglich sind. Es wird angegeben, von welcher Art ihre Inhalte sind und welche Attribute verwendet werden dürfen.

Die Elementtyp-Deklaration legt folgende Regeln innerhalb einer Dokumenttyp-Definition fest:

- ✓ die Syntax eines Elements,
- ✓ ob bestimmte Elemente mehrfach vorkommen dürfen,
- ✓ den Typ eines Elements,
- ✓ die Reihenfolge der Elemente in dem XML-Dokument.

Zu beachten ist dabei Folgendes:

- ✓ Ein Element muss eindeutig definiert sein.
- ✓ Elementnamen sind „case-sensitive“, sodass auf Groß- und Kleinschreibung zu achten ist.
- ✓ Das zur Elementtyp-Deklaration notwendige Schlüsselwort `ELEMENT` muss immer in Großbuchstaben geschrieben werden.

4.4 Angabe der Elemente

Zur näheren Erläuterung wird die bereits erstellte Musiksammlung als Beispiel verwendet und das zusätzliche Element `AUTOR` eingefügt. In diesem Element soll der Name der Person gespeichert werden, die den Datensatz hinzugefügt hat.

```
<MUSIKSAMMLUNG>
  <ALBUM>
    <AUTOR></AUTOR>
    <INTERPRET></INTERPRET>
    <TITEL></TITEL>
    <LIED></LIED>
  </ALBUM>
</MUSIKSAMMLUNG>
```

Elementgruppen

Eine Elementgruppe ist ein XML-Element mit weiteren Unterelementen und wird in der Dokumenttyp-Definition mithilfe des Schlüsselworts

```
<!ELEMENT name (Unterelemente)>
```

angegeben. Das Schlüsselwort `<!ELEMENT` leitet die Festlegung für ein Element mit der Bezeichnung `name` ein. Innerhalb der nachfolgenden Klammern werden die weiteren Unterelemente angegeben, die sich innerhalb des Elements `name` befinden dürfen.

```
<!ELEMENT MUSIKSAMMLUNG (ALBUM)>
```

Das erste Element in dem Beispiel ist das Hauptelement `MUSIKSAMMLUNG`. Dieses enthält ein weiteres Element mit der Bezeichnung `ALBUM`. Die obige Definition besagt daher, dass sich in dem Element `MUSIKSAMMLUNG` nur das Unterelement `ALBUM` befinden darf.

Innerhalb des Elements `<ALBUM>` befinden sich weitere Unterelemente, die Sie ebenfalls definieren müssen. Folgende Möglichkeiten der Verknüpfung der einzelnen Elemente stehen Ihnen dabei zur Verfügung:

Operator	Bedeutung
[()]	Die Unterelemente werden generell in Klammern geschrieben.
[.]	Mehrere Unterelemente werden mit einem Komma (Sequenz-Operator) voneinander getrennt und geben die Reihenfolge der anzugebenden Elemente an.
[]	Der senkrechte Strich ist der Oder-Operator. Sie können eines der angegebenen Unterelemente wählen.
#PCDATA	Der Inhalt des Elements kann aus einer beliebigen Zeichenkette bestehen.

Im nachfolgenden Beispiel werden die Elemente `AUTOR`, `INTERPRET`, `TITEL` und `LIED` als Unterelemente des Elements `ALBUM` mit dem Sequenz-Operator verknüpft. Damit ist festgelegt, in welcher Reihenfolge die Elemente anzugeben sind.

```
<!ELEMENT MUSIKSAMMLUNG (ALBUM) >
<!ELEMENT ALBUM (AUTOR, INTERPRET, TITEL, LIED) >
```

Da der Interpret eines Albums sehr allgemein gehalten ist, könnten Sie statt des Elements `INTERPRET` auch angeben, ob es sich um eine Gruppe handelt. Dazu verknüpfen Sie die möglichen Elementnamen über den Oder-Operator und setzen diese in Klammern.

```
<!ELEMENT MUSIKSAMMLUNG (ALBUM) >
<!ELEMENT ALBUM (AUTOR, (INTERPRET | GRUPPE), TITEL, LIED) >
```

Durch diese Definition müssen die Elemente `AUTOR`, `TITEL`, `LIED` im Element `ALBUM` in dieser Reihenfolge angegeben werden. Aus der Gruppe `(INTERPRET | GRUPPE)` können Sie ein beliebiges Element wählen.

Einzelne Elemente

Ein einzelnes Element, das keine weiteren Unterelemente enthält, wird mit der Definition

```
<!ELEMENT name (#PCDATA) >
```

angegeben. Durch die Angabe von `#PCDATA` wird der Datentyp des Elements festgelegt, der besagt, dass der Inhalt aus beliebigen Zeichenketten bestehen darf.

Jedes als Unterelement festgelegte Element und jedes Element, das über einen Oder-Operator verknüpft ist, muss in der Dokumenttyp-Definition als Einzelelement definiert werden.



Die Definition der restlichen Elemente der Musiksammlung erfolgt auf die folgende Art und Weise:

①	<pre><!ELEMENT MUSIKSAMMLUNG (ALBUM) > <!ELEMENT ALBUM (AUTOR, (INTERPRET GRUPPE), TITEL, LIED) > <!-- Es folgen die Unterelemente von ALBUM --></pre>
②	<pre><!ELEMENT AUTOR (#PCDATA) > <!ELEMENT INTERPRET (#PCDATA) > <!ELEMENT GRUPPE (#PCDATA) > <!ELEMENT TITEL (#PCDATA) > <!ELEMENT LIED (#PCDATA) ></pre>

- ① Das Element `ALBUM` umschließt die Unterelemente `AUTOR`, `TITEL`, `LIED` und eines von `INTERPRET` oder `GRUPPE`.
- ② Jedes Element muss definiert werden und enthält die Daten einer beliebigen Zeichenfolge. Dies wird über die Angabe von `#PCDATA` kenntlich gemacht.

Genereller Aufbau einer DTD

Struktur des XML-Dokuments	Struktur der Dokumenttyp-Definition
<pre> <HAUPTELEMENT> <UNTERELEMENT> <ELEMENT1>...</ELEMENT1> <ELEMENT2>...</ELEMENT2> <ELEMENT3>...</ELEMENT3> </UNTERELEMENT> </HAUPTELEMENT> </pre>	<pre> <!DOCTYPE HAUPTELEMENT [<!ELEMENT HAUPTELEMENT (UNTERELEMENT)> <!ELEMENT UNTERELEMENT (ELEMENT1, ELEMENT2, ↵ ELEMENT3)> <!ELEMENT ELEMENT1 (#PCDATA)> <!ELEMENT ELEMENT2 (#PCDATA)> <!ELEMENT ELEMENT3 (#PCDATA)>]> </pre>

Inhalt eines Elements

Die bisherige Definition erzwingt für ein Element auch die Angabe aller anderen Unterelemente. Dies bedeutet, dass Sie für jeden Interpreten den Titel des Albums angeben müssen, aber nur ein Lied benennen dürfen. Jedoch besteht ein Musikalbum nicht nur aus einem Lied, sondern aus mehreren Liedern. Ebenso könnte es sein, dass Sie kein Lied eines Albums mit Namen kennen und dies deshalb erst später nachtragen möchten.

Für die Daten eines Albums gelten in der Musiksammlung die folgenden Regeln:

- ✓ Ein Album ist von **mindestens einem** Interpreten.
- ✓ Das Album besitzt **einen** Titel.
- ✓ Zu einem Album gehört eine vorher nicht bekannte Anzahl von Liedern, d. h. eine **beliebige** Anzahl.

Ob ein Element einen Inhalt besitzen muss oder ob es mehrmals innerhalb des übergeordneten Elements vorkommen kann, bestimmen Sie mithilfe der nachfolgenden Zeichen zur Angabe der Häufigkeit.

Zeichen	Bedeutung
[?]	<i>0 oder 1</i> Wird ein Fragezeichen hinter einer Elementbezeichnung oder einem Klammerausdruck angegeben, darf das Element weggelassen oder nur einmal angegeben werden.
[*]	<i>beliebig oft, auch 0-mal</i> Eine mit einem Stern gekennzeichnete Elementbezeichnung erlaubt eine beliebige Anzahl der Verwendung des Elements. Es darf auch weggelassen werden.
[+]	<i>beliebig oft, aber mindestens einmal</i> Sie können beliebig viele Elemente einfügen. Jedoch darf dieses Element nicht weggelassen werden.
	<i>einmal</i> Geben Sie kein Zeichen an, muss das Element genau einmal innerhalb des übergeordneten Elements verwendet werden.

Beispiel: *kap04\musiksammlung-element.xml*

Anhand eines Beispiels soll Ihnen der Einsatz dieser Zeichen nähergebracht werden. Dazu wird die Musiksammlung erweitert, sodass Sie beispielsweise beliebig viele Alben verwalten und für jedes Album beliebig viele Lieder (auch 0) angeben können.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?></code>
	<code><!DOCTYPE MUSIKSAMMLUNG [</code>
①	<code><!ELEMENT MUSIKSAMMLUNG (ALBUM+)></code>
②	<code><!ELEMENT ALBUM (AUTOR,</code>
③	<code>(INTERPRET GRUPPE),</code>
④	<code>TITEL,</code>
⑤	<code>LIED*)></code>
	<code><!ELEMENT AUTOR (#PCDATA)></code>
	<code><!ELEMENT INTERPRET (#PCDATA)></code>
	<code><!ELEMENT GRUPPE (#PCDATA)></code>
	<code><!ELEMENT TITEL (#PCDATA)></code>
	<code><!ELEMENT LIED (#PCDATA)></code>
	<code>]></code>

- ① In einer Musiksammlung möchten Sie mehr als nur ein Album speichern. Markieren Sie dazu das Element `ALBUM` mit dem Zeichen `+`. Damit stellen Sie sicher, dass innerhalb des XML-Dokuments das Element mehrfach vorkommen darf, aber mindestens einmal angegeben werden muss.
- ② Das Element `AUTOR` muss angegeben werden, um nachzuvollziehen, wer den Datensatz eingegeben hat, und erhält deshalb kein Zeichen zur Angabe der Häufigkeit.
- ③ Das Element `INTERPRET` oder `GRUPPE` kann innerhalb des übergeordneten Elements `ALBUM` nur einmal gesetzt werden.
- ④ Das Element `TITEL` muss angegeben werden, um die CD, Schallplatte, MP3-Datei oder Musikkassette eindeutig zu identifizieren.
- ⑤ Da die Anzahl der Lieder auf einem Album nicht von vornherein bekannt ist, wird das Element so festgelegt, dass es gar nicht oder beliebig oft angegeben werden darf.

Sollen in der Musiksammlung alle Elemente gleich oft vorkommen, müssen Sie nicht jedes einzelne Element mit dem entsprechenden Zeichen versehen. In diesem Fall geben Sie das Zeichen außerhalb der Elemente an.

①	<code><!ELEMENT ALBUM (AUTOR, (INTERPRET GRUPPE), TITEL, LIED)*></code>
	oder
②	<code><!ELEMENT ALBUM (AUTOR, (INTERPRET GRUPPE), TITEL, LIED)+></code>

- ① Die Unterelemente des Hauptelements `ALBUM` müssen nicht zwingend angegeben werden.
- ② Die Elemente `AUTOR`, `TITEL`, `LIED` und eines der Elemente `INTERPRET`, `GRUPPE` müssen mindestens einmal im Dokument erscheinen.

Inhaltsmodelle mit `EMPTY` und `ANY`

Bisher haben Sie Elemente definiert, die als Inhalt eine beliebige Zeichenfolge besitzen können (`#PCDATA`). Es kann jedoch auch Elemente ohne einen Inhalt geben. Derartige Elemente benötigen kein Ende-Tag und werden auch als „leere Elemente“ bezeichnet. In der Auszeichnungssprache HTML sind dies beispielsweise `br` für den Zeilenumbruch oder `hr` für die horizontale Linie.

In der DTD eines XML-Dokuments werden die Elemente ohne einen Inhalt mit dem Schlüsselwort `EMPTY` definiert.

<code><!ELEMENT name EMPTY></code>
--

Zusätzlich können Sie XML-Elemente mit dem Schlüsselwort `ANY` deklarieren.

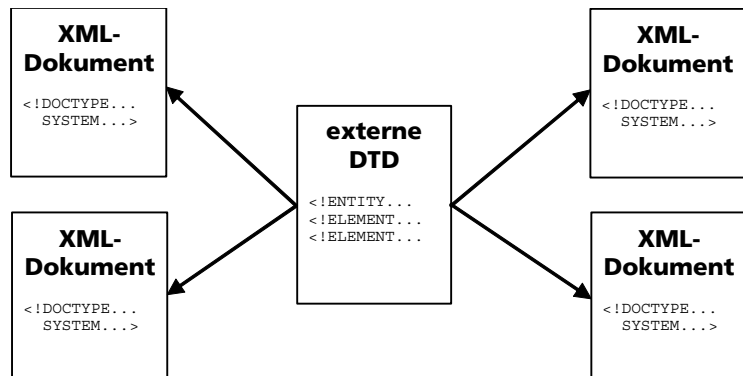
<code><!ELEMENT name ANY></code>
--

Ein Element mit diesem Inhaltsmodell kann jedes andere Element der DTD beinhalten, auch sich selbst. Es wird jedoch selten angewandt, weil keine Einschränkungen zum Inhalt gemacht werden.

4.5 Externe Teilmenge der DTD

Möchten Sie eine Dokumenttyp-Definition für mehrere XML-Dokumente verwenden, empfiehlt es sich, diese als externe Datei auszulagern. Statt die Definition des Dokumentes immer wieder in allen Dokumenten festzulegen, können Sie eine zentrale Datei anlegen und von jeder einzelnen XML-Datei darauf zugreifen. Diese externe DTD enthält damit die Definitionen für alle XML-Dateien, die darauf zugreifen.

Ändern Sie die Definition eines Elements in der externen DTD, wirkt sich diese Änderung in allen anderen XML-Dokumenten aus, in denen die DTD eingebunden ist.



Vollständige externe Teilmenge

Eine externe Dokumenttyp-Definition ist fast identisch mit einer internen Dokumenttyp-Definition. Der Unterschied besteht darin, dass in einer externen Teilmenge der DTD die Angabe der Verarbeitungsvorschrift `<!DOCTYPE name []>` fehlt. Diese Angabe wird innerhalb des XML-Dokuments eingefügt und darf nicht in die externe Teilmenge geschrieben werden.

Das Einbinden einer externen Teilmenge in ein XML-Dokument erfolgt über die modifizierte Verarbeitungsvorschrift `<!DOCTYPE ...>`.

```
<!DOCTYPE name SYSTEM "URI">
```

Durch das Schlüsselwort **SYSTEM** wird der entsprechenden Anwendung mitgeteilt, dass eine externe Dokumenttyp-Definition zu verwenden ist. Der Parameter **URI** (Uniform Resource Identifier) ist durch den Dateinamen oder die URL der DTD-Datei zu ersetzen und immer in Anführungszeichen anzugeben.



Ein **Uniform Resource Identifier (URI)** ist eine Zeichenkette, die eine Internetressource darstellt. Der meist-verwendete Parameter ist der **Uniform Resource Locator (URL)**, der eine Internetadresse bezeichnet.

```

<!-- DTD im selben Ordner -->
① <!DOCTYPE name SYSTEM "extern.dtd">
<!-- DTD in einem anderen Ordner -->
② <!DOCTYPE name SYSTEM "../dtd/extern.dtd">
<!-- DTD von einem anderen Server -->
③ <!DOCTYPE name SYSTEM "http://www.example.com/dtd/extern.dtd">
  
```

- ① Wenn die DTD-Datei im selben Ordner gespeichert ist wie die XML-Datei, genügt die Angabe des Dateinamens.
- ② Befindet sich die DTD in einem anderen Ordner, müssen Sie den Pfad angeben.
- ③ Wird auf eine DTD auf einem anderen Server verwiesen, ist die gesamte URL anzugeben.



Verwenden Sie eine externe Dokumenttyp-Definition, dann können Sie in den darauf zugreifenden XML-Dokumenten den einleitenden XML-Befehl `<?xml ...?>` um die Angabe `standalone="no"` erweitern.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
```

Die Angabe von `standalone="no"` teilt der Anwendung mit, dass für die Dokumenttyp-Definition eine oder mehrere externe Dateien geladen werden müssen. Da dies eine standardmäßige Einstellung ist, muss sie nicht unbedingt angegeben werden.

Beispiel: *kap04|extern.dtd*

Lagern Sie die folgende Dokumenttyp-Definition in eine externe Datei aus und speichern Sie die Datei unter dem Namen *extern.dtd*.

```
<!ELEMENT MUSIKSAMMLUNG (ALBUM+)>
<!ELEMENT ALBUM (AUTOR, (INTERPRET | GRUPPE)+, TITEL, LIED*)>
  <!ELEMENT AUTOR      (#PCDATA)>
  <!ELEMENT INTERPRET  (#PCDATA)>
  <!ELEMENT GRUPPE     (#PCDATA)>
  <!ELEMENT TITEL      (#PCDATA)>
  <!ELEMENT LIED       (#PCDATA)>
```

Beispiel: *kap04|musiksammlung-extdtd.xml*

Ändern Sie die Angabe `<!DOCTYPE ...>`, sodass die Dokumenttyp-Definition aus der externen Datei *extern.dtd* geladen wird.

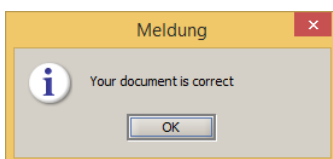
```
① <?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
② <!DOCTYPE MUSIKSAMMLUNG SYSTEM "extern.dtd">
③ <MUSIKSAMMLUNG>
    <ALBUM>...</ALBUM>
</MUSIKSAMMLUNG>
```

- ① Der XML-Prolog wird um die Angabe `standalone="no"` erweitert, um das verarbeitende Programm zu informieren, dass externe Dokumenttyp-Definitionen geladen werden müssen.
- ② Mit dieser Angabe binden Sie die Datei *extern.dtd* für die Dokumenttyp-Definition der nachfolgenden Elemente ein.
- ③ Ab hier folgen die XML-Elemente.

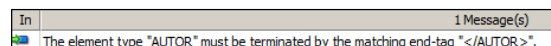
- ▶ Speichern Sie das XML-Dokument unter dem Namen *musiksammlung-extdtd.xml* in demselben Ordner wie die zuvor extern angelegte Datei *extern.dtd*.
- ▶ Laden Sie die Datei in das Programm Editix.
- ▶ Wählen Sie das Menü *XML - Check for a well-formed / valid document*.

Alternative: **Strg** **K**

Die Software testet das XML-Dokument anhand der vorgegebenen DTD auf Gültigkeit und gibt eine entsprechende Meldung aus.



Das XML-Dokument ist gültig



Fehler: Ein Element wurde nicht geschlossen

Der Sinn von XML ist es, die Struktur einer XML-Datei über eine DTD zu definieren, die für jeden zugänglich und nutzbar sein soll. Dazu wird die DTD jedem anderen Autor auf einem Webserver zur weiteren Nutzung zur Verfügung gestellt. Zum Einbinden einer solchen öffentlich zugänglichen Dokumenttyp-Definition benutzen Sie das Schlüsselwort `PUBLIC`.

```
<!DOCTYPE name PUBLIC "Kennung" "URI">
```

Die Einbindung einer öffentlichen DTD wird um den Parameter *Kennung* in Form des **Formal Public Identifier** (FPI) erweitert. Er beinhaltet die Angaben zum Herausgeber, die Angabe des Dokumententyps sowie die Sprache, in der sie verfasst wurde.

Beispielsweise können Sie die öffentlich verfügbaren Dokumenttyp-Definitionen des W3-Konsortiums zur Definition einer XHTML-Struktur benutzen und in das entsprechende Dokument einbinden.

```
<!-- Öffentliche XHTML1.1-DTD von W3.org -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Aufteilung der Kennung

-	Der Herausgeber ist nicht als Besitzer registriert (ein '+' steht für registrierte Besitzer).
W3C	Der Herausgeber der Definition ist das W3-Konsortium.
DTD XHTML 1.1	Der Dokumententyp ist die Definition der XHTML-1.1-Spezifikation.
EN	Die Sprache, in der die Definition verfasst wurde, ist Englisch.

Erweiterte externe Teilmenge

Definitionen von Elementen, Entities oder Attributen einer externen DTD können Sie in XML durch eine interne Definition erweitern. Entities sind Platzhalter für bestimmte festgelegte Inhalte.



In HTML erfolgt eine analoge Kombination von externen und internen Daten bei der Einbindung von Cascading Stylesheets und JavaScript-Funktionen.

```
<!DOCTYPE name SYSTEM "URI" [interne Definition]>
```

Zum Erweitern der externen DTD geben Sie hinter der Angabe der URI innerhalb der eckigen Klammern `[]` die internen Dokumenttyp-Definitionen für Ihr XML-Dokument an.

Beispiel: *kap04\musiksammlung.dtd*

Sie stellen anderen Nutzern Ihre externe DTD für den Aufbau einer Musiksammlung im Internet zur Verfügung.

```
<!ELEMENT MUSIKSAMMLUNG (ALBUM+)>
<!ELEMENT ALBUM (AUTOR, (INTERPRET | GRUPPE)+, TITEL, LIED*)>
  <!ELEMENT AUTOR (#PCDATA)>
  <!ELEMENT INTERPRET (#PCDATA)>
  <!ELEMENT GRUPPE (#PCDATA)>
  <!ELEMENT TITEL (#PCDATA)>
  <!ELEMENT LIED (#PCDATA)>
```

Beispiel: *kap04\musiksammlung-entity.xml*

Da Sie nicht immer im Element `<AUTOR>` Ihren Namen vollständig angeben möchten, legen Sie ein Namens-kürzel fest, das nur in Ihrem XML-Dokument gültig sein soll.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
① <!DOCTYPE MUSIKSAMMLUNG SYSTEM "musiksammlung.dtd"
② [
③   <!ENTITY mm "Max Mustermann">
④ ]>
⑤ <MUSIKSAMMLUNG>
  <ALBUM>
⑥   <AUTOR>&mm;</AUTOR>
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
```

```

<LIED>Honey</LIED>
<LIED>Find my baby</LIED>
<LIED>Porcelain</LIED>
<LIED>...</LIED>
</ALBUM>
</MUSIKSAMMLUNG>

```

- ① Die Definition in der Datei *musiksammlung.dtd* wird in das XML-Dokument eingebunden.
- ② Die interne Definition wird eingeleitet.
- ③ Mit der Deklaration `<!ENTITY ... >` legen Sie für Ihren Namen den Platzhalter fest. In diesem Fall wird für den Namen `Max Mustermann` der Platzhalter `mm` festgelegt. Nähere Informationen zu Entities folgen im nächsten Kapitel.
- ④ Die interne Definition wird abgeschlossen.
- ⑤ Es folgen die eigentlichen XML-Elemente.
- ⑥ Der Inhalt des Elements `AUTOR` wird mit dem entsprechenden Namen gefüllt. Angegeben wird in diesem Fall der Platzhalter `mm`, der von den Zeichen `&` und `:` eingeschlossen ist. Das Anwendungsprogramm mit dem XML-Parser erkennt, dass es sich hierbei um einen Entity-Verweis handelt, und verwendet automatisch die entsprechende Zeichenfolge `Max Mustermann`.

4.6 Gültiges Dokument

Definition eines gültigen Dokuments

In der Definition der Wohlgeformtheit ist festgelegt, dass ein XML-Dokument aus dem Prolog und mindestens einem Element bestehen muss. In einem gültigen Dokument, im Englischen „valid document“, müssen zusätzlich die folgenden Kriterien erfüllt sein.

- ✓ Es muss eine interne oder externe Dokumenttyp-Definition enthalten.
- ✓ Das Dokument muss sich an die aufgestellten Regeln der DTD halten.
- ✓ Es dürfen nur die in der DTD definierten Elemente innerhalb des Dokuments benutzt werden.
- ✓ Alle notwendigen Attribute müssen verwendet werden.
- ✓ Die Werte der Attribute müssen gültig sein.
- ✓ Alle Inhalte der Elemente müssen den festgelegten Datentypen entsprechen.

Um ein Dokument auf Gültigkeit zu testen, ist je nach verwendetem XML-Editor ein zusätzliches Programm notwendig.



Parser

Ein XML-Dokument kann nicht direkt von einer Anwendung verarbeitet werden, sondern wird von einem Parser in seine Bestandteile zerlegt. Dabei wird es auf Wohlgeformtheit und Gültigkeit überprüft. Der Parser liest ein Dokument ein und übermittelt an die Anwendung (z. B. einen Browser) die Eigenschaften und die Struktur der Daten. In XML nutzen die verarbeitenden Programme Parser, um die Gültigkeit eines XML-Dokuments zu überprüfen. Erst wenn die Daten keinen Fehler aufweisen, werden sie zur Weiterverarbeitung durch die Anwendung genutzt.

Es werden zwei Typen von Parsern unterschieden.

Nicht validierende Parser	Sie kontrollieren lediglich, ob die XML-Elemente gegen die Wohlgeformtheit des Dokuments verstoßen.
Validierende Parser	Sie überprüfen die Datenstruktur eines XML-Dokuments auf die Einhaltung der Dokumenttyp- oder Schema-Definitionen. Das Prüfen auf Gültigkeit schließt dabei automatisch das Prüfen auf die Wohlgeformtheit ein.

Stößt der Parser auf Verstöße gegen die XML-Syntax und die festgelegten Dokumenttyp-Definitionen, wird die Anwendung darüber benachrichtigt. Eine weitere Auswertung der Daten ist nicht sinnvoll, da sich Folgefehler einstellen können.



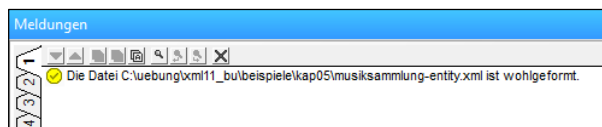
Die XML-Spezifikation gibt keine Auskunft darüber, auf welche Art eine XML-Prüfung durchzuführen ist. Es bleibt somit Ihre Entscheidung, welche Parser Sie verwenden und welche Prüfungen stattfinden sollen.

Dokument auf Gültigkeit testen

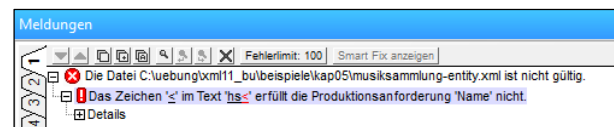


Im Programm Editix können Sie die Gültigkeit einer XML-Datei und deren Dokumenttyp-Definition direkt über den Menüpunkt *XML - Check for a well-formed / valid document* überprüfen lassen.

Im Programm XMLSpy erfolgt der Test auf Wohlgeformtheit über den Menüpunkt *XML - Wohlgeformtheit prüfen*, die Validierung über *XML - XML validieren*.



Das XML-Dokument ist wohlgeformt



Eine nicht definierte Abkürzung wurde gefunden

4.7 Schnellübersicht

Was bedeutet in der Element-Definition einer DTD ...?	
<code>()</code>	Angabe eines oder mehrerer Unterelemente
<code>,</code>	Reihenfolge der Unterelemente
<code> </code>	Oder-Operator für die Auswahl der Elemente
<code>?</code>	Das Element darf weggelassen oder einmal angegeben werden (0.. 1).
<code>*</code>	Das Element darf weggelassen oder beliebig oft angegeben werden (0.. n).
<code>+</code>	Das Element muss mindestens einmal und kann mehrmals angegeben werden (1.. n).
<code>#PCDATA</code>	Der Dateninhalt kann aus beliebigen Zeichenketten bestehen.
<code>EMPTY</code>	Definition eines Elements ohne Inhalt
<code>ANY</code>	Das Element darf jedes andere Element der DTD enthalten.

Sie möchten ...	
eine interne Teilmenge der DTD anlegen	<code><!DOCTYPE name [Element-Definitionen] ></code>
eine externe Teilmenge der DTD einbinden	<code><!DOCTYPE name SYSTEM "URI" ></code> <code><!DOCTYPE name PUBLIC "URI" ></code>
eine interne Definition hinzufügen	<code><!DOCTYPE name SYSTEM "URI" [interne DTD] ></code> <code><!DOCTYPE name PUBLIC "URI" [interne DTD] ></code>
Elementgruppen deklarieren	<code><!ELEMENT name (Unterelemente) ></code>
Elemente definieren	<code><!ELEMENT name (#PCDATA) ></code> <code><!ELEMENT name (#EMPTY) ></code>

4.8 Übungen

Übung 1: Theoretische Fragen

Übungsdatei: --

Ergebnisdatei: *kap04\uebung1-3.html*

- ① Erklären Sie, wozu Parser notwendig sind und welche Unterschiede es zwischen den zwei Parsertypen gibt.
- ② Erläutern Sie, wann ein Dokument als "gültiges Dokument" bezeichnet wird.
- ③ Legen Sie die Vorteile einer externen Dokumenttyp-Definition dar.

Übung 2: DTD festlegen und auslagern

Übungsdatei: --

Ergebnisdateien: *kap04\uebung4.xml*,
kap04\uebung5.xml,
kap04\seminar.dtd

- ① Erstellen Sie eine interne Dokumenttyp-Definition für die XML-Struktur des Dokuments, das die Informationen zum Seminar aus der Übung des Kapitels 3 enthält. Ignorieren Sie dabei die angegebenen Attribute.
- ② Lagern Sie die erstellte DTD aus und prüfen Sie die Gültigkeit des Dokuments mithilfe des Programms Microsoft XML Validation Tool.

Übung 3: Externe DTD erstellen

Übungsdatei: --

Ergebnisdateien: *kap04\uebung6.xml*,
kap04\kfz.dtd

- ① Erstellen Sie eine externe Dokumenttyp-Definition für die Fahrzeugverwaltung aus der Übung von Kapitel 3.

5 DTD – Attribute von Elementen

In diesem Kapitel erfahren Sie

- ✓ wie Sie die Attribute eines Elements festlegen und nutzen
- ✓ welche Attributtypen es gibt
- ✓ wie Sie Platzhalter mithilfe von Entities einsetzen
- ✓ wie Sie auf externe Binärdaten verweisen

Voraussetzungen

- ✓ Aufbau eines XML-Dokuments
- ✓ Elemente der DTD

5.1 Attributlisten-Definition

Attribute werden verwendet, um Elemente genauer zu spezifizieren. Auch in XML können Sie die Attribute eines Elements festlegen.

Eine Attributlisten-Definition in einer DTD hat die folgenden Eigenschaften:

- ✓ Sie legt fest, welchem Element bestimmte Attribute zugewiesen werden können.
- ✓ Es werden der Typ eines Attributs und die möglichen Werte beschrieben.
- ✓ Sie können eine bestimmte Voreinstellung eines Attributs festlegen. Diese kann zu einer speziellen Interpretation durch eine Anwendung dienen.

Grundlegend wird ein Attribut über das Schlüsselwort `ATTLIST` innerhalb der Dokumenttyp-Definition festgelegt.

```
<!ATTLIST Element AttributName (AttributTyp)>
```

Das Element, dem das Attribut zugewiesen werden soll, wird als Parameter `Element` bezeichnet. Dahinter folgt die Angabe des Attributnamens. Über die Angabe `AttributTyp` wird festgelegt, welche Werte dem Attribut übergeben werden können.

Es gibt drei Gruppen von Attributtypen:

- ✓ Eine beliebige Zeichenkette, die über das Schlüsselwort `CDATA` definiert wird.
- ✓ Eine Aufzählung, für die es kein spezielles Schlüsselwort gibt. Bei einer Aufzählung werden die möglichen Werte direkt vorgegeben.
- ✓ Die Schlüsselwörter, die unter dem Begriff "Token" zusammengefasst werden. Die möglichen Schlüsselwörter werden im weiteren Verlauf des Kapitels näher erläutert.

Beispiel

Das HTML-Element `IMG` zum Darstellen einer Grafik besitzt mehrere Attribute, wie die Rahmenbreite `border`, die alternative Textangabe `alt` bzw. `title` oder die Größenangaben `height` und `width`. Das Attribut, das Sie angeben müssen, um die Grafik im Browser anzeigen zu lassen, ist die Angabe des Bildnamens über das Attribut `src`. Die Kurzform der Attributdefinition, wenn alle anderen Attribute erst einmal nicht betrachtet werden, lautet:

```
<!ATTLIST IMG src CDATA #REQUIRED>
```


In der Attributliste des Elements `IMG` wird `src` als Attribut vereinbart. Die Angabe von `CDATA` besagt, dass der Name der Quelldatei `src` eine beliebige Zeichenfolge enthalten kann, die zwingend angegeben werden muss (`#REQUIRED`).

Das Schlüsselwort `CDATA` gibt an, dass einem Attribut jede beliebige Zeichenkette zugeordnet werden kann.

Folgendes sollten Sie beachten:

- ✓ Das Schlüsselwort `ATTLIST` ist immer in Großbuchstaben zu schreiben.
- ✓ Die Definition eines Attributs kann an einer beliebigen Stelle innerhalb der DTD vorgenommen werden. Um sie jedoch leichter lesbar zu machen, notieren Sie die Definition direkt hinter dem entsprechenden Element.
- ✓ Für ein Element können auch mehrere Attributlisten-Definitionen angelegt werden. Sie werden vom Parser automatisch zu einer Liste zusammengefasst.
- ✓ Werden mehrere Deklarationen für ein und dasselbe Attribut eines Elements vorgenommen, wird nur die erste berücksichtigt. Nachfolgende Definitionen werden ignoriert.

Attributvorgaben

Oft ist es sinnvoll, bestimmte Werte eines Attributs vorzugeben. Das Attribut kann dann nur einen der angebotenen Werte annehmen. Die Festlegung der Attributvorgabe erfolgt über die Definition der Wertvorgaben.

```
<!ATTLIST IMG src CDATA
                align (left|center|right)>
```

Die Werte werden in Klammern angegeben und mit dem Zeichen `|` voneinander getrennt. Der Nutzer kann nur eine der Vorgaben in dem entsprechenden Attribut verwenden. Das obige Beispiel legt fest, dass das Attribut `align` entweder den Wert `left`, `center` oder `right` annehmen kann. Andere Angaben verursachen Fehler innerhalb des verarbeitenden Programms.

Wertvorgabe

Standardmäßig können Sie dem Attribut einen bestimmten Vorgabewert zuweisen. Dabei wird ein Wert vorgegeben, den das Attribut automatisch annehmen soll, wenn es nicht angegeben wird.

Ein Beispiel ist das Attribut `align` des HTML-Elements `IMG`. Geben Sie das Attribut nicht an, wird das Bild automatisch linksbündig dargestellt.

```
<!ATTLIST IMG src CDATA
                align (left|center|right) "left">
```

Für das Attribut `align` werden die möglichen Werte `left`, `center` und `right` definiert. Mit der nachfolgenden Angabe von `left` wird automatisch festgelegt, dass das Attribut `align` den Wert `left` erhält, wenn das Attribut nicht angegeben wird.

Bei der Angabe eines vorgegebenen Wertes ist darauf zu achten, dass dieser immer in Anführungszeichen `"` gesetzt werden muss.

Wenn Sie ein Attribut definiert haben, muss es gegebenenfalls nicht unbedingt angegeben werden. Darum müssen Sie festlegen, ob das Attribut angegeben werden muss oder ob es ein optionaler Parameter ist. Hierfür stehen die Schlüsselworte `#REQUIRED` und `#IMPLIED` zur Verfügung. Über eine Wertvorgabe können Sie festlegen, welcher Wert automatisch als Vorgabewert genutzt werden soll. Feste Werte definieren Sie mit `#FIXED`.

#REQUIRED

Dieses Schlüsselwort legt fest, dass ein bestimmtes Attribut unbedingt angegeben werden muss. Wird das Attribut nicht angegeben, ist das XML-Dokument nicht gültig.

```
<!ATTLIST IMG src CDATA #REQUIRED>
```

Dieses zusätzliche Schlüsselwort legt eindeutig fest, dass innerhalb des Elements `IMG` unbedingt das Attribut `src` angegeben werden muss.

Beispiel: *kap05\attlist_required.xml*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE HTML [
  <!ELEMENT HTML (IMG)>
  <!ELEMENT IMG EMPTY>
  <!ATTLIST IMG src CDATA #REQUIRED
                align (left|center|right) "left">
]>
<HTML>
  <IMG src="bild.jpg" align="center" />
</HTML>
```

#IMPLIED

Ein weiteres Schlüsselwort ist die optionale Angabe `#IMPLIED`. Mit dieser Angabe stellen Sie frei, ob das definierte Attribut genutzt wird. Die Definition der alternativen Beschreibung einer Grafik lautet beispielsweise:

```
<!ATTLIST IMG src CDATA #REQUIRED
                alt CDATA #IMPLIED>
```

Es ist Ihnen also freigestellt, ob Sie das Attribut innerhalb des Elements angeben oder nicht. Auch das HTML-Element `IMG` enthält Attributvorgaben, wie z. B. `alt` für die Textalternative zur Grafik.

Beispiel: *kap05\attlist_implied.xml*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE HTML [
  <!ELEMENT HTML (IMG)>
  <!ELEMENT IMG EMPTY>
  <!ATTLIST IMG src CDATA #REQUIRED
                alt CDATA #IMPLIED
                align (left|center|right) "left">
]>
<HTML>
  <IMG src="bild.jpg" alt="Ein Bild" />
</HTML>
```

#FIXED

Über das Schlüsselwort `#FIXED` können Sie einen festen Wert definieren. In diesem Fall kann das Attribut keinen anderen Wert annehmen, da dieses Schlüsselwort für die Definition einer Art konstanter Attribute bestimmt ist.

```
<!ATTLIST IMG src CDATA #REQUIRED
                alt CDATA #IMPLIED
                align (left|center|right) "left"
                typ CDATA #FIXED "image">
```

Auch wenn in dem betreffenden Element dem erdachten Attribut `typ` kein Wert zugewiesen wird, erhält es standardmäßig den festen Vorgabewert `image`.

Die fast vollständige Definition der möglichen Attribute für das Einbinden einer Grafik in eine Webseite wird anhand der Dokumenttyp-Definition des HTML-Elements `IMG` gezeigt.

Beispiel: *kap05\img.xml*

Im folgenden Beispiel legen Sie die Dokumenttyp-Definition für das HTML-Element `IMG` an und definieren die verschiedenen Attribute. In der XML-Struktur werden die beiden Attribute `src` und `alt` verwendet.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?></code>
	<code><!DOCTYPE HTML [</code>
①	<code><!ELEMENT HTML (IMG)></code>
②	<code><!ELEMENT IMG EMPTY></code>
③	<code><!ATTLIST IMG</code>
④	<code>src CDATA #REQUIRED</code>
⑤	<code>alt CDATA #IMPLIED</code>
⑥	<code>align (left center right) "left"</code>
	<code>title CDATA #IMPLIED</code>
	<code>height CDATA #IMPLIED</code>
	<code>width CDATA #IMPLIED</code>
	<code>border CDATA #IMPLIED</code>
	<code>hspace CDATA #IMPLIED</code>
	<code>vspace CDATA #IMPLIED></code>
	<code>]></code>
⑦	<code><HTML></code>
	<code></code>
	<code></HTML></code>

- ① Innerhalb der Dokumenttyp-Definition wird das Element `IMG` als Unterelement des Elements `HTML` festgelegt.
- ② Das Element `IMG` enthält keinen Wert und wird somit als leerer Elementinhalt `EMPTY` definiert.
- ③ Für das Element `IMG` wird die Definition der möglichen Attribute eingeleitet.
- ④ Um eine Grafik darzustellen, wird über das Schlüsselwort `#REQUIRED` der Name der zu verwendenden Datei als Wert des Attributs `src` angegeben. Die Angabe des Attributs wird dadurch erzwungen. Der Wert des Attributs kann eine beliebige Zeichenfolge sein (`CDATA`).
- ⑤ Die Alternativangabe der Bilddatei wird über das Attribut `alt` angegeben. Da diese Angabe nicht notwendig ist, um ein Bild anzuzeigen, wird es, wie auch die nachfolgenden Attribute, als `#IMPLIED` definiert.
- ⑥ Die Ausrichtung einer Grafik ist eine mögliche Angabe, die nicht zwingend erforderlich ist. Die Werte für das Attribut `align` sind in Klammern vorgegeben. Entweder kann das Attribut den Wert `left`, `center` oder `right` enthalten. Andere Werte sind nicht möglich. Wird kein Attribut angegeben, wird automatisch das Attribut `align` mit dem Wert `left` benutzt.
- ⑦ Das Element `IMG` wird mit einigen Attributangaben verwendet.

Attributliste der Musiksammlung

Zur näheren Erläuterung wird die bereits erstellte Musiksammlung als Beispiel verwendet. Das Attribut `typ` soll, wie bisher, innerhalb des Elements `ALBUM` verwendet werden, um die Art des Tonträgers anzuzeigen.

```
<MUSIKSAMMLUNG>
  <ALBUM typ="...">
    <AUTOR></AUTOR>
    <INTERPRET></INTERPRET>
    <TITEL></TITEL>
    <LIED></LIED>
  </ALBUM>
</MUSIKSAMMLUNG>
```

Zur Auswahl stehen hierbei die Typen CD, Vinyl als Synonym für die Schallplatte, MP3 für die Alben im entsprechenden Soundformat und MC für die gute alte Musikkassette. Die Attributlisten-Definition für das Element ALBUM legen Sie folgendermaßen fest:

```
<!ATTLIST ALBUM typ (CD | MP3 | Vinyl | MC)>
```

Da heutzutage mehr MP3s als CDs, Schallplatten oder Musikkassetten verkauft werden, können Sie bei der Attributlisten-Definition davon ausgehen, dass in der Musiksammlung die Mehrzahl der Alben als MP3 angegeben werden. In diesem Fall ist es besser, das Attribut `typ="MP3"` als Voreinstellung zu definieren.

```
<!ATTLIST ALBUM typ (CD | MP3 | Vinyl | MC) "MP3">
```

Wenn das Attribut `typ` jetzt nicht angegeben wird, erhält das Album automatisch den Typ MP3. Demzufolge wären beide nachfolgende Angaben, mit dem Attribut ① und ohne Attribut ②, identisch.

①	<pre><ALBUM typ="MP3"> <TITEL>Minor Earth Major Sky</TITEL> </ALBUM></pre>
②	<pre><ALBUM> <TITEL>Minor Earth Major Sky</TITEL> </ALBUM></pre>

Beispiel: *kap05\musiksammlung-dtd.xml*

Dies ist die vollständige XML-Datei für die Musiksammlung inklusive der Attributlisten-Definition.

①	<pre><?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?> <!DOCTYPE MUSIKSAMMLUNG [<!ELEMENT MUSIKSAMMLUNG (ALBUM+)> <!ELEMENT ALBUM (AUTOR, (INTERPRET GRUPPE)+, TITEL, LIED*)> <!ATTLIST ALBUM typ (CD MP3 Vinyl MC) "MP3"> <!ELEMENT AUTOR (#PCDATA)> <!ELEMENT INTERPRET (#PCDATA)> <!ELEMENT GRUPPE (#PCDATA)> <!ELEMENT TITEL (#PCDATA)> <!ELEMENT LIED (#PCDATA)>]> <MUSIKSAMMLUNG> <ALBUM> <AUTOR></AUTOR> <INTERPRET></INTERPRET> <TITEL></TITEL> <LIED></LIED> </ALBUM> </MUSIKSAMMLUNG></pre>
---	--

- ① Das Hauptelement MUSIKSAMMLUNG darf mehrere Elemente mit der Bezeichnung ALBUM enthalten.
- ② Innerhalb des Elements ALBUM dürfen wiederum die in Klammern aufgelisteten Elemente angegeben werden.
- ③ Zusätzlich wird für das Element ALBUM das Attribut `typ` festgelegt. Dieses darf die Werte CD, MP3, Vinyl oder MC enthalten. Geben Sie später in der XML-Struktur das Attribut nicht an, wird der Attributwert durch das verarbeitende Programm automatisch auf den Wert MP3 gesetzt.
- ④ Die möglichen Inhalte der nachfolgenden Elemente, die innerhalb des Elements ALBUM notiert werden müssen, werden festgelegt.
- ⑤ Die XML-Struktur der Musiksammlung wird angelegt.

5.2 Attributtypen

CDATA

Die am meisten verwendeten Attributtypen sind CDATA-Attribute. Deren Inhalte können, genau wie #PCDATA in Elementen, aus beliebigen Zeichenfolgen bestehen. Der Unterschied zwischen beiden besteht darin, dass bei CDATA auch Zeichen enthalten sein dürfen, die normalerweise als XML-Zeichen interpretiert werden könnten, wie z. B. `>`, `<`. Bei #PCDATA (parsed character data) zählen diese Zeichen zur XML-Sprache.

①	<code><!ATTLIST PERSON name CDATA></code>
	<code>...</code>
②	<code><PERSON name="Max Mustermann"></PERSON></code>

- ① Der Name einer Person kann beliebige Zeichen enthalten.
- ② Dem Element PERSON wird das Attribut `name="Max Mustermann"` zugewiesen.

Aufzählung

Neben der Angabe beliebiger Zeichen als Wert für ein Attribut können Sie auch aufzählen, welche Werte vorgegeben sind. Aus diesen Werten ist einer auszuwählen.

①	<code><!ATTLIST TELEFON nutzungsart (geschäftlich privat g p)></code>
	<code>...</code>
②	<code><TELEFON nutzungsart="p">555-1234</TELEFON></code>

- ① Ein gültiger Wert für die Nutzung des Telefons ist in diesem Fall `geschäftlich` oder `privat`. Auch die entsprechende Abkürzung `g` oder `p` kann mit dieser Definition verwendet werden.
- ② Das Element TELEFON wird über das Attribut `nutzungsart` als private Telefonnummer gekennzeichnet.

Token

Unter dem Begriff Token werden verschiedene Attributtypen zusammengefasst. Ein Token ist eine Zeichenfolge, die für einen Parser eine syntaktische Einheit darstellt und über die nachfolgenden Attributtypen festgelegt und angesprochen wird.

ID

Für jedes Element in einem XML-Dokument können Sie eine eindeutige Bezeichnung (ID) vergeben. Mit dieser ID können Sie beispielsweise das entsprechende Element über eine Skriptsprache direkt ansprechen und auswerten. Die ID bezieht sich global auf alle Elemente des Dokuments. Der Wert darf nur einmal vorkommen.

①	<code><!ATTLIST PERSON kennung ID #IMPLIED></code>
	<code>...</code>
②	<code><PERSON kennung="P1487">Max Mustermann</PERSON></code>

- ① Die Attributlisten-Definition legt fest, dass innerhalb des Elements PERSON über das Attribut `kennung` eine eindeutige Kennzeichnung zugewiesen werden kann (Attributtyp ID).
- ② Die eindeutige Bezeichnung wird festgelegt.

IDREF und IDREFS

Ein Attribut von diesem Typ verweist auf ein Element, das zuvor mithilfe des Attributtyps ID eindeutig bestimmt wurde. Zusätzlich müssen Sie angeben, ob es sich um eine notwendige Angabe (`#REQUIRED`) oder um eine optionale Angabe (`#IMPLIED`) handelt. Der Typ IDREF kann nur verwendet werden, wenn ein Element mit der entsprechenden Referenz-ID vorhanden ist.

Auf diese Weise können Sie Verknüpfungen zwischen den Daten eines XML-Dokuments herstellen. So ist es beispielsweise möglich, zwischen mehreren Personen eine Verwandtschaft zu kennzeichnen.

Beispiel: *kap05/token-idref.xml*

Im folgenden Beispiel werden verschiedenen Personen über den Attributtyp `ID` eindeutige Kennungen zugewiesen und über den Attributtyp `IDREF` angesprochen. Damit soll das Verwandtschaftsverhältnis der aufgelisteten Personen gekennzeichnet werden.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?></code>
	<code><!DOCTYPE TOKEN [</code>
	<code><!--ELEMENT TOKEN (PERSON*)--></code>
①	<code><!--ELEMENT PERSON (#PCDATA)--></code>
②	<code><!--ATTLIST PERSON kennung ID #REQUIRED--></code>
	<code><!--ATTLIST PERSON verwandt IDREF #IMPLIED--></code>
	<code>] ></code>
	<code><TOKEN></code>
③	<code><PERSON kennung="P1">Max Müller</PERSON></code>
④	<code><PERSON kennung="P2" verwandt="P3">Max Mustermann</PERSON></code>
⑤	<code><PERSON kennung="P3" verwandt="P2">Erwin Mustermann</PERSON></code>
	<code></TOKEN></code>

- ① Für das Element wird das notwendige Attribut `kennung` bereitgestellt, um eine Person eindeutig über den Attributtyp `ID` zu kennzeichnen.
- ② Des Weiteren ist die Angabe der Verwandtschaft der Person über das Attribut `verwandt` vom Typ `IDREF` möglich.
- ③ Der ersten Person wird die Kennung `P1` zugewiesen.
- ④ Die nächste Person erhält die Kennung `P2` und ist verwandt mit einer Person, welche die Kennung `P3` hat.
- ⑤ Die Person Erwin Mustermann besitzt die Kennung `P3` und steht somit in Verwandtschaft mit Max Mustermann.

Das Attribut `IDREFS` ist die Pluralform von `IDREF` und erlaubt die Angabe mehrerer Verknüpfungen. Die einzelnen Angaben werden durch Leerzeichen voneinander getrennt.

Beispiel: *kap05/token-idrefs.xml*

	<code><!--ATTLIST PERSON kennung ID #REQUIRED--></code>
	<code><!--ATTLIST PERSON verwandt IDREFS #IMPLIED--></code>
	<code>...</code>
	<code><TOKEN></code>
	<code><PERSON kennung="P1">Max Müller</PERSON></code>
	<code><PERSON kennung="P2" verwandt="P3 P4">Max Mustermann</PERSON></code>
	<code><PERSON kennung="P3" verwandt="P2 P4">Erwin Mustermann</PERSON></code>
	<code><PERSON kennung="P4" verwandt="P2 P3">Erika Mustermann</PERSON></code>
	<code></TOKEN></code>

NMTOKEN und NMTOKENS

Mit dem Attributtyp `CDATA` kann jedes beliebige Zeichen benutzt werden. Mit `NMTOKEN` schränken Sie die erlaubten Zeichen auf Buchstaben, Ziffern und die Zeichen Punkt `.`, Doppelpunkt `:`, Bindestrich `-` und Unterstrich `_` ein. Weder Leerzeichen noch andere Zeichen sind zulässig. Mit der Pluralform `NMTOKENS` lassen sich mehrere Einschränkungen festlegen.

	<code><!--ATTLIST PERSON kuerzel NMTOKEN #IMPLIED--></code>
	<code>...</code>
	<code><TOKEN></code>
	<code><PERSON kuerzel="M.M.">Max Müller</PERSON></code>
	<code></TOKEN></code>

ENTITY und ENTITIES

Mithilfe des Attributtyps `ENTITY` können Sie auf Dateien und andere Objekte verweisen. Zur Definition einer Entity vgl. Abschnitt 5.3. Die Syntax für die Deklaration eines Entity-Attributs in einer externen DTD lautet:

```
<!ATTLIST ElementName AttributName ENTITY>
```

NOTATION und NOTATIONS

Notationen werden eingesetzt, um auf externe Informationen zuzugreifen. So wäre es beispielsweise denkbar, dass ein Programm aufgerufen werden soll, um eine spezielle Datei anzuzeigen.

```
<!ATTLIST ElementName AttributName (Vorgaben) NOTATION>
```

Innerhalb der Attributlisten-Definition bestimmen Sie das Attribut für ein Element. In Klammern setzen Sie die möglichen Vorgaben. Das Schlüsselwort `NOTATION` schreiben Sie an das Ende der Definition. Zur Definition eines Notationsdatentyps vgl. Abschnitt 5.4.

```
<!ATTLIST PERSON foto NOTATION>
...
<TOKEN>
  <PERSON foto="images/maxmueller.jpg">Max Müller</PERSON>
</TOKEN>
```

5.3 Referenz auf Entitäten

Allgemeine Entities

Über Entities (Platzhalter) werden Sonderzeichen mithilfe von Bezeichnungen dargestellt. Der Aufbau von Entities ist einheitlich. Jeder Entity-Verweis beginnt mit dem UND-Zeichen `&`, gefolgt von der Bezeichnung des Sonderzeichens und dem abschließenden Semikolon `;`.

Die vordefinierten Entities für XML sind:

Code	Erläuterung	Zeichen
<code>&lt;</code>	kleiner als	<code><</code>
<code>&gt;</code>	größer als	<code>></code>
<code>&amp;</code>	Ampersand	<code>&</code>

Code	Erläuterung	Zeichen
<code>&apos;</code>	einfaches Anführungszeichen	<code>'</code>
<code>&quot;</code>	doppeltes Anführungszeichen	<code>"</code>

In XML können Sie statt der Sonderzeichen auch Abkürzungen für lange und häufig eingesetzte Zeichenfolgen in den Inhalten eines Elements festlegen. Statt einen bestimmten Text immer erneut anzugeben, können Sie die Abkürzung verwenden. Ebenso können Sie Entities für immer wiederkehrende Elementlisten einsetzen.

Entities können Sie nicht für die Bezeichnung eines Elements oder eines Attributs verwenden.



Syntax

Das Schlüsselwort für das Festlegen eines Platzhalters ist `ENTITY`. Danach folgt der Name, der später für den konkreten Text verwendet werden soll.

```
<!ENTITY abkuerzung "Text, auf den sich die Abkürzung bezieht">
<!-- Verwenden als: &abkuerzung; -->
```

Möchten Sie den definierten Platzhalter in Ihrem Dokument verwenden, setzen Sie die entsprechende Entity ein. Dabei umschließen Sie den Entity-Namen mit den Zeichen `&` und `;`.

Beispiel: *kap05/entity.xml*

Es sollen die Namen von Personen als Entities definiert und innerhalb der XML-Elemente als Inhalte verwendet werden.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?></code>
	<code><!DOCTYPE ALLGENTITY [</code>
	<code><!ELEMENT ALLGENTITY (PERSON*)></code>
	<code><!ELEMENT PERSON (#PCDATA)></code>
①	<code><!ENTITY mm "Max Müller"></code>
②	<code><!ENTITY em "Erwin Mustermann"></code>
	<code>] ></code>
	<code><ALLGENTITY></code>
③	<code><PERSON>&mm;</PERSON></code>
④	<code><PERSON>&em;</PERSON></code>
	<code></ALLGENTITY></code>

- ① Mit dem Datentyp ENTITY wird der Abkürzung mm der Wert Max Müller übergeben.
- ② Es wird die Abkürzung em definiert.
- ③ Dem Element PERSON wird die Entity mm als Inhalt übergeben. Sie liefert den Namen Max Müller.
- ④ Hier wird als Inhalt der Wert Erwin Mustermann verwendet.

Parameter-Entity

Parameter-Entities können Sie als Platzhalter innerhalb einer DTD betrachten. Diese Entities wenden Sie an, wenn innerhalb von Elementlisten häufig gleiche Zeichenketten wiederkehren. In der DTD von HTML sind beispielsweise die Möglichkeiten einer Ausrichtung über das Attribut align in einer Parameter-Entity definiert. Die Ausrichtungen von Inhalten in Überschriften, Tabellenzellen und Absätzen erfolgen hierbei immer über das Attribut align mit den Werten left, center oder right.

<code><!ELEMENT HTML (H1 H2 H3 H4 H5 H6 P TD)*></code>
<code><!ELEMENT H1 align (left center right) #IMPLIED></code>
<code><!ELEMENT H2 align (left center right) #IMPLIED></code>
<code>...</code>
<code><!ELEMENT TD align (left center right) #IMPLIED></code>
<code><!ELEMENT P align (left center right) #IMPLIED></code>



Eine Parameter-Entity darf in einer Deklaration eines Elements, eines Attributs oder einer anderen Entität nur dann referenziert werden, wenn diese Referenz in einer externen DTD festgelegt wurde.

Syntax

Mithilfe von Parameter-Entities können Elementlisten als Abkürzung definiert und später wieder abgerufen werden. Die Definition erfolgt über die folgende Syntax:

<code><!ENTITY % umschreibung "wiederkehrende Elementlisten"></code>
<code><!-- Einbinden über: %umschreibung; --></code>

Im Unterschied zur allgemeinen Entity wird in der Parameter-Entity der Platzhalter mit einem Prozentzeichen `%` gekennzeichnet. Danach folgt die Umschreibung der Elementliste. Eingebunden wird der Platzhalter mit einem vorangestellten Prozentzeichen `%` und einem abschließenden Semikolon `;`.

Beispiel: *kap05/htmlalign.dtd*

In einer externen Datei wird die Definition der Parameter-Entities für die Elementlisten festgelegt. Dabei sollen über die Entity *align* die Werte *left*, *center* und *right* an die Elemente übergeben werden. Über die Entity *heading* können Sie verschiedene Größen von Überschriften verwenden.

①	<!ENTITY % heading "H1 H2 H3 H4 H5 H6">
②	<!ENTITY % align "align (left center right) #IMPLIED">
③	<!ELEMENT HTML ((%heading;) P TD)*>
④	<!ELEMENT H1 (#PCDATA)>
⑤	<!ATTLIST H1 %align;>
	<!-- <!ATTLIST H1 align (left center right) #IMPLIED> -->
	<!ELEMENT H2 (#PCDATA)>
	<!ATTLIST H2 %align;>
	<!ELEMENT H3 (#PCDATA)>
	<!ATTLIST H3 %align;>
	<!ELEMENT H4 (#PCDATA)>
	<!ATTLIST H4 %align;>
	<!ELEMENT H5 (#PCDATA)>
	<!ATTLIST H5 %align;>
	<!ELEMENT H6 (#PCDATA)>
	<!ATTLIST H6 %align;>
	<!ELEMENT TD (#PCDATA)>
	<!ATTLIST TD %align;>
	<!ELEMENT P (#PCDATA)>
	<!ATTLIST P %align;>

- ① Die Parameter-Entity wird als *heading* definiert und enthält die Auflistung aller HTML-Elemente für die Überschriften.
- ② Die Angabe *align (left|center|right) #IMPLIED* wird als abgekürzte Parameter-Entity *align* definiert.
- ③ Statt alle möglichen Überschriften (H1-H6) in der Inhaltsdefinition des Hauptelements *HTML* anzugeben, wird die definierte Kurzform *%heading;* verwendet. Da nach der Auflistung der Überschriften noch eine weitere Angabe von Elementen folgt (P und TD), muss die Parameter-Entity in Klammern gesetzt werden.
- ④ Es folgt die Definition aller Elemente, die im Hauptelement *HTML* möglich sind.
- ⑤ Dem Element *H1* wird die Parameter-Entity *%align* zugewiesen. Der Inhalt des Attributs verbirgt sich hinter der festgelegten Abkürzung *%align;* daraus ergibt sich die als Kommentar angegebene Attributlisten-Definition für die möglichen Ausrichtungsarten.

Innerhalb von Elementtyp-Definitionen dürfen für die Namen keine Entities verwendet werden, da jedes Element einzeln definiert werden muss. Die Angabe von <!ELEMENT %heading; (#PCDATA)> würde z. B. beim Parsen der XML-Datei zu einer Fehlermeldung führen.

**Beispiel: *kap05/htmlalign.xml***

Die externe Teilmenge der DTD wird zur Nutzung der Parameter-Entities in die XML-Datei eingebunden.

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE HTML SYSTEM "htmlalign.dtd">
<HTML>
<H1 align="center">zentrierte Überschrift</H1>
<H2 align="right">rechtsbündige Überschrift</H2>
</HTML>

Externe Entities

Eine weitere Möglichkeit, Entities zu definieren, ist der Verweis auf bestehende XML-Dateien oder binäre Dateien, wie Bilder, Dokumente, Musikdateien usw.

```
<!ENTITY abkuerzung SYSTEM "Dateiname" NDATA Binärtyp>
```

Der Entity `abkuerzung` wird durch die Angabe einer Notation (`NDATA=Notation Data`) ein bestimmter Binärtyp zugewiesen. Dieser besagt, dass es sich um nicht interpretierbare Daten handelt, zumindest nicht in XML. Der Parameter `Dateiname` gibt an, auf welche externe Datei sich die Notation bezieht. Der `Dateiname` kann als relativer oder absoluter Pfad angegeben werden.

Externe Entities werden folgendermaßen definiert:

```
<!ENTITY pngdatei SYSTEM "abc123.png" NDATA png>
<!ENTITY bild1 SYSTEM "../images/bild1.gif" NDATA gif89a>
<!ENTITY pdfdatei SYSTEM "http://www.example.com/xyz.pdf" NDATA pdf>
```

Die Entity-Referenz wird anders verwendet als bisher. Sie darf nur als Attributwert vom Typ `Entity` auftreten.

```
<!ATTLIST Elementname Attributname ENTITY Attributvorgabe>
```

Beispiel: *kap05\ndata.xml*

Es werden Entities für einzelne Grafiken gebildet, die eine festgelegte Quelldatei besitzen und vom Datentyp `gif89a` sind. Eingebunden werden diese Grafiken über den Aufruf der entsprechenden Entities.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!DOCTYPE BILDERGALERIE [
  ① <!NOTATION gif89a SYSTEM "GIF">
  ② <!ENTITY bild1 SYSTEM "images/bild1.gif" NDATA gif89a>
  ③ <!ENTITY bild2 SYSTEM "images/bild2.gif" NDATA gif89a>
  <!ELEMENT BILDERGALERIE (IMG)+>
  <!ELEMENT IMG EMPTY>
  ④ <!ATTLIST IMG src ENTITY #REQUIRED>
]>
<BILDERGALERIE>
  ⑤ <IMG src="bild1" />
  ⑥ <IMG src="bild2" />
</BILDERGALERIE>
```

- ① Über das Schlüsselwort `NOTATION` wird das Datenformat `GIF` definiert, welches über das Kürzel `gif89a` angesprochen werden kann. Zu den notwendigen Angaben für diese Definition und deren Auswirkung auf das Verhalten des Programms vgl. Abschnitt 5.4.
- ② Der Entity `bild1` wird der Name einer binären Datei zugewiesen, die unter dem Pfad `images/bild1.gif` zu finden ist. Die Datei ist vom Notationstyp `gif89a`.
- ③ Hier wird der Entity `bild2` der Name der Datei `images/bild2.gif` im Format `gif89a` zugewiesen.
- ④ Das Attribut `src`, das zwingend angegeben werden muss, enthält die soeben definierten Entities.
- ⑤ Dem Element `IMG` wird das Attribut `src` mit dem Wert `bild1` zugewiesen. Der Parser stellt durch die Attributdeklaration ④ fest, dass es sich bei dem Wert um ein Entity handelt. Die Angabe des Wertes `bild1` und dessen Deklaration ② zeigt, wo die Grafikdatei zu finden ist.
- ⑥ Dem zweiten Element `IMG` wird der Name der Binärdatei `images/bild2.gif` ③ zugewiesen.

5.4 Datentyp Notation

In einer Notation werden die Datenformate definiert, die zur Anzeige oder Verarbeitung auf andere Anwendungen verweisen sollen. Dem Parser wird auf diese Weise mitgeteilt, wie er die Entities behandeln soll.

```
<!NOTATION abkuerzung SYSTEM "Verweis oder Datentyp">
```

Die Definition legt fest, welche Abkürzung mit dem angegebenen Dateiformat gleichzusetzen ist. Beispielsweise können Sie über das Schlüsselwort **SYSTEM** die URI bestimmter Anwendungsprogramme angeben. Öffentlich zugängliche Definitionen können Sie auch über das Schlüsselwort **PUBLIC** kenntlich machen.

Nachdem in einer Notationsdeklaration ein Kürzel mit einer Anwendung verbunden wurde, kann ein Attribut vom Typ **NOTATION** deklariert werden.

```
<!ATTLIST ElementName AttributName NOTATION (abkuerzung)>
```

Ein Attribut vom Typ **NOTATION** darf nicht auf einem Element deklariert werden, das als **EMPTY** deklariert wurde.



Beispiel: *kap05\notation.xml*

Es wird die Abkürzung **tif** angelegt, die automatisch bei einem Aufruf als Attributwert eines Elements das angegebene Programm oder den Verweis ausführt.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!DOCTYPE BILDERGALERIE [
  ① <!NOTATION tif SYSTEM "C:\Programme\TV\tifviewer.exe">
    <!ELEMENT BILDERGALERIE (IMG)>
    <!ELEMENT IMG (#PCDATA)>
  ② <!ATTLIST IMG
      typ NOTATION (tif) #REQUIRED
      quelle CDATA #REQUIRED>
]>
<BILDERGALERIE>
  ③ <IMG typ="tif" quelle="images/bild.tif" />
</BILDERGALERIE>
```

- ① In der Notation wird über die Abkürzung **tif** der Pfad zu einer lokalen Anwendung definiert, um das Bild anzuzeigen.
- ② Dem Attribut **typ** des Elements **IMG** wird der Datentyp **NOTATION** zugewiesen.
- ③ Das Element **IMG** erhält über das Attribut **typ** den Notationstyp **tif**. Um das nachfolgende Bild **quelle** in einem Anwendungsprogramm darzustellen, wird auf den Pfad des ausführbaren Bildbetrachters **tifviewer.exe** verwiesen.

Weitere Beispiele für Notationsdeklarationen sind:

```
<!NOTATION mpeg SYSTEM "http://www.example.com/programs/mpeg32.exe">
<!NOTATION mid SYSTEM "midplayer.exe">
<!NOTATION html PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!NOTATION gif PUBLIC "-//Compuserve
//NOTATION Graphics Interchange Format//EN">
<!NOTATION pdf PUBLIC "-//Adobe Inc.//NOTATION Portable Document Format//EN">
```

Die Strukturen der möglichen übergebenen Werte in den Notationen sind noch nicht standardisiert. Wie die Werte interpretiert werden, ist von der verwendeten Anwendung abhängig, welche die Daten verarbeiten soll. Somit ist es derzeit möglich, beliebige Informationen als Werte anzugeben.



Beispiel: *kap05\ms.dtd*

Die bisherige Musiksammlung soll um die Informationen zur Gesamtspielzeit, zur veröffentlichenden Musikfirma (Label) und zum Erscheinungsjahr erweitert werden. Zusätzlich soll es möglich sein, jedem Album ein oder mehrere Bilder in den Formaten gif oder jpg zuweisen zu können.

①	<!NOTATION jpg PUBLIC "-//ISO DIS 10918//NOTATION JPEG Graphics Format//EN">
②	<!NOTATION gif PUBLIC "-//CompuServe//NOTATION Graphics Interchange Format//EN">
③	<!ENTITY MRL "Mute Records Limited">
④	<!ELEMENT MUSIKSAMMLUNG (ALBUM+)>
⑤	<!ELEMENT ALBUM (AUTOR, (INTERPRET GRUPPE), TITEL, GESAMTZEIT?, LABEL?, JAHR?, BILD*, LIED*)>
⑥	<!ATTLIST ALBUM typ (CD MP3 Vinyl MC) "MP3">
⑦	<!ELEMENT AUTOR (#PCDATA)> <!ELEMENT INTERPRET (#PCDATA)> <!ELEMENT GRUPPE (#PCDATA)> <!ELEMENT TITEL (#PCDATA)> <!ELEMENT GESAMTZEIT (#PCDATA)> <!ELEMENT LABEL (#PCDATA)> <!ELEMENT JAHR (#PCDATA)> <!ELEMENT BILD (#PCDATA)> <!ATTLIST BILD typ NOTATION (gif jpg) #REQUIRED quelle CDATA #REQUIRED >
⑧	
⑨	<!ELEMENT LIED (#PCDATA)>

- ① Mit der Notation wird das Datenformat jpg definiert.
- ② Das Graphics Interchange Format der Firma CompuServe wird als Notation gif festgelegt.
- ③ Die Entity MRL wird definiert und kann später als Abkürzung für die Zeichenkette Mute Records Limited eingesetzt werden.
- ④ Das Element MUSIKSAMMLUNG enthält das Unterelement ALBUM, das mehrfach vorkommen kann.
- ⑤ Das Element ALBUM beinhaltet die Elemente AUTOR, TITEL, GESAMTZEIT, LABEL, JAHR, BILD und LIED sowie die über den ODER-Separator verknüpften Elemente INTERPRET und GRUPPE. Diese können je nach Angabe einmalig oder mehrfach im Element ALBUM verwendet werden.
- ⑥ Dem Element ALBUM wird ein optionales Attribut typ zugewiesen. Wird es nicht angegeben, erhält das Attribut automatisch den Wert MP3.
- ⑦ Es folgt die Datentyp-Definition der einzelnen Elemente.
- ⑧ Dem leeren Element BILD werden die Attribute typ und quelle zugewiesen. Das Attribut typ ist vom Datentyp NOTATION und darf entweder den Wert gif oder den Wert jpg enthalten. Der Wert des Attributs quelle hingegen kann eine beliebige Zeichenfolge sein. Beide Attribute sind bei der Angabe des Elements BILD unbedingt mit anzugeben (#REQUIRED).
- ⑨ Zuletzt wird das Element LIED als beliebige Zeichenfolge definiert.

Beispiel: *kap05\ms.xml*

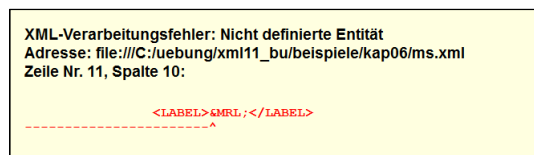
Die Dokumenttyp-Definition *ms.dtd* wird in die XML-Datei der Musiksammlung eingebunden und innerhalb der XML-Struktur verwendet.

①	<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
②	<!DOCTYPE MUSIKSAMMLUNG SYSTEM "ms.dtd" [③ <!ENTITY hs "Heiko Schröder">]>
④	<MUSIKSAMMLUNG>
⑤	<ALBUM>
⑥	<AUTOR>&hs;</AUTOR> <INTERPRET>Moby</INTERPRET> <TITEL>Play</TITEL>

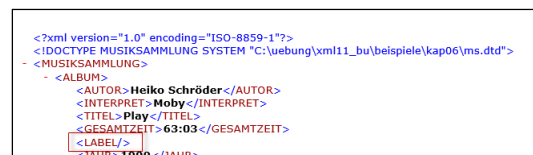
⑦	<code><GESAMTZEIT>63:03</GESAMTZEIT></code>
	<code><LABEL>&MRL;</LABEL></code>
	<code><JAHR>1999</JAHR></code>
	<code><BILD typ="jpg" quelle="images/moby_play1.jpg" /></code>
⑧	<code><BILD typ="jpg" quelle="images/moby_play2.jpg" /></code>
	<code><LIED>Honey</LIED></code>
	<code><LIED>Find my baby</LIED></code>
	<code><LIED>Porcelain</LIED></code>
	<code><LIED>Why does my heart feel so bad?</LIED></code>
	<code></ALBUM></code>
⑨	<code><ALBUM typ="Vinyl"></code>
	<code><AUTOR>&hs;</AUTOR></code>
	<code><GRUPPE>a-ha</GRUPPE></code>
	<code><TITEL>Minor earth major sky</TITEL></code>
	<code><GESAMTZEIT>63:03</GESAMTZEIT></code>
	<code><LABEL>&MRL;</LABEL></code>
	<code><JAHR>1999</JAHR></code>
⑩	<code><BILD typ="gif" quelle="images/aha_minor.gif" /></code>
	<code><LIED>Minor earth major sky</LIED></code>
	<code><LIED>Little black</LIED></code>
	<code><LIED>Velvet</LIED></code>
	<code><LIED>Summer moved on</LIED></code>
	<code></ALBUM></code>
	<code></MUSIKSAMMLUNG></code>

- ① Die XML-Datei wird mit dem Prolog eingeleitet. Die Angabe `standalone="no"` teilt der Anwendung mit, dass dieses Dokument eine externe Datei benötigt.
- ② Die Dokumenttyp-Definition wird als externe Datei *ms.dtd* in die XML-Datei geladen.
- ③ Innerhalb der XML-Datei wird die externe Teilmenge der DTD um die Entity *hs* erweitert. Diese Entity wird später als Platzhalter für den Namen des Autors verwendet.
- ④ Die XML-Daten werden durch das Hauptelement *MUSIKSAMMLUNG* eingeleitet.
- ⑤ Das Untererelement *ALBUM* enthält kein Attribut, besitzt jedoch durch die DTD automatisch das Attribut *typ* mit dem Wert *MP3*.
- ⑥ Der Name des Autors wird über die Entity *hs* angegeben.
- ⑦ Das Element *LABEL* enthält den Wert der Entity *MRL* (*Mute Records Limited*).
- ⑧ Dem Album des Interpreten *Moby* werden zwei Bilder zugewiesen. Diese sind vom Notationstyp *jpg* und befinden sich im Unterordner *images* des XML-Dokuments.
- ⑨ Das zweite Album erschien als Schallplatte und erhält somit das Attribut *typ* mit dem Wert *Vinyl*.
- ⑩ Das Bild dieses Albums ist vom Notationstyp *gif*.

Entitys in externen DTDs werden von den aktuellen Browsern nicht unterstützt. Die Ausführung der Datei führt zu einer Fehlermeldung im Mozilla Firefox bzw. zu einer fehlerhaften Darstellung im Internet Explorer. Opera und Chrome verhalten sich analog zum Firefox.



Fehlermeldung im Mozilla Firefox



Fehlerhafte Darstellung im Internet Explorer

Befindet sich die Entity im internen Definitionsteil der DTD des XML-Dokuments, wird das Dokument korrekt dargestellt.

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft. Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```

- <MUSIKSAMMLUNG>
  - <ALBUM>
    <AUTOR>Heiko Schröder</AUTOR>
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
    <GESAMTZEIT>63:03</GESAMTZEIT>
    <LABEL>Mute Records Limited</LABEL>
    <JAHR>1999</JAHR>

```

Mozilla Firefox

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE MUSIKSAMMLUNG SYSTEM "C:\uebung\xml11_bu\beispiele\kap06\ms2.dtd">
- MUSIKSAMMLUNG
  - ALBUM
    <AUTOR>Heiko Schröder</AUTOR>
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
    <GESAMTZEIT>63:03</GESAMTZEIT>
    <LABEL>Mute Records Limited</LABEL>
    <JAHR>1999</JAHR>
    <BILD quelle="images/moby_play1.jpg" typ="jpg"/>
    <BILD quelle="images/moby_play2.jpg" typ="jpg"/>
    <LIED>Honey</LIED>
    <LIED>Find my baby</LIED>
    <LIED>Porcelain</LIED>

```

Internet Explorer

5.5 Schnellübersicht

Attributtypen	Beschreibung
#REQUIRED	Attribut muss mit dem Element angegeben werden
#IMPLIED	Attribut kann mit dem Element angegeben werden
CDATA	Attributwert darf beliebige Zeichenfolge enthalten
Aufzählung	Mögliche Werte werden vorgegeben
Token	Oberbegriff für verschiedene Attributtypen
ID	Attribut erhält eine eindeutige Bezeichnung
IDREF / IDREFS	Attribut verweist auf ein mit einer ID gekennzeichnetes Element
NMTOKEN / NMTOKENS	Attributwert darf nur eingeschränkte Zeichen enthalten
ENTITY / ENTITIES	Anlegen eines Verweises auf eine Zeichenkette
NOTATION / NOTATIONS	Zugriff auf externe, meist binäre Daten

Sie möchten ...	
Attributlisten anlegen	<! ATTLIST Element AttributName (AttributTyp)>
mögliche Werte vorgeben	<! ATTLIST Element AttributName (Wert1 Wert2...) "Vorgabe">
allgemeine Entities definieren	<! ENTITY abkuerzung "Text, auf den sich die Abkürzung bezieht">
allgemeine Entities einsetzen	&abkuerzung;
Parameter-Entities definieren	<! ENTITY % umschreibung "wiederkehrende Elementlisten">
Parameter-Entities einsetzen	%umschreibung;
externe Entities definieren	<! ENTITY abkuerzung SYSTEM "Dateiname" NDATA Binärtyp>
externe Entities einsetzen	abkuerzung
Datenformate definieren	<! NOTATION abkuerzung SYSTEM "Verweis oder Datentyp">

5.6 Übung

Übung 1: Mit Attributvorgaben arbeiten

Übungsdatei: --

Ergebnisdateien: *kap05\uebung1.xml*,
kap05\uebung1.dtd

Erweitern Sie die Musiksammlung, indem Sie für das Element ALBUM zusätzlich eine Bewertung des Albums zulassen. Es soll jedoch dem Autor überlassen sein, ob er eine Bewertung von 0 bis 5 Punkten für die Qualität des Albums vergeben möchte.

Übung 2: Attributlisten festlegen

Übungsdatei: --

Ergebnisdateien: *kap05\uebung2.xml*,
kap05\uebung2.dtd

- ① Das Element TITEL soll als Attribut `stil` die Werte Schlager, Klassik, Pop, Rock, Independent und Volksmusik enthalten.
- ② Legen Sie in der DTD fest, dass das Attribut `stil` unbedingt angegeben werden muss.

Übung 3: Attributwerte hinzufügen

Übungsdatei: --

Ergebnisdateien: *kap05\uebung3.xml*,
kap05\uebung3.dtd

Fügen Sie in die Dokumenttyp-Definition des XML-Dokuments, das die Angaben zu Ihrem Seminar enthält, die Attribute `anrede` und `typ` ein. Diese Werte sollen für die Elemente eingesetzt werden, die sich auf Personenangaben beziehen. Die Werte sind nachfolgend fest vorgegeben.

`anrede` = Herr oder Frau

`typ` = Seminarleiter oder Teilnehmer

Übung 4: Mit Attributwerten arbeiten

Übungsdatei: --

Ergebnisdateien: *kap05\uebung4.xml*,
kap05\uebung4.dtd

Legen Sie in der Fahrzeugverwaltung Attribute an, mit denen Sie die Maßeinheiten der technischen Details festlegen.

Anmerkung: Nutzen Sie für die Definition der möglichen Attributwerte „ccm“ für „cm³“ sowie „kmh“ statt „km/h“.

6 Namensräume

In diesem Kapitel erfahren Sie

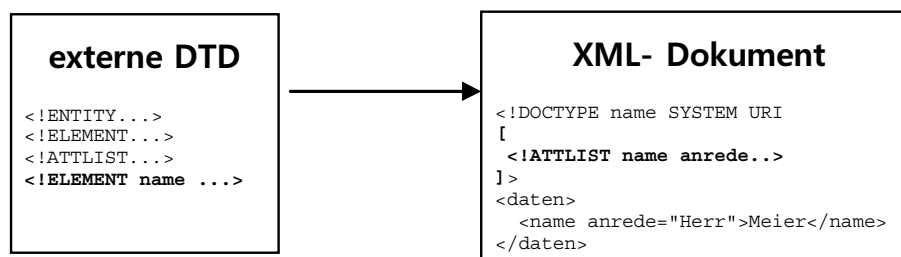
- ✓ was unter Namensräumen zu verstehen ist
- ✓ welche Möglichkeiten es gibt, Namensräume zu definieren
- ✓ wie Sie mithilfe von Namensräumen HTML-Elemente in XML einsetzen können

Voraussetzungen

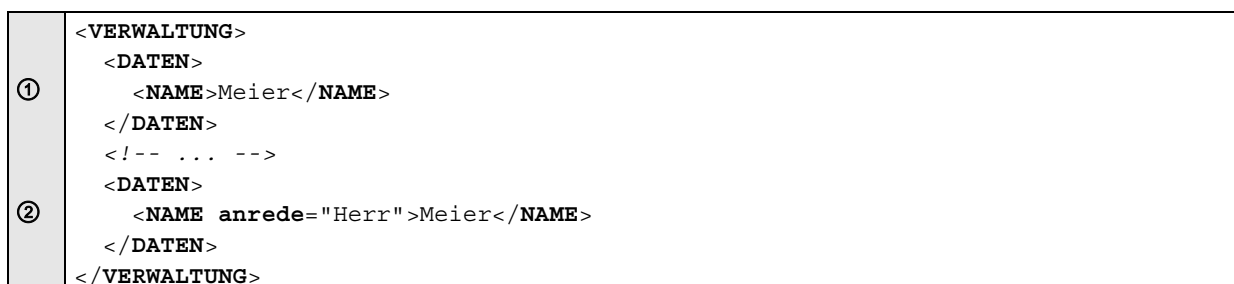
- ✓ Grundlegende HTML-Kenntnisse

6.1 Grundlagen zu Namensräumen

Sie erstellen Ihre XML-Datei und verweisen dabei, um die Struktur Ihres Dokumentes zu definieren, auf eine öffentliche, bereits bestehende Dokumenttyp-Definition. Da Sie in Ihrem Dokument noch einige zusätzliche Elemente verwenden möchten, müssen Sie diese Elemente innerhalb Ihres XML-Dokuments definieren.



Beim Einsatz von fremden externen XML-Daten kann es vorkommen, dass die Namen von Elementen doppelt vergeben werden. Sie können nicht sicherstellen, dass diese Dokumente nur Elemente verwenden, die Sie nicht in Ihrer eigenen XML-Datei definiert haben. Falls Sie dazu noch andere Attributvorgaben als in der DTD verwenden, würde dies zu einer Fehlermeldung beim Parsen des XML-Dokuments führen.



- ① In der externen Teilmenge der DTD wurde das Element `NAME` festgelegt, damit es in dieser Form genutzt werden kann.
- ② In Ihrer XML-Datei möchten Sie jedoch jeden Namen mit einer eindeutigen Anrede als Attribut kennzeichnen, sodass Sie zwischen Frau und Herr unterscheiden können. Dazu legen Sie in der internen Definition das Attribut `anrede` fest.

Wenn Sie das vorgestellte Beispiel verwenden, informiert Sie der Parser, dass das Dokument ungültig ist. Er teilt Ihnen mit, dass das Element `NAME` bereits in der externen DTD festgelegt wurde. Sie möchten jedoch nicht auf das Attribut `anrede` verzichten.

Zu diesem Zweck wurde am 14. Januar 1999 vom W3-Konsortium die Empfehlung zur Verwendung von sogenannten XML-Namensräumen (engl. XML Namespaces) veröffentlicht. Diese Spezifikation stellt sicher, dass die verwendeten Elementnamen eindeutig gekennzeichnet werden können. Namensräume können über eine URI identifiziert werden, also über eine eindeutige Bezeichnung. Die Angabe einer URI stellt sicher, dass der erstellte Namensraum auch wirklich eindeutig ist, z. B. eine Internetadresse.

Diese Datei oder Internetadresse muss nicht existieren, es kann ein beliebiger Fantasienamen sein. Hierbei geht es nur um die Eindeutigkeit der Bezeichnung.



6.2 Deklaration von Namensräumen

Jeder Namensraum muss deklariert werden, bevor Sie ihn verwenden können. Die Deklaration erfolgt über die Verwendung von reservierten Attributen. Der Attributname muss entweder `xmlns` (`xmlns` = XML Name-spaces) lauten oder es muss das Präfix `xmlns:` (mit dem Doppelpunkt) verwendet werden.

Voreingestellter Namensraum als Dateninsel

Einen Namensraum, der standardmäßig in einem XML-Dokument verwendet wird, deklarieren Sie über das Attribut `xmlns`. Dieser wird auch als Dateninsel bezeichnet.

```
<Hauptelement xmlns="URI">
  <Element>Daten</Element>
  <Element />
</Hauptelement>
```

Das Attribut `xmlns` leitet die Bezeichnung des Namensraums für das Element mit all seinen Unterelementen ein. Die URI stellt die Eindeutigkeit des Namensraums sicher und kann eine beliebige Bezeichnung sein.

Beispiel: *kap06\xmlns-html.xml*

Es wird ein Namensraum angelegt, der im gesamten XML-Dokument gültig ist und daher bei jedem Element angewendet werden kann. Für das Beispiel wird die Festlegung des HTML-Namensraums vorgenommen.

```

① <?xml version="1.0" encoding="ISO-8859-1"?>
② <html xmlns="http://www.w3.org/TR/REC-html40">
  <body >
    <title>Namensraum</title>
    <h2>Namensräume in XML</h2>
    
    <p align="center">Dies ist ein Absatz mit XML-Daten.</p>
    <p style="color:red;">Farbiger Absatz.</p>
    <a href="#">Dies ist ein Hyperlink in einem XML-Dokument.</a>
  </body>
③ </html>
```

- ① Für das XML-Dokument wird über das Attribut `xmlns` der Namensraum voreingestellt. Im Falle von HTML 4.0 sieht das W3-Konsortium die URI `http://www.w3.org/TR/REC-html40` vor. Alle Elemente, die zwischen `<html>` und `</html>` stehen, gehören damit zum HTML-Namensraum.
- ② Es folgen die einzelnen Elemente, die sich auf den HTML-Namensraum beziehen.
- ③ Der Namensraum wird mit dem schließenden Element `</html>` beendet.

Das XML-Dokument ist damit nicht automatisch ein HTML-Dokument, das dementsprechend bei der Anzeige im Browser formatiert wird. Die Elemente gehören nur dem HTML-Namensraum an. Eine URI muss nicht unbedingt eine tatsächliche Internetadresse sein.



Mehrere Dateninseln

Um einen Namensraum nicht für ein gesamtes Dokument, sondern nur für einzelne Elementstrukturen zu definieren, muss er in dem entsprechenden Unterelement vereinbart werden. Er findet dann auch auf die darin geschachtelten Elemente Anwendung.

Beispiel: *kap06\xmlns-all.xml*

Es wird ein Namensraum angelegt, der im gesamten XML-Dokument gültig ist und daher bei jedem Element angewendet werden kann. In einem Unterelement wird ein weiterer Namensraum angelegt.

	<code><?xml version="1.0" encoding="ISO-8859-1"?></code>
①	<code><VERWALTUNG xmlns="http://www.irgendeineURI.de"></code>
②	<code> <DATEN></code>
	<code> <NAME>Meier</NAME></code>
	<code> </DATEN></code>
③	<code> <DATEN xmlns="http://www.irgendeineURI.de/mit-anrede"></code>
	<code> <NAME anrede="Herr">Meier</NAME></code>
④	<code> </DATEN></code>
	<code></VERWALTUNG></code>

- ① Im Hauptelement `VERWALTUNG` ist eine Namensraumdeklaration enthalten. Dabei wird zur Eindeutigkeit auf die URI `http://www.irgendeineURI.de` Bezug genommen.
- ② Das Element `DATEN` wird mit seinem Unterelement `NAME` festgelegt.
- ③ Für das nächste Element `DATEN` wird ein neuer Namensraum deklariert, sodass im Unterelement zusätzlich das Attribut `anrede` angegeben werden kann, ohne gegen die Gültigkeit des XML-Dokuments zu verstoßen.
- ④ Mit dem schließenden Element `</DATEN>` ist wieder der erste Namensraum aktiv.

Namensräume mit qualifizierten Namen

Nicht immer ist es sinnvoll, einen Namensraum für mehrere Elemente gleichzeitig festzulegen. Hierfür definieren Sie einen Namensraum mit einem qualifizierten Namen, auch Präfix genannt. Dieses Präfix können Sie innerhalb der umschlossenen Elemente einsetzen, wenn Sie es benötigen.

```
<Hauptelement>
  <Element xmlns:Präfix="URI">
    <Präfix:Unterelement>Daten</Präfix:Unterelement>
    <Unterelement />
  </Element>
  <-- weitere Elemente -->
</Hauptelement>
```

Durch die Einbindung eines Namensraums am Element beziehen sich alle untergeordneten Elemente automatisch auf den angegebenen Namensraum. Präfixe sind Teil des Elementnamens und müssen immer an dem jeweiligen Element angegeben werden, um anzuzeigen, dass das Element zu dem entsprechenden Namensraum gehört. Das Präfix wird immer durch einen Doppelpunkt `:` vom Elementnamen getrennt.

Beispiel: *kap06\xmlns.xml*

Es wird ein Namensraum angelegt, der an einem XML-Element definiert wird und somit auch für dessen Unterelemente gültig ist.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<VERWALTUNG>
  <DATEN>
    <NAME>Meier</NAME>
  </DATEN>
  ① <DATEN xmlns:extra="http://www.irgendeineURI.de/mit-anrede">
    ② <extra:NAME anrede="Herr">Meier</extra:NAME>
    ③ </DATEN>
  </VERWALTUNG>
```

- ① Sie legen einen Namensraum für das Element `DATEN` und seine untergeordneten Elemente an. Um den Namensraum anzusprechen, erhält er die Bezeichnung `extra`.
- ② Damit Sie das zusätzliche Attribut `anrede` anwenden können, notieren Sie das Unterelement `NAME` mit dem Namensraum `extra`.
- ③ Das Element `DATEN` wird geschlossen. Damit wird auch der Namensraum `extra` beendet. Ein möglicherweise nachfolgendes Element `Name` kann nicht mehr das Attribut `anrede` besitzen.

Namensraum für ein Element

Sie können einen Namensraum auch einmalig verwenden. Dazu geben Sie ihn direkt an dem Element an.

Das Element wird mit der Bezeichnung des Namensraums eingeleitet. Innerhalb des Elements wird der Namensraum über die Angabe `xmlns` definiert. Beachten Sie hierbei, dass beim Schließen des Elements der Namensraum ebenfalls anzugeben ist.

```
<Hauptelement>
  <Präfix:Element xmlns:Präfix="URI">Daten</Präfix:Element>
  <Element />
</Hauptelement>
```

Beispiel: *kap06\xmlns-element.xml*

Es wird ein Namensraum angelegt, der nur für ein bestimmtes Element gültig ist und daher nicht bei anderen Elementen angewendet werden kann.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<VERWALTUNG>
  <DATEN>
    <NAME>Meier</NAME>
  </DATEN>
  <DATEN>
    ① <extra:NAME xmlns:extra="http://www.irgendeineURI.de/mit-anrede"
    ② <extra:NAME anrede="Herr">Meier</extra:NAME>
  </DATEN>
</VERWALTUNG>
```

- ① Der Namensraum `extra` wird vor dem Element `NAME` angegeben. Danach erfolgt die Festlegung des Namensraums mit der Bezeichnung `extra`.
- ② Das Element `NAME` wird mit der zusätzlichen Angabe des Namensraums `extra` wieder geschlossen.

Mehrere Namensräume mit qualifizierten Namen

Mehrere Namensräume mit qualifizierten Namen können Sie ebenfalls im Hauptelement eines XML-Dokuments definieren.

Beispiel: *kap06\xmlns-multiple.xml*

Im XML-Dokument sollen gleichzeitig Elemente mit unterschiedlichen Attributdefinitionen verwendet werden. Die Nutzung können Sie durch zwei verschiedene Namensräume ermöglichen. Die Namensräume werden im Hauptelement des XML-Dokuments deklariert.

	<code><?xml version="1.0" encoding="ISO-8859-1"?></code>
①	<code><standard:VERWALTUNG xmlns:standard="http://www.irgendeineURI.de/standard"</code>
	<code>xmlns:extra="http://www.irgendeineURI.de/extra"></code>
②	<code><standard:DATEN></code>
	<code><standard:NAME>Meier</standard:NAME></code>
	<code></standard:DATEN></code>
③	<code><extra:DATEN></code>
	<code><extra:NAME anrede="Herr">Meier</extra:NAME></code>
	<code></extra:DATEN></code>
	<code></standard:VERWALTUNG></code>

- ① Für das Dokument werden die beiden Namensräume `standard` und `extra` mit jeweils verschiedenen URLs festgelegt. Für die eindeutige Nutzung wird jedem Element das entsprechende Präfix vorangestellt.
- ② Das Element `DATEN` wird mit seinen Unterelementen als Standard-Variante gekennzeichnet.
- ③ Dieses Element `DATEN` wird dem Namensraum `extra` zugewiesen, und deshalb kann das Element `Name` z. B. das zusätzliche Attribut `anrede` enthalten.

6.3 Externe DTD und eigener Namensraum

Zur exakten Gültigkeit eines XML-Dokuments ist es notwendig, dass Sie die Definition der Dokumenttypen einbinden, indem Sie auf eine bereits bestehende DTD zugreifen. Damit Sie beispielsweise einem Element eigene, zusätzliche Attribute hinzufügen können, müssen Sie die bestehende Definition überschreiben und das neue Attribut definieren. Dazu müssen Sie einen Namensraum deklarieren.

Im nachfolgenden Beispiel binden Sie die DTD für die XML-Struktur einer Mitarbeiterverwaltung ein. Diese Struktur soll so erweitert werden, dass Sie neben dem Namen des Mitarbeiters als Attribut die entsprechende Anrede angeben können.

Beispiel: *kap06\verwaltung.dtd*

Der Inhalt der bisherigen DTD lautet folgendermaßen:

<code><!ELEMENT VERWALTUNG (DATEN+) ></code>
<code><!ELEMENT DATEN (NAME) ></code>
<code><!ELEMENT NAME (#PCDATA) ></code>

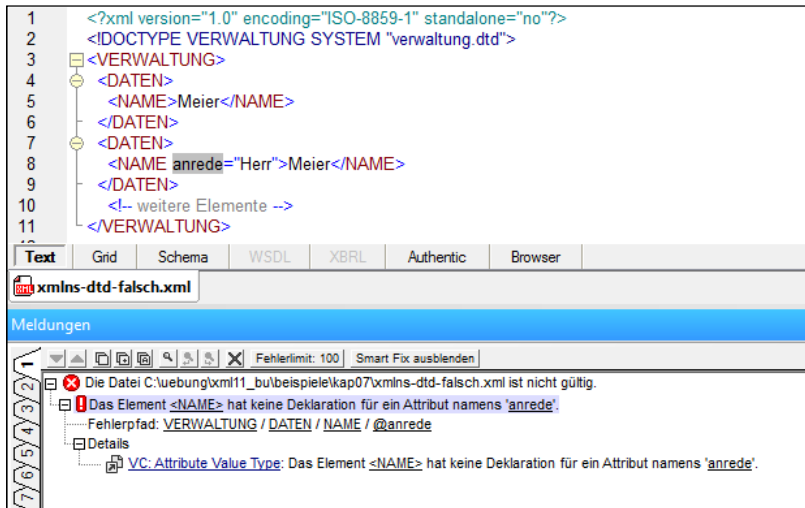
Hiermit wird die Struktur des XML-Dokuments festgelegt. Dem Element `NAME` ist standardmäßig kein Attribut `anrede` zugeordnet.

Beispiel: *kap06\xmlns-dtd-falsch.xml*

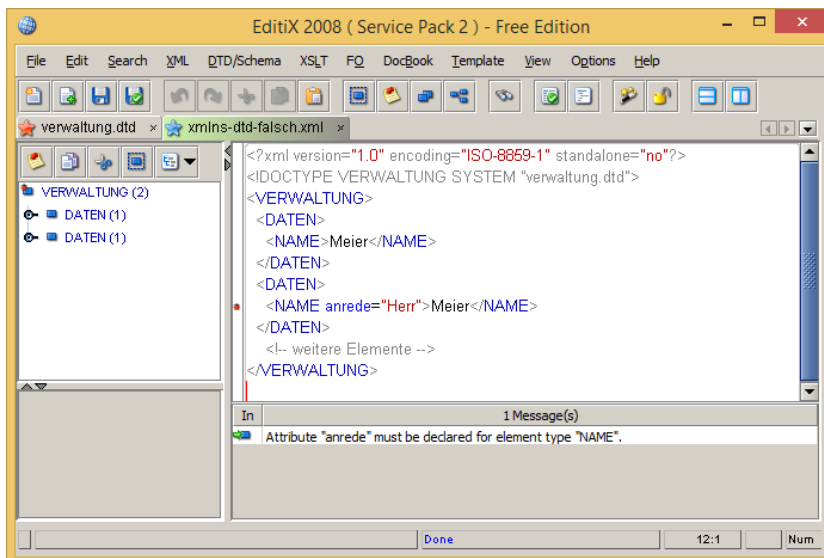
Wenn Sie die Datei *verwaltung.dtd* als externe DTD in Ihr Dokument einbinden, sind Sie gezwungen, die Strukturdefinition einzuhalten. Durch die Angabe des Attributs `anrede` wird das XML-Dokument aufgrund der DTD ungültig.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE VERWALTUNG SYSTEM "verwaltung.dtd">
<VERWALTUNG>
  <DATEN>
    <NAME>Meier</NAME>
  </DATEN>
  <DATEN>
    <NAME anrede="Herr">Meier</NAME>
  </DATEN>
</VERWALTUNG>
```

Wenn Sie die Datei mit dem Programm XMLSPY erfassen, zeigt dieses bei einer Überprüfung mittels *XML - XML validieren* den Fehler an, dass das Attribut `anrede` nicht in der DTD definiert ist.



Ausgabe, dass ein nicht definiertes Attribut verwendet wurde



Im Programm Editix wird der Fehler ebenfalls angezeigt

Das Problem kann mit dem Anlegen eines separaten Namensraums umgangen werden. Damit bei einer eingebundenen DTD die Definitionen überschrieben werden können, müssen Sie zusätzlich eine interne DTD erstellen. Damit das XML-Dokument gültig ist, müssen Sie den Attributnamen `xmlns` für den Namensraum in der DTD deklarieren.

Beispiel: *kap06\xmlns-dtd.xml*

Sie definieren einen Namensraum, der im Element `DATEN` angewendet werden kann, damit im Unterelement `NAME` die Angabe des Attributs `anrede` möglich ist.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="no"?></code>
①	<code><!DOCTYPE VERWALTUNG PUBLIC "-//Meine eigene DTD" "verwaltung.dtd" [</code>
②	<code><!ATTLIST DATEN xmlns:extra CDATA #FIXED "http://www.mitanrede.de"></code>
③	<code><!ATTLIST NAME extra:anrede (Herr Frau) #IMPLIED></code>
	<code>]></code>
	<code><VERWALTUNG></code>
	<code><DATEN></code>
	<code><NAME>Meier</NAME></code>
	<code></DATEN></code>
④	<code><DATEN xmlns:extra="http://www.mitanrede.de"></code>
⑤	<code><NAME extra:anrede="Herr">Meier</NAME></code>
	<code></DATEN></code>
	<code></VERWALTUNG></code>

- ① Über die Anweisung `DOCTYPE` binden Sie die externe Dokumenttyp-Definition *verwaltung.dtd* ein. Somit ist die Syntax eines Elements vorgegeben.
- ② Für das Element `DATEN` legen Sie die Attributliste für den Namensraum `extra` fest. Der Attributwert kann eine beliebige Zeichenfolge beinhalten (`CDATA`), die jedoch mit `http://www.mitanrede.de` festgelegt ist (`#FIXED`).
- ③ Dem Element `NAME` wird über den Namensraum `extra` das Attribut `anrede` hinzugefügt. Die Attributwerte sind mit `Herr` bzw. `Frau` vorgegeben. Die Angabe des Attributs ist aber keine Pflicht (`#IMPLIED`).
- ④ Im zweiten Element `DATEN` geben Sie das Präfix `xmlns` mit dem Namensraum `extra` an. Der Namensraum wird damit für alle weiteren Unterelemente aktiv.
- ⑤ Der Namensraum `extra` ermöglicht Ihnen damit trotz Nutzung einer vorgegebenen, externen DTD die Angabe des zusätzlichen Attributs `anrede`.

6.4 Übungen

Übung 1: Theoriefragen zu Namensräumen

Übungsdatei: --

Ergebnisdatei: *kap06uebung1-2.html*

- ① Erklären Sie, warum das Anlegen eines Namensraums bei der Verwendung mehrerer Dokumenttyp-Definitionen wichtig ist.
- ② Zählen Sie die verschiedenen Arten eines Namensraums auf.

Übung 2: Verwenden mehrerer Namensräume

Übungsdatei: --

Ergebnisdatei: *kap06uebung3.xml*

Fügen Sie im XML-Dokument mit den Seminardaten einen beliebigen Namensraum ein, der es dem Seminarleiter ermöglicht, für die Angabe seines Alters das Attribut `alter` zu hinterlegen. Diese Angabe soll nicht zwingend vorgeschrieben sein.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

7 XML Schema

In diesem Kapitel erfahren Sie

- ✓ was XML Schema ist
- ✓ wie XML Schema die bisherige DTD ersetzen kann
- ✓ wie Sie einfache Elemente und Attribute definieren
- ✓ welche Einschränkungen der Elementinhalte möglich sind
- ✓ welche verschiedenen Datentypen es gibt

Voraussetzungen

- ✓ Kenntnisse des Aufbaus von XML-Dokumenten
- ✓ Kenntnisse in der Anwendung von DTD

7.1 Der Unterschied zwischen Schema und DTD

Entwicklung XML Schema

Die bisherige Festlegung der möglichen XML-Elemente, Attribute und Inhalte über die Dokumenttyp-Definition (DTD) benutzt eine SGML-verwandte Syntax. Dies hat zur Folge, dass Sie die XML-Instanz in einem anderen Format als die Beschreibungsregeln definieren müssen. Außerdem unterstützen die DTDs nur wenige Datentypen. Somit können Sie über eine DTD nicht festlegen, dass der Wert eines bestimmten Attributs nur innerhalb eines definierten Bereichs liegen darf.

Bereits im Jahr 1999 wurden Anforderungen zur Modellierung von Schema für XML-Daten veröffentlicht (siehe <http://www.w3.org/TR/NOTE-xml-schema-req.html>). Seit 2001 gibt es mit der XML **Schema Definition Language** (XSD), kurz XML Schema, eine Empfehlung des W3-Konsortiums, um die Struktur und den Inhalt eines XML-Dokuments zu beschreiben. Das Erstellen eines Schemas erfolgt über die XML-Syntax. Schema ist somit ein eigenes XML-Dokument. Aufgrund der XML-Struktur werden die Beschreibungsregeln eines XML-Dokuments maschinenlesbar.

Mit XML Schema haben Sie Zugriff auf verschiedene Datentypen, wie z. B. `date`, `time`, `boolean`, `Float`, `decimal`, `hexBinary` usw. Zudem können Sie eigene Datentypen erstellen, die auf vorhandenen Datentypen basieren können. Es lassen sich in XML Schema auch Namensräume definieren, damit Elemente mit derselben Bezeichnung in verschiedenen Dokumenten verwendet werden können.

Seit Mai 2012 gibt es die Weiterentwicklung XML Schema 1.1. Das Ziel dieser Version ist in erster Linie, Schwachpunkte von XML Schema 1.0 zu beheben. Daneben stehen Änderungen im Mittelpunkt, welche die Erstellung von flexiblen Schemas unterstützen und die Anwendung von XML Schema generell erleichtern. Auf Grund der Abwärtskompatibilität validiert ein Prozessor für die Version 1.1 auch 1.0 Schema Dateien. Allerdings wird zum gegenwärtigen Zeitpunkt XML Schema 1.1 von vielen Tools noch nicht unterstützt. Der Abschnitt 8.5 gibt einen kurzen Überblick über einige Neuerungen in Schema 1.1.

Ausführliche Informationen zu XML Schema finden Sie im Internet unter <http://www.w3.org/XML/Schema>.

Die Unterschiede von Schema und DTD

Folgende Regeln können festgelegt werden:	DTD	Schema
Syntax eines Elements	✓	✓
Attribute eines Elements	✓	✓
mögliche Kindelemente	✓	✓
Standardwerte und feste Werte für Elemente und Attribute	✓	✓

Folgende Regeln können festgelegt werden:	DTD	Schema
Reihenfolge der Elemente	✓	✓
Anzahl der möglichen Elemente		✓
Datentyp der Elemente		✓
Datentyp der Attribute		✓
eigene Datentypen		✓

Es stellt sich die Frage, warum XML Schema nicht die DTDs ablöst. Die Schwächen der DTD können auch als deren Stärke angesehen werden. Abgesehen davon, dass für die DTD eine neue Syntax erlernt werden muss, ist diese sehr gut zu überschauen. Die DTD ist daher für nicht kritische Anwendungen attraktiv, bei denen z. B. kein besonderer Wert auf detaillierte Datentypen gelegt werden muss. Die Standardanwendung von XML Schema wird überwiegend der Datenaustausch zwischen Geschäftspartnern (B2B - Business to Business) sowie zwischen Firmen und Kunden (B2C - Business to Consumer) sein, bei denen eine genaue Definition von Datentypen und Wertebereichen von grundlegender Bedeutung ist.

Beispiel: *kap07\start.xml*

Eine einfache XML-Struktur mit Adressdaten hat beispielsweise den nachfolgenden Inhalt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Adresse>
  <Name>Herbert Hagedorn</Name>
  <Strasse>Hasenweg 13</Strasse>
  <Ort>Frankfurt/Main</Ort>
  <PLZ>60000</PLZ>
</Adresse>
```

Beispiel: *kap07\start.dtd*

Die entsprechende DTD für die XML-Instanz lautet:

```
<!ELEMENT Adresse (Name, Strasse, Ort, PLZ)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Strasse (#PCDATA)>
<!ELEMENT Ort (#PCDATA)>
<!ELEMENT PLZ (#PCDATA)>
```

Die erste Zeile legt fest, dass das Element *Adresse* vier Unterelemente besitzt. Die einzelnen Elemente sind vom Datentyp *#PCDATA*, also einer beliebigen Zeichenkette.

Beispiel: *kap07\start.xsd*

Das XML Schema für die Daten einer XML-Instanz sieht folgendermaßen aus:

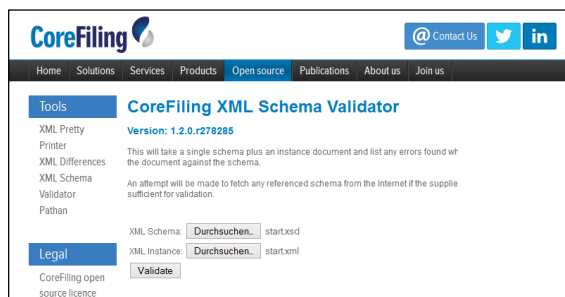
```
① <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
② <xs:element name="Adresse">
  <xs:complexType>
    <xs:sequence>
③   <xs:element name="Name" type="xs:string" />
      <xs:element name="Strasse" type="xs:string" />
      <xs:element name="Ort" type="xs:string" />
      <xs:element name="PLZ" type="xs:integer" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

- ① Für die Nutzung von Schema wird der Namensraum `xs` angelegt.
- ② Das Oberelement `Adresse` wird als komplexer Typ festgelegt, der eine bestimmte Reihenfolge von Elementen besitzt.
- ③ Die vier Unterelemente werden als Sequenz definiert. Drei sind einfache Zeichenketten vom Datentyp `string` und das Element `PLZ` ist vom Typ `integer`. Sie besitzen keine weiteren Unterelemente oder Attribute.

Schema-Definition validieren

Die Software Editix unterstützt Sie auch bei der Überprüfung von Schema-Definitionen. Wie bei DTD-Beschreibungen können Sie diese über das Menü *XML - Check for a well-formed / valid document* oder alternativ über **Strg** **K** überprüfen lassen.

Im Internet können Sie ebenfalls eine Vielzahl kostenloser Angebote zum Testen einzelner XML Schemata nutzen, z. B. das Angebot der Firma CoreFiling (ehemals DecisionSoft). Dieser Validator läuft auf dem Webserver des Anbieters. Zu finden ist das Angebot unter <http://www.corefiling.com/opensource/schemaValidate.html>. Auch hier geben Sie zur Überprüfung die Schema- sowie die XML-Datei an.



Online XML Schema Validator

7.2 Grundlagen zu XML Schema

Namensraum von XML Schema

Das Element `schema` ist das Hauptelement eines XML Schemas. Das Element kann folgende Attribute beinhalten:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="URI"
            targetNamespace="URI"
            elementFormDefault="qualified|unqualified"
            attributeFormDefault="qualified|unqualified">
...
</xs:schema>
```

- ✓ Das Attribut `xmlns:xs` besagt, dass die Elemente und Datentypen, die in diesem Schema genutzt werden, aus dem Namensraum `http://www.w3.org/2001/XMLSchema` kommen. Es legt auch fest, dass die Elemente und Datentypen dieses Namensraums mit dem Präfix `xs:` gekennzeichnet werden müssen. Das Präfix kann von Ihnen beliebig benannt werden. Mit `xmlns="URI"` geben Sie den standardmäßig zu verwendenden Namensraum an.
- ✓ Über das Attribut `targetNamespace="URI"` können Sie festlegen, in welchem Namensraum sich die Elemente in der XML-Instanz befinden müssen, damit die nachfolgende Definition angewendet wird.
- ✓ Ob jedem Element und jedem Attribut das Namensraum-Präfix vorangestellt werden muss, können Sie über die optionalen Attribute `elementFormDefault` und `attributeFormDefault` angeben. Standardmäßig ist jeweils der Wert `unqualified` eingestellt, sodass das Präfix nicht angegeben werden muss. Mit `qualified` erzwingen Sie einen qualifizierten Namen, also das Voranstellen des Präfixes.

Schema-Datei mit XML-Dokument verknüpfen

```
<Hauptelement
  xmlns="URI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="Namensraum Schema-Datei">
  ...
</Hauptelement>
```

- ✓ In einer XML-Datei definieren Sie mit dem Attribut `xmlns` einen standardmäßigen Namensraum, dem alle Elemente des Dokuments angehören, denen kein Präfix vorangestellt ist. Außerdem wird der Namensraum `xsi` für die XML-Schema-Instanz angegeben. Genau wie das Präfix `xs` für das XML Schema hat sich hierfür die Angabe von `xsi` durchgesetzt.
- ✓ Das Attribut `schemaLocation` mit dem vorangestellten Namensraum besitzt einen Wert, der aus zwei Teilen besteht. Sie geben damit dem Parser einen Hinweis, wo er für den angegebenen Namensraum eine geeignete Schema-Datei finden kann. Der Parser ist jedoch nicht unbedingt an diese Schema-Datei gebunden.
- ✓ Die Schema-Datei besitzt entsprechend der Konvention immer den Dateityp `.xsd`.

Verwenden Sie in Ihrem XML-Dokument keinen Namensraum, haben Sie die Möglichkeit, die Schema-Datei über das Attribut `xsi:noNamespaceSchemaLocation` einzubinden. Als Wert wird nur die absolute oder relative Angabe der Schema-Datei erwartet. Die Angabe von `xsi:schemaLocation` entfällt dementsprechend.

```
<Hauptelement
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Schema-Datei">
  ...
</Hauptelement>
```

Beispiel: *kap07/adresse-start.xml*

Die Einbindung der bereits erstellten Schema-Datei *start.xsd* erfolgt über die Angabe des entsprechenden Namensraums.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
① <Adresse
②   xmlns="http://www.herdt.com"
③   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
④   xsi:schemaLocation="http://www.herdt.com start.xsd">
⑤   <Name>Herbert Hagedorn</Name>
   <Strasse>Hasenweg 13</Strasse>
   <Ort>Frankfurt/Main</Ort>
   <PLZ>60000</PLZ>
</Adresse>
```

- ① Die Angaben zur einzubindenden Schema-Datei erfolgen hinter dem Hauptelement der XML-Instanz.
- ② Zuerst wird der Standard-Namensraum für das gesamte Dokument festgelegt. Hier ist es der Namensraum `http://www.herdt.com`.
- ③ Der Namensraum der XML-Instanz wird über die angegebene URI definiert. Das Präfix `xsi` steht für die XML-Schema-Instanz und kann von Ihnen beliebig benannt werden.
- ④ Für den Namensraum `xsi` wird die Schema-Datei *start.xsd* empfohlen, die über das Schlüsselwort `schemaLocation` angegeben wird.
- ⑤ Es folgen die Elemente der XML-Instanz.

► Öffnen Sie das XML-Dokument im Browser.

Der Browser prüft das Dokument nur auf Wohlgeformtheit. Mögliche Fehler in der Schema-Datei werden Ihnen nicht angezeigt. Somit können Sie nicht mit hundertprozentiger Sicherheit sagen, dass die XML-Datei korrekt (valid) ist. Hierfür benötigen Sie die bereits erwähnten Parser, die XML Schema verarbeiten und auswerten können.

7.3 Schema-Grundgerüst

Bevor Sie die einfachen oder komplexen Elemente, die Attributangaben, die möglichen Datentypen und die Möglichkeiten zur Einschränkung von Daten kennenlernen, erhalten Sie einen Einblick in die bisherige Dokumenttyp-Definition und Sie erfahren, wie diese Definition jetzt über Schema realisiert wird.

Beispiel: *kap07/labfolge.xsd*

Das folgende Beispiel zeigt Ihnen die grundlegenden Elemente des Schemas, die bei einer Definition erwartet werden.

```

① <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
②   <xs:element name="Hauptelement">
③     <xs:complexType>
④       <xs:sequence>
⑤         <xs:element name="Unterelement1" />
⑥         <xs:element name="Unterelement2" />
⑦       </xs:sequence>
⑧     </xs:complexType>
⑨   </xs:element>
</xs:schema>

```

- ①,⑨ Das Attribut `xmlns` des Elements `schema` legt über das Präfix `xs` den Namensraum der Elemente fest. Mit der Zeit haben sich die Abkürzungen `xs`: bzw. `xsd`: als Standardangabe durchgesetzt.
- ②,⑧ Mit dem Element `element` hinterlegen Sie den Namen des möglichen Elements, das im XML-Dokument erwartet wird.
- ③,⑦ Die Angabe weiterer Unterelemente wird als komplexer Typ angesehen und deshalb mit dem Element `complexType` eingeleitet.
- ④,⑥ Wenn sich mehrere Elemente unterhalb eines Elements befinden können, werden diese über das Element `sequence` eingeleitet. Damit geben Sie an, dass die nachfolgenden Elemente in der angegebenen Reihenfolge auftreten müssen.
- ⑤ Analog zu Punkt ② geben Sie über `element name` die Namen der untergeordneten Elemente an.

Mit diesem Beispiel haben Sie eine Struktur definiert, mit der in einem XML-Dokument das Element `Hauptelement` mit den beiden Unterelementen `Unterelement1` und `Unterelement2` erwartet wird.

Beispiel: *kap07/bestell.xml*

Ein Warenkorbsystem einer beliebigen Webseite im Internet generiert zum Beispiel die nachfolgenden Bestell-Informationen in Form einer XML-Datei.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<Bestellung bestelldatum="2014-09-27">
  <Adresse>
    <Name>Herbert Hagedorn</Name>
    <Strasse>Hasenweg 13</Strasse>
    <Ort>Frankfurt/Main</Ort>
    <PLZ>60000</PLZ>
  </Adresse>
  <Ware>
    <Buch BestellNr="3453865219">
      <Titel>Wolfsmond. Der dunkle Turm.</Titel>
      <Autor>Stephen King</Autor>
      <Anzahl>1</Anzahl>
      <PreisEUR>15.00</PreisEUR>
    </Buch>
  </Ware>
</Bestellung>

```

Beispiel: *kap07\bestell.dtd*

Eine DTD, die diese Struktur beschreibt, könnte beispielsweise so aussehen:

```
<!ELEMENT Bestellung (Adresse, Ware)>
<!ATTLIST Bestellung bestelldatum CDATA #REQUIRED>

<!ELEMENT Adresse (Name, Strasse, Ort, PLZ)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Strasse (#PCDATA)>
<!ELEMENT Ort (#PCDATA)>
<!ELEMENT PLZ (#PCDATA)>

<!ELEMENT Ware (Buch+)>
<!ELEMENT Buch (Titel, Autor, Anzahl, PreisEUR)>
<!ATTLIST Buch BestellNr CDATA #REQUIRED>
<!ELEMENT Titel (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
<!ELEMENT Anzahl (#PCDATA)>
<!ELEMENT PreisEUR (#PCDATA)>
```

Mit dieser Definition ist die grundlegende Struktur der XML-Datei festgelegt.

Sie haben jedoch keine Möglichkeit, Einfluss auf die angegebenen Werte zu nehmen. So können Sie z. B. über eine DTD nicht sicherstellen, dass das Datum immer im richtigen Format *jjjj-mm-tt* (Jahr, Monat, Tag) angegeben wird. Theoretisch könnte auch das Datum 27.09.2014 angegeben werden. Der Parser kann dies nicht unterscheiden. Außerdem soll in diesem Beispiel das Aussehen einer Bestellnummer standardisiert sein. Solch eine Nummer muss aus zehn Ziffern bestehen. Die richtige Angabe der Bestellnummer ist mit dieser DTD auch nicht sichergestellt.

Mit XML Schema können Sie diese Vorschriften definieren. Sie können sogar festlegen, dass pro Bestellung nur eine maximale Anzahl an Büchern möglich sein soll. Dies können Sie dann mithilfe eines Validators prüfen.

Beispiel: *kap07\bestell.xsd*

Die DTD wird in das XML-Schema konvertiert und die angegebenen Bedingungen bezüglich Datum und Bestellnummer werden hinzugefügt.

```
① <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
② <xs:element name="Bestellung">
  <xs:complexType>
③    <xs:sequence>
      <xs:element name="Adresse" type="BestellAdresse" />
      <xs:element name="Ware" type="WarenTyp" />
    </xs:sequence>
④    <xs:attribute name="bestelldatum" type="xs:date" />
  </xs:complexType>
</xs:element>

⑤ <xs:complexType name="BestellAdresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Strasse" type="xs:string" />
    <xs:element name="Ort" type="xs:string" />
    <xs:element name="PLZ" type="xs:integer" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="WarenTyp">
  <xs:sequence>
⑥    <xs:element name="Buch" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
```

```

<xs:sequence>
  <xs:element name="Titel" type="xs:string" />
  <xs:element name="Autor" type="xs:string" />
  <xs:element name="Anzahl">
    <xs:simpleType>
      <xs:restriction base="xs:positiveInteger">
        <xs:maxInclusive value="10" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="PreisEUR" type="xs:decimal" />
</xs:sequence>
<xs:attribute name="BestellNr" type="BestellNrTyp" use="required" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

<xs:simpleType name="BestellNrTyp">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{10}" />
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

- ① Jedes Element hat das Präfix `xs:`, das über `xmlns:xs="http://www.w3.org/2001/XMLSchema"` mit dem XML Schema Namensraum assoziiert wird.
- ② Es wird das Hauptelement `Bestellung` definiert.
- ③ Dieses besitzt eine Folge von weiteren Elementen (Sequence). Die Elemente `Adresse` und `Ware` sind von einem selbstdefinierten Datentyp.
- ④ Das Attribut `bestelldatum`, das dem Element `Bestellung` zugeordnet wird, ist vom Datentyp `date` und wird somit als Datum betrachtet.
- ⑤ Der eigene Datentyp `BestellAdresse` wird definiert. Dieser besteht aus einer Reihe von weiteren Elementen, wobei deren Reihenfolge durch `sequence` vorgegeben wird.
- ⑥ Im eigenen Datentyp `WarenTyp` wird über das Attribut `maxOccurs` festgelegt, dass das Element `Buch` in der XML-Struktur mehrmals vorkommen kann.
- ⑦ Für das Element `Anzahl` wird über das Element `restriction` eine positive Ganzzahl definiert, die maximal den Wert 10 haben kann.
- ⑧ Der Preis des Buches wird über den dezimalen Datentyp für das Element `PreisEUR` festgelegt.
- ⑨ Ein Buch muss das Attribut `BestellNr` besitzen, das dem Datentyp `BestellNrTyp` entspricht.
- ⑩ Wie der Datentyp `BestellNrTyp` aufgebaut ist, wird hier festgelegt.
- ⑪ Über das Element `pattern` und den regulären Ausdruck wird die notwendige Form der Bestellnummer festgelegt. In diesem Fall wird eine 10-stellige Zahl erwartet.

7.4 Einfache Typen

Einfache Elemente

Ein einfaches Element ist ein XML-Element, das keine weiteren Attribute und Elemente enthalten kann. Der Inhalt des Elements kann von einem beliebigen Datentyp sein, z. B. `boolean`, `string`, `integer`, `date` usw.

```
<xs:element name="Elementname" type="xs:Typ" default="Wert" fixed="Wert" />
```

Ein einfaches Element definieren Sie über die Angabe des Elements `xs:element` und das Attribut `name`. Den Datentyp, aus dem der Inhalt des Elements bestehen muss, legen Sie über das Attribut `type` fest. Die meistgenutzten Datentypen sind:

- ✓ `xs:string` für Zeichenketten, z. B. "Schema ist recht umfangreich"
- ✓ `xs:decimal` für Dezimalzahlen, z. B. 1.4, 3.14, -456.78901
- ✓ `xs:integer` für Ganzzahlen, z. B. 1, -2, 456, 1000
- ✓ `xs:boolean` für die Wahrheitswerte `true` oder `false`
- ✓ `xs:date` für englischsprachig formatierte Datumsangaben, z. B. 2014-09-27 oder 14-09-27
- ✓ `xs:time` für Zeitangaben, z. B. 11:37:52, 11:37:52+01:00

Über das Element-Attribut `default` können Sie einen Standardwert des Elements festlegen, der automatisch eingesetzt werden soll, wenn der Wert in der XML-Instanz nicht angegeben wird. Mit `fixed` können Sie sogar bestimmen, dass nur dieser bestimmte Wert eingesetzt werden soll und sonst kein anderer.

Beispiel

```
<xs:element name="Name" type="xs:string" />
<xs:element name="PLZ" type="xs:integer" />
<xs:element name="Land" type="xs:string" default="Deutschland" />
<xs:element name="Geburtstag" type="xs:date" />
<xs:element name="Alter" type="xs:integer" />
```

Das Element `Name` wird als Zeichenkette und `PLZ` als Zahl erwartet. Das Element `Land` ist ebenfalls eine Zeichenkette, wobei der Wert `Deutschland` automatisch eingesetzt wird, wenn im XML-Dokument kein Elementwert angegeben wird. Der Geburtstag kann nur als Datum angegeben werden, wobei die englische Schreibweise zu beachten ist. Für das Element `Alter` muss eine Ganzzahl angegeben werden.

Attribute

Einfache Elemente können keine Attribute besitzen. Soll ein Element ein Attribut erhalten, so muss es ein komplexes Element sein. Das Attribut selbst wird innerhalb des komplexen Elements als einfaches Element deklariert. Die Syntax für die Festlegung eines Attributs lautet:

```
<xs:attribute name="Attributname" type="xs:Datentyp" default="Wert"
fixed="Wert" use="required|optional|prohibited" />
```

- ✓ Den Namen des Attributs legen Sie über das Schlüsselwort `name` fest. Der Datentyp wird, wie beim einfachen Element, über `type` angegeben. Für die Werte von `default` und `fixed` gilt dasselbe wie bei den einfachen Elementen.
- ✓ Zusätzlich gibt es das Schlüsselwort `use`, mit dem Sie festlegen können, ob das Attribut des Elements angegeben werden muss. Mögliche Werte sind `required` (Pflicht), `optional` (wahlweise) bzw. `prohibited` (verboten).

Die Definition von Attributen darf in Schema-Dateien nur am Ende eines komplexen Elements, direkt vor dem schließenden Tag `</xs:element>`, angegeben werden.



Beispiel

Dies ist ein Element mit einem Attribut, das den Namen bezüglich des Geschlechts der Person näher beschreibt.

```
<Name geschlecht="w">Ulli Werner</Name>
```

Die mögliche Attributdefinition dazu lautet:

```
<xs:attribute name="geschlecht" type="xs:string" use="required" />
```

Elemente einschränken

Die Einschränkungen (engl. Restrictions) werden genutzt, um für XML-Elemente und Attribute spezielle Werte vorzuschreiben bzw. die Auswahl einzuengen. Einschränkungen an XML-Elementen werden **Fassetten** (engl. Facets) genannt. Die Festlegungen werden vom Element `restriction` umschlossen.

```
<xs:restriction base="Datentyp">
```

Mit dem Attribut `base` legen Sie den Datentyp fest, von dem das Element abgeleitet werden soll. Wenn das Element eine bestimmte Ganzzahl enthalten soll, dann erbt es die Haupteigenschaften des Datentyps `Integer`. Eine bestimmte Zeichenkette wird grundlegend vom Datentyp `String` abgeleitet.

Einschränkung auf Wertebereiche

```
<xs:restriction base="Datentyp">
  <xs:minInclusive value="Wert" />  <!-- angegebener Wert inklusive -->
  <xs:maxInclusive value="Wert" />
</xs:restriction>

<xs:restriction base="Datentyp">
  <xs:minExclusive value="Wert" />  <!-- Wert nicht inklusive -->
  <xs:maxExclusive value="Wert" />
</xs:restriction>
```

Innerhalb der Festlegung `restriction` definieren Sie den Wertebereich des Elements. Beim Verwenden von `minInclusive` und `maxInclusive` gehören die angegebenen Werte mit zu dem möglichen Bereich, wogegen bei `minExclusive` und `maxExclusive` die Werte ausgeschlossen werden.

Beispiel: *kap07\alter.xsd*

In einer XML-Datei sind zu den möglichen Personenangaben auch Angaben zum Alter enthalten. Menschen können nicht jünger als 0 Jahre sein und in den seltensten Fällen älter als 120 werden. Also soll z. B. ein Element `Alter` nur ganze Zahlen zwischen 0 und 120 enthalten.

```
① <xs:element name="Alter">
②   <xs:simpleType>
③     <xs:restriction base="xs:integer">
④       <xs:minInclusive value="0" />
⑤       <xs:maxInclusive value="120" />
     </xs:restriction>
   </xs:simpleType>
</xs:element>
```

- ① Das Element `Alter`, dessen Inhalte eingeschränkt werden sollen, umschließt die gesamte Definition.
- ② Da das Element keine weiteren Elemente oder Attribute enthält, ist es vom einfachen Typ und wird dementsprechend mit `simpleType` gekennzeichnet.
- ③ Mit `restriction` leiten Sie die Einschränkung ein und legen fest, dass der Inhalt vom Datentyp `integer` abgeleitet wird.
- ④ Das Element `minInclusive` beschreibt den unteren Wert, wobei der angegebene Wert ebenfalls möglich ist (inklusive).
- ⑤ Mit `maxInclusive` legen Sie den oberen Wert fest.

Einschränkung auf Werte

```
<xs:restriction base="Datentyp">
  <xs:enumeration value="Wert1" />
  <xs:enumeration value="Wert2" />
</xs:restriction>
```


Mit dem Element `enumeration` legen Sie mögliche Elementinhalte fest. Die Werte sind Zeichenketten, welche die mögliche Auswahl darstellen.

Beispiel: *kap07/land.xsd*

Bei Länderangaben möchten Sie z. B. sichergehen, dass die Inhalte nur die Länder beinhalten, in denen Deutsch gesprochen wird. Dann geben Sie die Zeichenketten vor, die verwendet werden dürfen.

```

① <xs:element name="Land">
    <xs:simpleType>
②     <xs:restriction base="xs:string">
③         <xs:enumeration value="Deutschland" />
            <xs:enumeration value="Österreich" />
            <xs:enumeration value="Schweiz" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>

```

- ① Die folgende Einschränkung bezieht sich auf das Element `Land`.
- ② Die Elementwerte sind vom Datentyp `string`, also Zeichenketten.
- ③ Die möglichen Elementinhalte, die in einer XML-Instanz vorkommen können, werden mit dem Element `enumeration` über die Angabe von `value` einzeln aufgelistet.

Einschränkung auf eine Serie von Werten

```

<xs:restriction base="Datentyp">
    <xs:pattern value="Wert1" />
</xs:restriction>

```

Das Element `pattern` erlaubt es Ihnen, über festgelegte Muster die zulässigen Werte eines Datentyps einzuschränken. Die Muster werden über sogenannte regular Expressions oder reguläre Ausdrücke definiert.

Für einen kompakten Einstieg in das komplexe Thema der regulären Ausdrücke ist der Wikipedia-Artikel http://de.wikipedia.org/wiki/Regulärer_Ausdruck zu empfehlen.



Als Beispiel folgt eine Auflistung verschiedener regulärer Ausdrücke und deren mögliche Ergebnismengen.

<code><xs:pattern value="..."></code>	Ergebnis
<code>[0-9]</code>	Eine einstellige Zahl zwischen 0 und 9
<code>[0-9][0-9]</code>	Eine zweistellige Zahl wie 11, 52 oder 99
<code>[a-f]</code>	Ein Kleinbuchstabe, entweder a, b, c, d, e oder f
<code>[xyz]</code>	Einer von den drei Kleinbuchstaben, entweder x, y oder z
<code>[a-zA-Z][0-9][a-zA-Z]</code>	Eine dreistellige Buchstaben- und Ziffernkombination wie a1b, F9E
<code>([a-z])*</code>	Wörter mit einer beliebigen Anzahl von Kleinbuchstaben, auch keine Angabe ist möglich
<code>([a-zA-Z])+</code>	Wörter mit einer beliebigen Anzahl von Buchstaben, aber mindestens einem Buchstaben
<code>Herr Frau</code>	Entweder das Wort "Herr" oder "Frau"
<code>[a-zA-Z0-9]{10}</code>	Angabe eines 10-stelligen Wortes, das aus Klein- und Großbuchstaben sowie aus Zahlen bestehen kann, z. B. ein 10-stelliges Passwort

Beispiel: *kap07\bestellnummer.xsd*

In einer XML-Datei sollen die Artikelnummern von Produkten hinterlegt werden. Artikelnummern haben bei vielen Firmen eine festgelegte Struktur, z. B. 123-C-45. Die möglichen Werte werden über reguläre Ausdrücke festgelegt.

```

① <xs:element name="Bestellnummer" type="BestellnummerTyp" />
② <xs:simpleType name="BestellnummerTyp">
③   <xs:restriction base="xs:string">
④     <xs:pattern value="1[0-9][0-9]-[A-E]-[0-9][0-9]" />
   </xs:restriction>
</xs:simpleType>

```

- ① Das Element `Bestellnummer` umschließt die Festlegung nicht. Der Typ wird direkt über das Attribut `type` zugewiesen. Der Typ `BestellnummerTyp` wird nachfolgend festgelegt.
- ② Über die Angabe des Attributs `name` legen Sie fest, auf welchen Typ sich die nachfolgende Definition bezieht. Hier wird über die Angabe von `BestellnummerTyp` die Einschränkung des Elements `Bestellnummer` festgelegt. Der Datentyp kann auch für weitere Elemente verwendet werden.
- ③ Die Werte sind vom Datentyp `string`, also Zeichenketten.
- ④ Der mögliche Elementinhalt wird über das Element `pattern` definiert. Jedes einzelne Feld `[]` legt fest, welche Zeichen an dieser Stelle stehen dürfen. Es wird festgelegt, dass als erstes Zeichen eine 1 stehen muss, danach darf ein Zeichen folgen, das eine Zahl zwischen 0 und 9 ist `[0-9]`; ebenso an der dritten Stelle. Dann muss ein Bindestrich gesetzt sein, gefolgt von einem der Zeichen A, B, C, D oder E. Zum Schluss müssen wieder zwei Ziffern stehen, denen auch ein Bindestrich vorangestellt ist. Diese Festlegung erlaubt beispielsweise die folgenden Bestellnummern: 123-C-45 oder 195-A-11 oder 111-D-00.

Einschränkung nicht druckbarer Zeichen

```

<xs:restriction base="Datentyp">
  <xs:whitespace value="preserve|replace|collapse" />
</xs:restriction>

```

Mit dieser Einschränkung können Sie die nicht druckbaren Zeichen (engl. whitespaces) beeinflussen. Mit nicht druckbaren Zeichen sind Tabulatoren, Leerzeichen, Zeilenumbrüche und -vorschübe gemeint. Das Schlüsselwort hierfür lautet `whitespace` mit einem der drei Werte `preserve` (keine Veränderung), `replace` (Zeichen werden durch Leerzeichen ersetzt) und `collapse`. Der letzte Wert hat dieselbe Wirkung wie `replace`, nur dass zusätzlich noch mehrfache Leerzeichen zu einem Leerzeichen zusammengefasst werden und am Anfang und am Ende des Elementinhalts entfernt werden.

Einschränkung der Länge einer Zeichenkette

```

<xs:restriction base="Datentyp">
  <xs:length value="Längenangabe" />
</xs:restriction>

<xs:restriction base="Datentyp">
  <xs:minLength value="Längenangabe" />
  <xs:maxLength value="Längenangabe" />
</xs:restriction>

```

Die Länge eines Elementwertes legen Sie über `length` fest. Dann muss der Wert die angegebene Anzahl Zeichen aufweisen. Einen variablen Bereich von Zeichenlängen definieren Sie mit `minLength` (Mindestlänge) und `maxLength` (Maximallänge). Geben Sie nur `maxLength` an, hat `minLength` automatisch den Wert 0. Beim fehlenden Element `maxLength` muss zwar die Zeichenanzahl mindestens `minLength` sein, hat aber keine maximale Begrenzung.



Die Angabe von `length` ist nur bei Daten vom Typ `string` möglich. Bei Daten vom Typ `integer` oder `decimal` schränken Sie den Bereich über `minExclusive` und `maxExclusive` ein.

Beispiel: *kap07/plz-vorwahl.xsd*

Wenn Adressdaten in einer XML-Datei gesichert werden, wird auch die Postleitzahl angegeben. In Deutschland besteht die Postleitzahl grundsätzlich aus fünf Ziffern. Dies können Sie über das Schlüsselwort `length` festlegen. Soll ein Elementinhalt innerhalb einer minimalen und maximalen Zeichenlänge liegen, so wird dies über `minLength` und `maxLength` definiert. Ein Beispiel dafür sind Telefon-Vorwahlnummern, wie 030 für Berlin oder 06221 für Heidelberg.

```

① <xs:element name="PLZ" type="PLZTyp" />
② <xs:simpleType name="PLZTyp">
③   <xs:restriction base="xs:string">
     <xs:length value="5" />
   </xs:restriction>
  </xs:simpleType>

  <xs:element name="Vorwahl">
    <xs:simpleType>
      <xs:restriction base="xs:string">
④       <xs:minLength value="3" />
         <xs:maxLength value="5" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

```

- ① Für das Element PLZ wird der Typ PLZTyp festgelegt.
- ② Dieser Typ ist in diesem Beispiel vom Datentyp `string`.
- ③ Der Elementinhalt muss fünf Stellen lang sein. Hier ist die Werteinschränkung auch über einen regulären Ausdruck möglich, wie z. B. `<xs:pattern value="\d{5}" />`.
- ④ Das Element `Vorwahl` ist ebenfalls vom Datentyp `string`. Es hat jedoch mindestens drei Zeichen (`minLength`) und maximal fünf Zeichen (`maxLength`) lang zu sein. Auch hier ist die alternative Angabe von `<xs:pattern value="\d{3,5}" />` möglich.

Einschränkung der Genauigkeit von Dezimalzahlen

```

<xs:restriction base="Datentyp">
  <xs:fractionDigits value="Längenangabe" />
</xs:restriction>

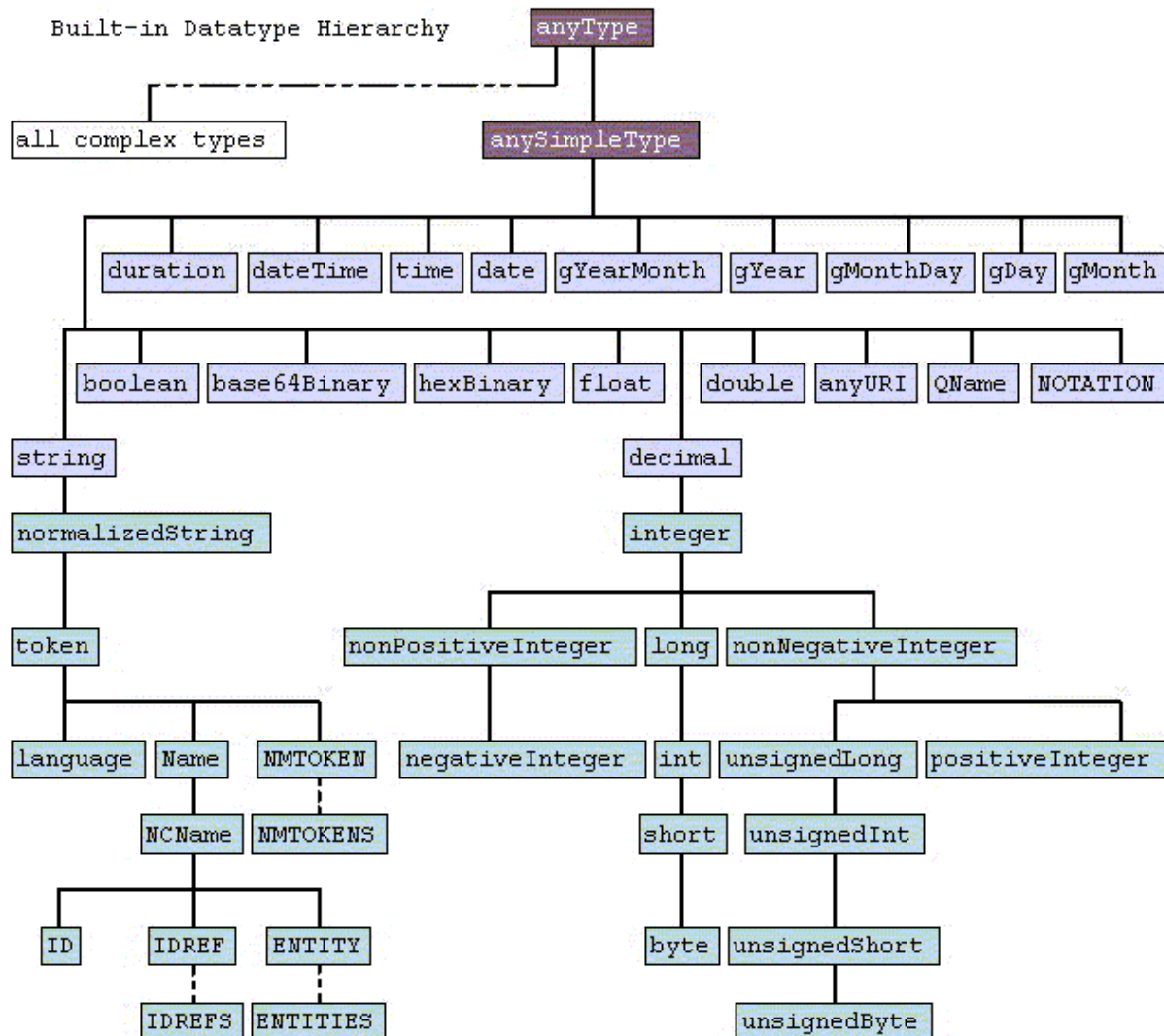
<xs:restriction base="Datentyp">
  <xs:totalDigits value="Längenangabe" />
</xs:restriction>

```

Des Weiteren gibt es die Einschränkung `fractionDigits`, mit der Sie die Anzahl der Nachkommastellen von Dezimalzahlen festlegen können. Der Wert muss größer oder gleich 0 sein. Die mögliche Gesamtanzahl eines Wertes bestimmen Sie über `totalDigits`. Als Wert wird hier eine Zahl größer 0 erwartet.

7.5 Datentypen

Werte von Elementen sind von einem bestimmten Typ, z. B. Ganzzahl, Dezimalzahl, Zeichenkette usw. Diese Typen werden Datentypen genannt.



Übersicht von W3.org

XML Schema stellt Ihnen standardmäßig eine Vielzahl von Datentypen zur Verfügung. Zusätzlich können Sie eigene Datentypen entwerfen. In diesem Abschnitt werden Ihnen die häufigsten Datentypen näher erläutert.

Datentyp string

Dieser Datentyp wird für Werte eingesetzt, die aus Zeichenketten bestehen. Dabei gibt es drei Unterscheidungen: `string`, `normalizedString` und `token`. Folgende Unterschiede weisen diese Datentypen auf:

string	Enthält Zeichen, Zeilenvorschübe, Zeilenumbrüche und Tabulatoren
normalizedString	Wie <code>string</code> , wobei der XML-Prozessor die Zeilenvorschübe, Zeilenumbrüche und Tabulatoren automatisch entfernt
token	Wie <code>normalizedString</code> , wobei der XML-Prozessor zusätzlich führende, anhängende und mehrfache Leerzeichen entfernt

Einschränkungen beim Datentyp `string`

Folgende Einschränkungen können beim Datentyp `string` angewendet werden:

- ✓ `enumeration`
- ✓ `maxLength`, `minLength`
- ✓ `whiteSpace`
- ✓ `length`
- ✓ `pattern`

Beispiel

```
<xs:element name="Name" type="xs:token" maxLength="100" />
```

Das Element `Name` ist vom Datentyp `token`, sodass der XML-Prozessor führende, anhängende und mehrfache Leerzeichen aus dem Elementinhalt entfernt. Die Angabe des Attributs `maxLength` schränkt die Länge des Inhalts auf maximal 100 Zeichen ein.

Numerische Datentypen

Dieser Datentyp wird eingesetzt, wenn Zahlen als Elementinhalte oder Attributwerte erwartet werden. Grundlegend gibt es zwei numerische Datentypen: die Dezimalzahlen sowie die Ganzzahlen. Der Datentyp `decimal` ist der allgemeine Datentyp und `integer` ist davon abgeleitet. Diese beiden Datentypen werden sehr häufig eingesetzt.

decimal	Festlegung von Fließkommazahlen, z. B. -12,345 oder 3,14
integer	Ist die Ableitung von <code>decimal</code> und erlaubt nur ganze Zahlen, z. B. -12 oder 314

Über den Datentyp `decimal` sind Dezimalzahlen mit einer Genauigkeit von 18 Nachkommastellen möglich. Bei beiden spielt es keine Rolle, ob es positive oder negative Zahlen sind. Der Datentyp `integer` stellt eine unendlich große Zahl dar.

Die Tabelle listet weitere numerische Datentypen auf, die vom Datentyp `decimal` abgeleitet sind.

Name	Wertebereich
<code>long</code>	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
<code>unsignedLong</code>	0 bis 18446744073709551615
<code>int</code>	-2.147.483.648 bis 2.147.483.647
<code>unsignedInt</code>	0 bis 4294967295
<code>short</code>	-32.768 bis 32.767
<code>unsignedShort</code>	0 bis 65535
<code>byte</code>	-128 bis 127
<code>unsignedByte</code>	0 bis 255
<code>negativeInteger</code>	integer mit negativen Werten (... , -2, -1)
<code>positiveInteger</code>	integer mit positiven Werten (1, 2, ...)
<code>nonNegativeInteger</code>	integer mit nicht negativen Werten (0, 1, 2, ...)
<code>nonPositiveInteger</code>	integer mit nicht positiven Werten (... , -2, -1, 0)

Einschränkungen bei numerischen Datentypen

Folgende Einschränkungen können bei den numerischen Datentypen angewendet werden:

- ✓ `enumeration`
- ✓ `totalDigits`, `fractionDigits`
- ✓ `whiteSpace`
- ✓ `maxInclusive`, `minInclusive`
- ✓ `maxExclusive`, `minExclusive`
- ✓ `pattern`

Beispiel: *kap07\numerisch.xsd*

```

① <xs:element name="Preis">
    <xs:simpleType>
        <xs:restriction base="xs:decimal">
            <xs:fractionDigits value="2" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>
② <xs:element name="Anzahl" type="xs:nonNegativeInteger" />
③ <xs:element name="Grad">
    <xs:simpleType>
        <xs:restriction base="xs:decimal">
            <xs:fractionDigits value="1" />
            <xs:minInclusive value="-90.0" />
            <xs:maxInclusive value="60.0" />
        </xs:restriction>
    </xs:simpleType>
</xs:element>

```

- ① Das Element `Preis` darf als Inhalt nur eine Dezimalzahl aufweisen. Die Genauigkeit der Zahl ist durch das Attribut `fractionDigits` auf zwei Nachkommastellen festgelegt.
- ② Das Element `Anzahl` ist vom Typ `nonNegativeInteger` und darf positive Ganzzahlen, auch Null, enthalten.
- ③ Die aktuelle Lufttemperatur soll im Element `Grad` festgehalten werden. Die Genauigkeit der Angabe wird auf eine Nachkommastelle festgelegt. Größere Angaben als 60.0°C für die Maximaltemperatur und weniger als -90.0°C für die Minimaltemperatur sind nicht zulässig. Zum Vergleich: Die höchste Lufttemperatur, die jemals gemessen wurde, liegt bei 57.8°C (1922 in Libyen) und die niedrigste bei -89.2°C (1983 in der Antarktis).

Kalendarische Datentypen

Für die Angabe einer Zeit, eines Datums oder einer bestimmten Zeitspanne werden die kalendarischen Datentypen genutzt.

Datum

Datumsangaben werden über den Datentyp `date` realisiert, wobei die Schreibweise in der Form `JJJJ-MM-TT` fest vorgegeben ist. Das bedeutet, dass bei Datumsangaben zuerst die vierstellige Jahreszahl, dann die zweistellige Monatszahl und zum Schluss der Tag in der zweistelligen Form anzugeben ist. Diese Angaben sind jeweils durch einen Bindestrich voneinander zu trennen.

Beispiel

Mit der Elementdefinition von

```
<xs:element name="Bestelldatum" type="xs:date" />
```

ist die folgende Datumsangabe möglich:

```
<Bestelldatum>2014-09-27</Bestelldatum>
```

Eine zusätzliche Angabe der Zeitzone ist ebenfalls möglich. Dazu wird nach dem Datum der Zeitunterschied zur Greenwich-Zeit (GMT) angegeben. Eine mitteleuropäische Zeitangabe erfolgt somit wie im Beispiel.

```
<Bestelldatum>2014-09-27-01:00</Bestelldatum>
```

Uhrzeit

Zeitangaben sind über den Datentyp `time` festgelegt. Hier muss die Schreibweise im Format `hh:mm:ss` erfolgen, mit der zweistelligen Stundenangabe, gefolgt von den Minuten und den Sekunden. Alle Angaben sind Pflicht.

Beispiel

Mit der Festlegung von

```
<xs:element name="Uhrzeit" type="xs:time" />
```

ist die folgende Zeitangabe möglich:

```
<Uhrzeit>11:34:02-01:00</Uhrzeit>
```

Auch hier kann, wie bei der Datumsangabe, die Abweichung zur Greenwich-Zeit (GMT) angegeben werden.

Zeitraum

Über den Datentyp `duration` haben Sie die Möglichkeit, Zeitintervalle anzugeben. Das Intervall wird über die Form `PnYnMnDTnHnMnS` festgelegt.

- ✓ `P` steht für Periode und muss immer angegeben werden.
- ✓ `n` ist der Platzhalter für den jeweiligen Wert, den Sie angeben können.
- ✓ `Y, M, D` stehen für die Datumsangaben Jahr, Monat, Tag.
- ✓ `T` kennzeichnet den Beginn der Zeitangabe und muss immer angegeben werden, sobald Sie eine Zeit festlegen möchten.
- ✓ `H, M, S` stehen für die Zeitangaben Stunde, Minute, Sekunde.

Durch das Voransetzen eines Minus-Zeichens sind auch negative Zeitintervalle möglich. So bedeutet z. B. `-P30D` die Intervallangabe „vor 30 Tagen“.

Einschränkungen

Folgende Einschränkungen können bei den kalendarischen Datentypen angewendet werden:

- | | |
|----------------------------|---|
| ✓ <code>enumeration</code> | ✓ <code>maxInclusive, minInclusive</code> |
| ✓ <code>pattern</code> | ✓ <code>maxExclusive, minExclusive</code> |

Beispiel

Mit der Festlegung des Datentyps `duration`

```
<xs:element name="Periode" type="xs:duration" />
```

kann das Element `Periode` folgende Inhalte besitzen:

<code><Periode>P1Y2M3D</Periode></code>	<code><!-- in einem Jahr, zwei Monaten, drei Tagen --></code>
<code><Periode>PT10H20M30S</Periode></code>	<code><!-- in zehn Stunden, 20 Minuten, 30 Sekunden --></code>
<code><Periode>P3D10H</Periode></code>	<code><!-- in drei Tagen, zehn Stunden --></code>
<code><Periode>-PT30M</Periode></code>	<code><!-- vor 30 Minuten --></code>
<code><Periode>-P30M</Periode></code>	<code><!-- im Gegensatz dazu: vor 30 Monaten --></code>

Zusammenfassung

Name	Wertebereich
time	Uhrzeit, die täglich wiederkehrt (09:56:12.000)
date	Datumsformat JJJJ-MM-TT gemäß ISO 8601 (2014-09-27)
dateTime	Datum und Uhrzeit (2014-09-27T09:56:12.000+01:00)
gDay	Tag eines Monats über ----TT (----27)
gMonth	Monatsformat --MM-- (--09--)
gYear	Jahreszahl (2014)
gYearMonth	Monat eines bestimmten Jahres über JJJJ-MM (2014-09)
duration	Zeitraumangabe (-P1DT12H = vor einem Tag und 12 Stunden)

Weitere Datentypen

Andere Datentypen sind `base64Binary`, `hexBinary`, `float`, `double`, `language`, `QName` und `NOTATION`. Mit der Version 1.1 von XML Schema (vgl. Abschnitt 8.5) wurden weitere Datentypen eingeführt, so zum Beispiel `anyAtomicType` (als Basisyp aller einfachen und unteilbaren Datentypen) und `yearMonthDuration` sowie `dayTimeDuration` (Ableitungen von `duration`).

Zwei weitere interessante Datentypen, `boolean` und `anyURI`, sollen im Folgenden näher beschrieben werden.

boolean

Dieser Datentyp kann nur zwei Wahrheitswerte annehmen: wahr oder falsch. Mögliche Werte sind `true` oder `1` bzw. `false` oder `0`. Boolean wird sehr oft zur Datentypfestlegung bei Attributen genutzt.

```
<xs:attribute name="aktiv" type="xs:boolean" />
```

Das entsprechende Element kann folgendermaßen festgelegt sein:

```
<Sportler aktiv="true">Fabian Hambüchen</Sportler>
<Sportler aktiv="0">Stefanie Maria Graf</Sportler>
```

Mit dem Element `Sportler` werden die Namen von Sportlern gekennzeichnet. Das Attribut `aktiv` beschreibt, ob der entsprechende Sportler noch aktiv tätig ist.

anyURI

Über diesen Datentyp können Sie die richtige Verwendung von Internetadressen und lokalen Adressen sicherstellen.

```
<xs:attribute name="src" type="xs:anyURI" />
```

Mit dieser Festlegung können Sie Elementen das Attribut `src` zuweisen, um beispielweise Angaben zu den dazugehörigen Bildern zu hinterlegen.

```



```



Sie sollten beachten, dass eventuelle Leerzeichen in Internetadressen durch die Zeichenkette `%20` ersetzt werden müssen.

[illegible]

8 Komplexe Elemente in Schema

In diesem Kapitel erfahren Sie

- ✓ wie komplexe Elemente definiert werden
- ✓ wie Sie die unterschiedlichen Elementinhalte festlegen
- ✓ welchen Einfluss die Indikatoren auf komplexe Elemente haben

8.1 Was ist ein komplexes Element?

Die bisher behandelten einfachen Elemente dürfen keine weiteren Elemente und auch keine Attribute beinhalten. Wenn ein Element Unterelemente oder Attribute beinhalten soll, muss dieses als komplexes Element definiert werden.

Beispiele

Das komplexe Element `Adresse` enthält vier weitere Unterelemente.

```
<Adresse>
  <Name>Herbert Hagedorn</Name>
  <Strasse>Hasenweg 13</Strasse>
  <Ort>Frankfurt/Main</Ort>
  <PLZ>60000</PLZ>
</Adresse>
```

Das Element `Name` besitzt das Attribut `kundennr`, hat aber keinen Inhalt. Es ist somit ein komplexes, leeres Schema-Element.

```
<Name kundennr="123" />
```

Besitzt das Element einen Inhalt, spricht man von einem komplexen Element mit Inhalt.

```
<Name kundennr="123"> Herbert Hagedorn </Name>
```

Ein komplexes Element, das ein weiteres Element und zusätzlich einen Inhalt enthält, wird komplexes Element mit gemischtem Inhalt genannt.

```
<Kommentar>
  Gekauft am <Datum>26.05.2013</Datum>.
</Kommentar>
```

8.2 Definition eines komplexen Elements

Komplexe Elemente werden im XML-Schema grundsätzlich über das Element `complexType` kenntlich gemacht. Dabei gibt es vier verschiedene Arten von komplexen Elementen:

- ✓ leere Elemente;
- ✓ Elemente, die andere einfache oder komplexe Elemente beinhalten dürfen;
- ✓ Elemente, die nur einen Inhalt haben können;
- ✓ Elemente, die Elemente und Text enthalten dürfen.

Jedes dieser Elemente kann zusätzlich auch Attribute beinhalten.

Komplexe, leere Elemente

```
<xs:element name="Elementname">
  <xs:complexType>
    <xs:attribute name="Attributname" type="Datentyp" />
  </xs:complexType>
</xs:element>
```

Ein komplexes, leeres Element kann Attribute besitzen, hat jedoch keinen Inhalt. Die Schema-Definition des Beispiels

```
<Name kundenr="123" />
```

beschränkt sich daher auf die Festlegung des möglichen Attributs `kundenr`.

Beispiel: *kap08|empty.xsd*

```
① <xs:element name="Name">
②   <xs:complexType>
③     <xs:attribute name="kundenr" type="xs:positiveInteger" use="required" />
     </xs:complexType>
</xs:element>
```

- ① Das Element, dessen Inhalte eingeschränkt werden sollen, umschließt die gesamte Definition.
- ② Da das Element ein Attribut besitzen soll, muss es ein komplexes Element werden und ist demzufolge über das Element `complexType` entsprechend auszuweisen.
- ③ Das Element `Name` erhält über das Element `attribute` das Attribut `kundenr`, dessen Wert als positive Ganzzahl angegeben werden muss (`required`).

Sie können aber auch das Element `complexType` mit einem eigenen Namen kennzeichnen. Der komplexe Typ steht damit für weitere Elementdefinitionen zur Verfügung und kann am Element über das Attribut `type` referenziert werden.

```
① <xs:element name="Name" type="NameTyp" />
② <xs:complexType name="NameTyp">
   <xs:attribute name="kundenr" type="xs:positiveInteger" use="required" />
</xs:complexType>
```

- ① Über das Attribut `type` wird für das Element `Name` der komplexe Typ `NameTyp` festgelegt.
- ② Das Element `complexType` erhält die Bezeichnung `NameTyp`. Innerhalb dieses Elements wird das entsprechende Attribut definiert.

Komplexes Element mit Unterelementen

Bei diesem Element wird eine Reihenfolge von Unterelementen festgelegt, die in der XML-Instanz erwartet werden.

```
<xs:element name="Elementname">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Elementname" type="Datentyp" />
      <xs:element name="Elementname" type="Datentyp" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Im folgenden Element `Adresse` sollen die Elemente `Name`, `Strasse`, `Ort` und `PLZ` in der festgelegten Reihenfolge angegeben werden.

```
<Adresse>
  <Name> Herbert Hagedorn </Name>
  <Strasse>Hasenweg 13</Strasse>
  <Ort>Frankfurt/Main</Ort>
  <PLZ>60000</PLZ>
</Adresse>
```

Beispiel: *kap08!sequence.xsd*

Die Schema-Definition für das Element mit weiteren Unterelementen sieht folgendermaßen aus:

```
<xs:element name="Adresse">
  ① <xs:complexType>
  ②   <xs:sequence>
  ③     <xs:element name="Name" type="xs:string" />
      <xs:element name="Strasse" type="xs:string" />
      <xs:element name="Ort" type="xs:string" />
  ④     <xs:element name="PLZ" type="PLZTyp" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  ⑤ <xs:simpleType name="PLZTyp">
      <xs:restriction base="xs:integer">
        <xs:pattern value="\d{5}" />
      </xs:restriction>
    </xs:simpleType>
```

- ① Das Element `Adresse` wird über `complexType` als komplexes Element angelegt.
- ② Über das Element `sequence` legen Sie fest, dass die nachfolgenden Elemente in der angegebenen Reihenfolge erscheinen müssen.
- ③ Dies bedeutet, dass zuerst das Element `Name` erwartet wird und dann `Strasse`, `Ort` und `PLZ`.
- ④ In diesem Fall ist für das Element `PLZ` als Datentyp die eigene Definition `PLZTyp` einzuhalten.
- ⑤ Diese Definition legt fest, dass die Postleitzahl genau fünf Ziffern aufweisen muss.



Auch in diesem Beispiel kann das Element `Adresse` mit einem Typ verknüpft werden, den Sie über das Element `complexType` namentlich definieren und dessen Inhalt Sie angeben.

Komplexes Element mit Inhalt

Dieser Elementtyp kann nur einen einfachen Inhalt und Attribute besitzen. Zu diesem Zweck wird bei der Definition das Element `simpleContent` eingesetzt, mit dem Sie den möglichen Inhalt des komplexen Elements definieren.

```
<xs:element name="Elementname">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:Datentyp">
        <xs:attribute name="Attributname" type="xs:Datentyp" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Über das Element `extension` haben Sie die Möglichkeit, ein Element, ausgehend von einem vorhandenen Typ, zu erweitern. Mit `base` legen Sie den Datentyp des Elements fest. Der Attributtyp wird direkt im Element `attribute` angegeben. Statt ein Element zu erweitern, können Sie es mit dem Element `restriction` auch einschränken.

```
<Name kundenr="123"> Herbert Hagedorn </Name>
```

In diesem Beispiel stellt das Element ein einfaches Element dar, das eigentlich nur um das Attribut `kundenr` erweitert wurde. Da jedoch nur komplexe Elemente ein Attribut besitzen können, werden die einfachen Elemente als komplexe Elemente mit einem einfachen Inhalt (`simpleContent`) definiert. In diesem Fall wird das Element um das Attribut `kundenr` erweitert.

Beispiel: *kap08/simplecontent.xsd*

```

① <xs:element name="Name">
  <xs:complexType>
②    <xs:simpleContent>
③      <xs:extension base="xs:string">
④        <xs:attribute name="kundenr" type="xs:positiveInteger"
          use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
```

- ① Das Element `Name` wird über `complexType` als komplexes Element angelegt.
- ② Über das Element `simpleContent` legen Sie fest, dass das Element einen einfachen Inhalt besitzt.
- ③ Das Element soll jedoch erweitert werden (`extension`) und dabei grundsätzlich vom Datentyp `string` sein.
- ④ Erweitert wird das Element um das Attribut `kundenr`, das aus einer positiven Zahl bestehen muss.

Somit können komplexe Elemente mit Attributen auch einen Inhalt besitzen.

Komplexe Elemente mit gemischtem Inhalt

Unter einem gemischten Inhalt ist die gleichzeitige Verwendung von Inhalt und zusätzlichen Unterelementen zu verstehen.

```

<Kommentar>
  Bestellt: <Datum>2014-09-27</Datum>.
</Kommentar>
```

Im Beispiel besitzt das Element `Kommentar` einen Inhalt und gleichzeitig ein mit Inhalt gefülltes Element. Zu diesem Zweck können Sie das Definitionselement `complexType` um das Attribut `mixed` erweitern. Über `mixed` legen Sie fest, ob das Element zusätzlich einen Inhalt besitzen darf (`true`). Standardmäßig hat das Attribut `mixed` den Wert `false`, sodass in dem Fall die Angabe entfallen kann.

```
<xs:complexType mixed="true|false" />
```

Die Definition für das XML-Element `Kommentar` gestaltet sich dementsprechend.

Beispiel: *kap08/mixed.xsd*

```

① <xs:element name="Kommentar">
  <xs:complexType mixed="true">
②    <xs:sequence>
③      <xs:element name="Datum" type="xs:date">
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

- ① Das Element `Kommentar` wird über `complexType` als Element mit gemischtem Inhalt angelegt.
- ② Den Inhalt leiten Sie als neues Element innerhalb des komplexen Elements über das Schlüsselwort `sequence` ein.
- ③ Das zusätzliche Element muss vom Datentyp `date` sein.

8.3 Indikatoren

Mit Indikatoren können Sie festlegen, wie die Elemente in XML-Instanzen genutzt werden sollen. Einige der nachfolgenden Indikatoren kennen Sie bereits aus den vorangegangenen Beispielen.

- ✓ Reihenfolge-Indikatoren: `all`, `choice`, `sequence`
- ✓ Vorkommens-Indikatoren: `maxOccurs`, `minOccurs`
- ✓ Gruppen-Indikatoren: `group`, `attributeGroup`

Reihenfolge-Indikatoren

Hiermit bestimmen Sie, in welcher Reihenfolge die angegebenen Elemente in der XML-Instanz auftreten müssen.

`sequence`

Dieser Indikator ist bereits in den Beispielen zur Anwendung gekommen und soll daher nur noch einmal kurz genannt werden. Das Indikator-Element `sequence` umschließt die Menge der Unterelemente des komplexen Elements. Die Reihenfolge der Unterelemente ist unbedingt einzuhalten.

`all`

Der Indikator `all` spezifiziert, dass die Unterelemente eines komplexen Elements in einer beliebigen Reihenfolge auftreten können. Jedes Element kann genau einmal auftreten.

① ②	<pre> <xs:element name="Adresse"> <xs:complexType> <xs:all> <xs:element name="Vorname" type="xs:string"> <xs:element name="Nachname" type="xs:string"> <!-- ... --> </xs:all> </xs:complexType> </xs:element> </pre>
--------	--

- ① Das Element `Adresse` wird über `complexType` als komplexes Element angelegt.
- ② Der Reihenfolge-Indikator `all` legt fest, dass die nachfolgenden Elemente in der XML-Instanz maximal einmal auftreten können. Ob das Element `Vorname` vor dem Element `Nachname` stehen muss oder umgekehrt, ist hierbei nicht von Bedeutung.

`choice`

Ein weiterer Indikator ist `choice`, mit dem Sie eine Auswahl von Elementen festlegen können. In einer XML-Instanz darf jedoch nur eines der angegebenen Elemente auftreten. Mehrere Elemente gleichzeitig sind nicht möglich.

① ②	<pre> <xs:element name="Person"> <xs:complexType> <xs:choice> <xs:element name="Angestellter" type="xs:string"> <xs:element name="Arbeiter" type="xs:string"> </pre>
--------	--

```

    <!-- ... -->
  </xs:choice>
</xs:complexType>
</xs:element>

```

- ① Der Reihenfolge-Indikator `choice` legt fest, dass eines der nachfolgenden Elemente in der XML-Instanz auftreten muss.
- ② Entweder das Element `Angestellter` oder das Element `Arbeiter` – oder eines der weiteren, wenn vorhanden – muss als Unterelement von `Person` erscheinen.

Vorkommens-Indikatoren

Diese Indikatoren kommen zum Einsatz, wenn festgelegt werden soll, wie oft ein Element angegeben werden kann.

`maxOccurs` und `minOccurs`

Diese beiden Indikatoren werden als Attribute direkt am Element angegeben. Mit `minOccurs` legen Sie fest, wie oft ein Element innerhalb eines Elements mindestens vorhanden sein muss. Mit `maxOccurs` legen Sie die maximale Obergrenze fest.

Als Wert wird jeweils die Angabe einer positiven Zahl erwartet. Sollen keine Obergrenzen festgelegt werden, sodass theoretisch eine unbegrenzte Anzahl von Elementen auftreten kann, wird die Bezeichnung `unbounded` (unbegrenzt) verwendet.

Für die Reihenfolge-Indikatoren und die nachfolgenden Gruppen-Indikatoren sind die Werte von `maxOccurs` und `minOccurs` standardmäßig immer 1. Dies bedeutet, wenn Sie weder `minOccurs` noch `maxOccurs` angeben, wird immer genau ein Element erwartet.



Beispiel: *kap08|maxOccurs.xml*

Ein gutes Beispiel für den Einsatz von Vorkommens-Indikatoren ist die XML-Datei mit der Musiksammlung, bei der innerhalb des Elements `ALBUM` das Element `LIED` mehrfach auftritt.

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<MUSIKSAMMLUNG
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="maxOccurs.xsd">
  <ALBUM>
    <AUTOR>Max Mustermann</AUTOR>
    <GRUPPE>a-ha</GRUPPE>
    <TITEL>Minor earth major sky</TITEL>
    <LIED>Minor earth major sky</LIED>
    <LIED>Little black</LIED>
    <LIED>Velvet</LIED>
    <LIED>Summer moved on</LIED>
  </ALBUM>
</MUSIKSAMMLUNG>

```

Beispiel: *kap08|maxOccurs.xsd*

Die Definition der entsprechenden Schema-Datei sieht folgendermaßen aus:

```

① <?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.herdt.com">
②   <xs:element name="MUSIKSAMMLUNG">
     <xs:complexType>
       <xs:sequence>
③     <xs:element maxOccurs="unbounded" ref="ALBUM" />

```

```

        </xs:sequence>
      </xs:complexType>
    </xs:element>

④    <xs:element name="ALBUM">
      <xs:complexType>
⑤        <xs:sequence>
⑥          <xs:element name="AUTOR" />
          <xs:choice>
            <xs:element name="INTERPRET" />
            <xs:element name="GRUPPE" />
          </xs:choice>
          <xs:element name="TITEL" />
⑦          <xs:element name="LIED" minOccurs="1" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>

```

- ① Die Schema-Datei wird als XML-Dokument definiert.
- ② Das Hauptelement `MUSIKSAMMLUNG` ist ein komplexes Element.
- ③ Es wird auf die Definition des Elements `ALBUM` referenziert. Da in der XML-Instanz der Musiksammlung beliebig viele Alben aufgenommen werden können, wird der Indikator `maxOccurs` mit dem Wert `unbounded` angegeben.
- ④ Dies ist die eigentliche Definition des komplexen Elements `ALBUM`.
- ⑤ Es enthält eine Sequenz von Elementen, deren Reihenfolge unbedingt eingehalten werden muss.
- ⑥ Eines der Unterelemente besteht aus einer Auswahl (`choice`) von Elementen. Entweder muss in der XML-Instanz als Unterelement von `ALBUM` das Element `INTERPRET` oder `GRUPPE` angegeben sein. Da kein Vorkommens-Indikator festgelegt ist, muss eines der Elemente vorkommen.
- ⑦ Anders hingegen sieht es bei dem Element `LIED` aus. Da auf einem Album in der Regel mehrere Lieder vorhanden sind, wird die Obergrenze auf eine unbegrenzte Menge gesetzt. Die zusätzliche Angabe von `minOccurs=1`, damit mindestens ein Lied angegeben wird, kann also theoretisch weggelassen werden.

Gruppen-Indikatoren

Mit diesen Indikatoren haben Sie die Möglichkeit, Elemente und Attribute in bestimmten Gruppen zu definieren. Im weiteren Verlauf einer Schema-Datei können Sie diese Gruppen über eine Referenz ansprechen und somit mehrfach nutzen.

group

Elementgruppen werden über das Schlüsselwort `group` festgelegt.

```

<xs:group name="Gruppenname">
  ...
</xs:group>

<xs:group ref="Gruppenname" />

```

Innerhalb einer Elementgruppe muss ein Reihenfolge-Indikator `all`, `choice` oder `sequence` vorhanden sein.

Der Unterschied zu den komplexen Typen ist, dass Elementgruppen nicht aus komplexen Typen bestehen können.

Beispiel: *kap08\group.xsd*

In diesem Beispiel wird eine Elementgruppe `Adressengruppe` definiert, die eine festgelegte Reihenfolge von Elementen beinhaltet.


```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:group name="AdressenGruppe">
  <xs:sequence>
    <xs:element name="Strasse" type="xs:string" />
    <xs:element name="Ort" type="xs:string" />
    <xs:element name="PLZ" type="xs:integer" />
  </xs:sequence>
</xs:group>
...

```

Nach der Festlegung der Gruppe können Sie in einer anderen Gruppe oder einem komplexen Element darauf verweisen und somit die Definition mehrfach nutzen. Erweitern Sie das Beispiel, indem Sie ein Hauptelement *Bestellung* festlegen, das als Elemente den Namen des Kunden sowie die Liefer- und die Rechnungsanschrift beinhalten soll. Bei der Anschrift können Sie jeweils die Gruppe *AdressenGruppe* referenzieren.

```

...
<xs:element name="Bestellung" type="BestellungTyp" />

<xs:complexType name="BestellungTyp">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />

    <xs:element name="Lieferanschrift">
      <xs:complexType>
        <xs:group ref="AdressenGruppe" />
      </xs:complexType>
    </xs:element>

    <xs:element name="Rechnungsanschrift">
      <xs:complexType>
        <xs:group ref="AdressenGruppe" />
      </xs:complexType>
    </xs:element>

  </xs:sequence>
</xs:complexType>

```

attributeGroup

Auch Attribute können Sie als Gruppe definieren und diese mehrfach referenzieren. Die Definition erfolgt über das Element *attributeGroup*.

```

<xs:attributeGroup name="Gruppenname">
  ...
</xs:attributeGroup>

<xs:attributeGroup ref="Gruppenname" />

```

Das folgende Beispiel definiert eine Attributgruppe mit dem Namen *nameAttributGruppe*, die dann für die Attribute eines komplexen Elements genutzt werden kann.

```

<xs:attributeGroup name="nameAttributGruppe">
  <xs:attribute name="Vorname" type="xs:string" />
  <xs:attribute name="Nachname" type="xs:string" />
</xs:attributeGroup>

```

8.4 Schema erweitern

Mit der Erweiterung des Schemas ist gemeint, dass Sie bei der Festlegung der Elemente einen Platzhalter einfügen können, der für ein beliebiges weiteres Element steht. Dies bedeutet: Sie können ein XML-Dokument mit Elementen erweitern, ohne dass diese in dieser speziellen Schema-Datei angegeben sind. Die Lösung hierfür bietet das Element `any`.

any (Wildcard)

Statt der eigentlichen Definition des Elements über `xs:element` geben Sie `xs:any` an.

Beispiel: *kap08\personalie.xsd*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.herdtd.com">

  <xs:element name="Personalie">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" />
        <xs:any minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

In dem Beispiel wird im Element `Personalie` das Unterelement `Name` festgelegt. `xs:any` steht für ein beliebiges weiteres Element. Mit `minOccurs` stellen Sie sicher, dass das beliebige Element nicht unbedingt angegeben werden muss.

Die Erweiterung der Schema-Datei erfolgt über die Definition einer weiteren Schema-Datei.

Beispiel: *kap08\adresse.xsd*

In der späteren XML-Instanz sollen nach dem Element `Name` innerhalb des Hauptelements `Personalie` die Adressangaben folgen. Diese werden über das Element `Adresse` in einer separaten Schema-Datei definiert.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.herdtd.com" elementFormDefault="qualified">

  <xs:element name="Adresse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Strasse" type="xs:string" />
        <xs:element name="Ort" type="xs:string" />
        <xs:element name="PLZ" type="xs:integer" minOccurs="5" maxOccurs="5" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Nach der Festlegung des Präfixes `xs` als XML-Schema-Namensraum wird über `targetNamespace` der Zielnamensraum festgelegt, in dem die nachfolgende Definition gültig werden soll. Elemente ohne Präfix werden dem Namensraum `http://www.herdtd.com` zugeordnet. Das Attribut `elementFormDefault` mit dem Wert `qualified` besagt, dass alle Elemente mit einem Präfix geschrieben werden müssen. Dem Element `Adresse` wird eine festgelegte Reihenfolge von drei Unterelementen zugewiesen.

Beispiel: *kap08lany.xml*

Das Beispiel bindet beide Schema-Dateien ein und erweitert somit die Struktur der Daten, sodass innerhalb des Hauptelements *Personalie* das Unterelement *Adresse* verwendet werden kann.

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<Personalie
①   xmlns="http://www.herdt.com"
②   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
③   xsi:schemaLocation="http://www.herdt.com personalie.xsd
                        http://www.herdt.com adresse.xsd">
④   <Name>Heiko Schröder</Name>
⑤   <Adresse>
      <Strasse>Musterstraße 123</Strasse>
      <Ort>Musterstadt</Ort>
      <PLZ>88888</PLZ>
    </Adresse>
</Personalie>

```

- ① Der standardmäßig zu verwendende Namensraum ist *http://www.herdt.com*.
- ② Der Namensraum *xsi* wird für die XML-Instanz angegeben.
- ③ Im Namensraum *xsi* kommen die Definitionen der beiden Schema-Dateien *personalie.xsd* und *adresse.xsd* zur Anwendung.
- ④ Als Unterelement von *Personalie* wurde das Element *Name* in der Schema-Datei *personalie.xsd* definiert.
- ⑤ Durch die Festlegung von *xs:any* können beliebige Elemente folgen. In diesem Fall ist es das komplexe Element *Adresse*. Durch die Einbindung der Schema-Datei *adresse.xsd*, in der dieses Element definiert wurde, ist die Nutzung des Elements *Adresse* und der Unterelemente *Strasse*, *Ort* und *PLZ* erlaubt.

anyAttribute

Auf die gleiche Art und Weise können Sie einen Platzhalter für Attribute in der Schema-Datei unterbringen. Dazu definieren Sie den Platzhalter *anyAttribute* und eine zweite Schema-Datei mit der Definition der Attribute.

```

<xs:element name="Elementname">
  <xs:complexType>
    <!-- ... -->
    <xs:anyAttribute />
  </xs:complexType>
</xs:element>

```

In der XML-Instanz müssen Sie nur noch beide Schema-Dateien laden und können dann entsprechend das erweiterte Attribut nutzen.

8.5 Schema 1.1

Neuerungen

Die im Jahr 2012 veröffentlichte Version XML Schema 1.1 bietet neben der Verbesserung einzelner bereits vorhandener Funktionen eine Reihe von Neuerungen, unter anderem:

- ✓ **Zusicherungen** bzw. Behauptungen (Assertions) durch das Element *<assert>* und die Fasette *<assertion>*
- ✓ Schemaweite Attribute
- ✓ Erhöhte Flexibilität bei der Weiterentwicklung eines Schemas mit den Elementen *<openContent>* bzw. *<defaultOpenContent>*
- ✓ Bedingte Typisierung in Form alternativer Datentypen in Abhängigkeit von Attributwerten

Zur Kennzeichnung der verwendeten Version gibt es den Version Control Namespace

`xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"`. Innerhalb eines Schema können unterschiedliche Parser-Implementierungen berücksichtigt werden. Durch die Auswertung der Attribute `vc:minVersion` bzw. `vc:maxVersion` ermittelt ein Parser, die für die Validierung des Dokuments zu verwendende Schema-Version.

Gegenwärtig wird XML Schema 1.1 noch nicht von allen Programmen und Werkzeugen unterstützt. Vor seiner Anwendung ist daher gerade bei unternehmensübergreifenden Projekten eine Abklärung der Möglichkeit des Einsatzes notwendig.

Zusicherungen

Zusicherungen ermöglichen die Validierung von XML-Dokumenten mit ihren konkreten Werten. Dies erfolgt in Abhängigkeit von der Erfüllung von im Schema-Dokument festgelegten Regeln bzw. Bedingungen für die Elemente und Attribute. Formuliert werden die Regeln bzw. Bedingungen über XPath Ausdrücke (vgl. Kapitel 10).

Die Umsetzung für komplexe Datentypen erfolgt mit dem Element `<assert>`, für simple Datentypen mit der Fasette `<assertion>`.

Beispiel: *kap08|budget.xsd*

Das Beispiel zeigt die Anwendung des Elements `<assert>`. Über einen Vergleich wird sichergestellt, dass der Wert des Attributes `minimal` höchstens genauso groß wie der des Attributes `maximal` ist.

①	<pre> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:herdt="http://www.herdt.com" vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"> <xs:element name="Budget"> <xs:complexType> <xs:sequence> <xs:element name="Bereich" type="xs:string"/> <xs:element name="Wert" type="werttyp"/> </xs:sequence> </xs:complexType> </xs:element> <xs:complexType name="werttyp"> <xs:simpleContent> <xs:extension base="xs:integer"> <xs:attribute name="minimal" type="xs:integer"/> <xs:attribute name="maximal" type="xs:integer"/> ② <xs:assert test="@minimal le @maximal"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:schema> </pre>
---	--

- ① Das Attribut `vc:minVersion` gibt dem Parser vor, mit welcher Schema-Version das Dokument validiert werden soll.
- ② Über das `<xs:assert>` Element wird bestimmt, dass der `minimal` angegebene Wert kleiner oder höchstens genauso groß (`le` = lower-equal) wie der bei `maximal` angegebene sein kann.

Beispiel: *kap08|budget.xml*

Das Resultat der Validierung des zugehörigen XML-Dokuments hängt von den verwendeten Attributwerten ab.

```

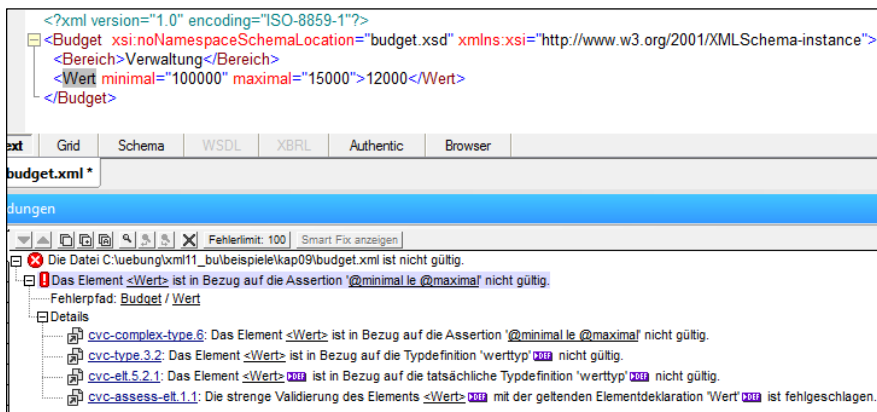
<?xml version="1.0" encoding="ISO-8859-1"?>
<Budget xsi:noNamespaceSchemaLocation="budget.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Bereich>Verwaltung</Bereich>
  <Wert minimal="10000" maximal="15000">12000</Wert>
</Budget>

```

- ① Für den Wert werden als Minimalwert 10000 und als Maximalwert 15000 festgelegt. Der eigentliche Wert liegt in diesem Bereich. Die Validierung des Dokuments ergibt keinen Fehler.



Erfolgreiche Validierung des Dokuments im XML-Editor XMLSpy>



Bei Eingabe eines größeren Minimal- als Maximalwertes meldet die Validierung einen Fehler

Schemaweite Attribute

Durch die Zuordnung einer Attributgruppe über das `<schema>` Attribut `defaultAttributes` gelten die in der Gruppe enthaltenen Attribute für alle Elemente des Schemas, in dem die Attributdeklaration erfolgt.

```

① <xs:schema ... defaultAttributes="StandardAttribute">
②   <xs:attribute-group name=" StandardAttribute ">
     <!-- Deklaration der Attribute -->
   </xs:attribute-group>
   <!-- alle weiteren Deklarationen -->
</xs:schema>

```

- ① Die Zuordnung der Attributgruppe zum `<schema>` Attribut `defaultAttributes`.
 ② Die Definition der zugeordneten Attributgruppe mit den einzelnen Attributen.

Für einzelne Elemente des Schemas kann die Gültigkeit der Standardattribute durch das Setzen des Wertes `false` für das Attribut `defaultAttributesApply` ausgeschlossen werden.

Flexibilität mittels offener Modelle

Über die Elemente `<xs:openContent>` und `<xs:defaultOpenContent>` können Modelle in XML Schema 1.1 so gestaltet werden, dass komplexe Typen nicht komplett fixiert sind, sondern ergänzende, nicht festgelegte Elemente erlauben. Damit kann eine bestehende Schema-Datei flexibel und offen für sich verändernde XML-Dateien nutzen, ehe zu einem späteren Zeitpunkt ggf. eine Anpassung des XML Schemas erfolgt.

In dem bereits verwendeten Beispiel *kap07/bestell.xsd* kann das Element *Bestellung* beispielsweise so erweitert werden, dass es weitere Unterelemente erlaubt.

<p>①</p> <p>②</p>	<pre> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="Bestellung"> <xs:complexType> <xs:openContent mode="interleave"> <xs:any processContents="lax" /> </xs:openContent> <xs:sequence> <xs:element name="Adresse" type="BestellAdresse" /> <xs:element name="Ware" type="WarenTyp" /> </xs:sequence> <!-- ... --> </xs:complexType> </xs:element> ... </pre>
-------------------	---

- ① Durch das Einfügen von `openContent` wird der komplexe Typ als offen für weitere Elemente markiert. Das Zuweisen von `interleave` zum Attribut `mode` sorgt dafür, dass diese Elemente zwischen den bereits vorhandenen Elementen angegeben werden können.
- ② Das Attribut `processContents` bestimmt für einen Knoten vom Typ `<xs:any>` die Überprüfungsebene. Der Wert `lax` legt fest, dass die Überprüfung nur durchgeführt wird, wenn zugehörige Elemente gefunden werden.

Ein komplett offenes Schema lässt sich durch den Einsatz von `<xs:defaultOpenContent>` als globale Komponente erreichen.

<p>①</p>	<pre> <xs:defaultOpenContent mode="interleave" appliesToEmpty="false"> <xs:any /> </xs:defaultOpenContent> </pre>
----------	---

- ① Das gesamte Schema wird als offen für beliebige Elemente gekennzeichnet. Der Wert des Attributes `appliesToEmpty` bestimmt dabei, ob zusätzliche Elemente auch in leeren Elementen zulässig sind.

Bedingte Datentypen

Bedingte Datentypen (Conditional Type Alternatives, CTA) ermöglichen es, für ein Element unterschiedliche Datentypen auszuwählen. Genau wie bei Zusicherungen werden XPath-Bedingungen ausgewertet, die einen Attributwert des Elements verwenden. Ergibt der Test `true`, wird der zugeordnete Datentyp verwendet.

Für die Realisierung der bedingten Datentypen wurden in XML Schema 1.1 das Element `<xs:alternative>` auf- sowie einige Änderungen am Element `<xs:element>` vorgenommen. Die prinzipielle Syntax lautet:

```
<xs:alternative test="XPath-Ausdruck" type="type" />
```

Voraussetzung für die Zuordnung eines alternativen Typs ist, dass dieser von dem ursprünglich für das Element deklarierten Typ abgeleitet sein muss. Möglich ist ebenfalls die Zuordnung des gleichfalls neuen Typs `xs:error`. Wird dieser zugeordnet, wird das betreffende Element immer als ungültig validiert.

Im Beispiel (Quelle: IBM) ist das Element `title` mit dem Basistyp `xs:anyType` deklariert. Daneben werden fünf alternative Typen deklariert. Bei der Überprüfung wird nach der Reihenfolge vorgegangen. Der Test, welcher zuerst den Wert `true` liefert, bestimmt den zu verwendenden Typ. Liefert keiner der ersten drei Tests den Wert `true`, wird geprüft, ob das zu testende Attribut vorhanden ist. Ist dies nicht der Fall, wird der Typ `xs:error` zugewiesen. Das Element ist damit als invalid gekennzeichnet. Ergibt kein Test `true`, wird der Default-Typ verwendet.

```

① <xs:element name="title" type="xs:anyType">
②   <xs:alternative test="@type='text'" type="xs:string"/>
③   <xs:alternative test="@type='html'" type="htmlContentType"/>
④   <xs:alternative test="@type='xhtml'" type="xhtmlContentType"/>
⑤   <xs:alternative test="@type" type="xs:error"/>
⑥   <xs:alternative type="xs:string"/>
</xs:element>

```

- ① Deklaration des Elements mit dem Basistyp `xs:anyType`.
- ② Deklaration des alternativen Typs `xs:string` für den Wert 'text' des Attributs `type`.
- ③ Deklaration des alternativen Typs `htmlContentType` für den Wert 'html' des Attributs `type`.
- ④ Deklaration des alternativen Typs `xhtmlContentType` für den Wert 'xhtml' des Attributs `type`.
- ⑤ Zuweisung des Typs `xs:error`, wenn das Attribut nicht vorhanden ist.
- ⑥ Deklaration des Default-Typs.

Die unterschiedlichen Varianten des Elementes `title` zeigen die Auswahl der verschiedenen Typen:

```

① <title type="text">Neuigkeiten</title>
② <title type="xhtml" xmlns:xhtml="http://www.w3.org/1999/xhtml">Neue
   <xhtml:em> Neuigkeiten </xhtml:em>!</title>
③ <title>Neuigkeiten</title>
④ <title type="unbekannt">Fehler!</title>

```

- ① Die erste Alternative wird gewählt (`xs:string`).
- ② Die dritte Alternative wird gewählt (`xhtmlContentType`).
- ③ Der Default-Typ wird gewählt (`xs:string`).
- ④ Die vierte Alternative wird gewählt (`xs:error`). Das Element ist damit invalid.

8.6 Übungen

Übung 1: DTD in Schema-Definition umwandeln

Übungsdatei: *kap08\kfz.dtd*

Ergebnisdateien: *kap08\kfz.xml*, *kap08\kfz.xsd*,
kap09\kfz2.xml, *kap08\kfz2.xsd*,

- ① Konvertieren Sie die DTD der Fahrzeugverwaltung in eine allgemeine XML-Schema-Syntax. Achten Sie darauf, dass Dezimalzahlen mit einem Punkt `.` statt mit einem Komma `,` notiert werden.
- ② Schränken Sie die möglichen Daten ein, sodass nur positive Zahlenwerte möglich sind. In einem weiteren Schritt definieren Sie sinnvolle Maximalangaben, z. B. nicht mehr als 10.000 cm³ Hubraum usw.

Übung 2: Schema-Datei definieren

Übungsdatei: --

Ergebnisdateien: *kap08\buchhaendler.xml*,
kap08\buch.xsd

Ein Buchhändler möchte mit dem Lieferanten die Buchdaten per XML austauschen. Die Daten haben folgendes Format:

Das Hauptelement `Lager` kann mindestens ein, dafür aber unendlich viele Elemente `Buch` haben. `Buch` selbst besteht aus den Elementen `Autor`, `Titel`, `Verlag`, `Seiten`, `Erscheinungsdatum`, `Bestellnummer` und `PreisEUR`. Die `Bestellnummer` besteht aus zwei beliebigen Buchstaben, gefolgt von fünf Ziffern, wie z. B. AB12345. Der Preis soll mit zwei Nachkommastellen angegeben werden.

Erstellen Sie das dafür notwendige XML-Schema.

9 Formatierungssprachen

In diesem Kapitel erfahren Sie

- ✓ Grundlegendes über die Möglichkeiten von XSL
- ✓ wie Sie mit Stylesheets XML-Daten formatieren

Voraussetzungen

- ✓ Kenntnisse in Cascading Style Sheets (CSS)

9.1 Übersicht der Sprachen

```
<INTERPRET>Alan Parsons Project</INTERPRET>
<TITEL>Ladyhawke</TITEL>
```

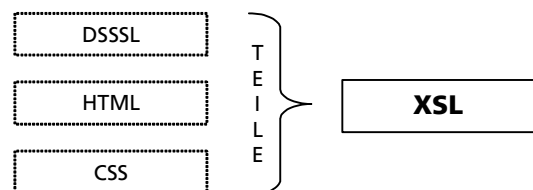
Der verkürzte Auszug aus einem XML-Beispiel legt durch die Bezeichnung der Elemente fest, dass es sich bei dem eingeschlossenen Wert um eine Information bezüglich des Interpreten sowie eines Titels handelt.

Die Angabe der Elemente gibt jedoch keine Auskunft darüber, wie die Information dargestellt werden soll. Um eine Formatierung der Daten zu ermöglichen, wird die Sprache **XSL** (**Extensible Stylesheet Language** = Erweiterbare Formatierungssprache) verwendet.

Die Formatierung von Daten in der Meta-Auszeichnungssprache **SGML** erfolgt über die recht komplizierte und sehr umfangreiche Formatierungssprache **DSSSL** (**D**ocument **S**tyle **S**emantics and **S**pecification **L**anguage). Diese Sprache ist eine reine Programmiersprache auf der Basis eines LISP-Dialekts.

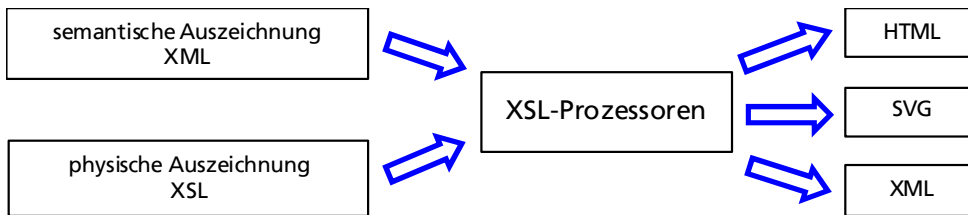
In HTML legen die Elemente fest, wie die jeweiligen Daten im Browser darzustellen sind. **CSS** (**C**ascading **S**tyle **S**heets) erlaubt eine Formatierung der Informationen mithilfe von Vorlagen für die Browser, die CSS als Formatierungssprache unterstützen.

Der Funktionsumfang von CSS hat sich bei der Formatierung von XML-Daten als eingeschränkt erwiesen. Es konnten nur einfache XML-Dokumente dargestellt werden. Im August 1997 wurde daher von Vertretern einiger Software-Firmen die Formatierungssprache XSL entwickelt und als Vorschlag beim W3-Konsortium eingereicht. Der Vorteil dieses Konzepts ist die Integration von DSSSL-Eigenschaften sowie die Einbeziehung von HTML- und CSS-Objekten in die XML-Syntax.



9.2 Grundlagen von XSL

Mit der Formatierungssprache XSL können vorhandene XML-Daten über entsprechende Prozessoren in andere Dokumentformate umgewandelt werden. Dabei legen Sie über Transformationsanweisungen die Formatierung und das Layout für die Darstellung fest. Ein Beispiel hierfür ist das Format **SVG** (**S**calable **V**ector **G**raphics), dessen Dokumentinhalt ebenfalls auf einer XML-Struktur aufbaut. Ebenso können Sie mit XSL aus einem XML-Dokument ein formatiertes XML-Dokument erstellen.



Eine semantische Auszeichnung umschreibt mithilfe des Elementnamens die Bedeutung des Inhalts.

```
<INTERPRET>Alan Parsons Project</INTERPRET>
<TITEL>Ladyhawke</TITEL>
```

Auch in HTML existiert eine Reihe semantischer Elemente.

```
<H1>Alan Parsons Project</H1>
<H2>Ladyhawke</H2>
```

Da die HTML-Tags fest definiert sind, konnten die für deren Darstellung notwendigen Layoutinformationen fest in die Browser integriert werden. Damit weiß z. B. jeder Browser, dass der Inhalt eines `h1`-Elements fetter und größer als der eines `p`-Elements darzustellen ist. Über die browserseitigen Voreinstellungen hinausgehend kann der Autor eines HTML-Dokuments die Darstellung der einzelnen Elemente zusätzlich über ein Cascading Style Sheet (CSS) umformatieren.

In XML werden Elemente verwendet, die dem Browser nicht bekannt sind. In diesem Fall müssen Sie dem Browser mitteilen, wie er das entsprechende Element darstellen soll. Diese physische Auszeichnung wird in XML über die Formatierungssprache XSL erreicht. Einmal erstellte Daten im XML-Format können über verschiedene XSL-Transformationen in andere Dokumentformate konvertiert werden.

Mit XSL können Sie:

- ✓ XML-Daten in HTML oder ein anderes Textformat transformieren,
- ✓ XML-Daten filtern und sortieren,
- ✓ XML-Daten abhängig vom Wert formatieren.

Die Extensible Stylesheet Language enthält verschiedene Komponenten.

XSL	Oberbegriff für die XSL-Transformation (XSLT), für XPath und XSL-FO
XSLT - XSL-Transformation	Sprache zur Beschreibung, wie ein XML-Dokument in ein anderes Dokument transformiert werden soll
XPath - XML Path Language	Ursprünglich eine Untersprache von XSLT zum Kennzeichnen von speziellen Teilen eines XML-Dokuments, wird jetzt separat weiterentwickelt
XSL-FO	Extensible Stylesheet Language - Formatting Objects beschreibt, wie Texte und grafische Elemente für den Druck und für die Darstellung am Bildschirm angeordnet werden sollen.

Warum gibt es noch eine Formatierungssprache?

Obwohl CSS 2.1 als Standard verabschiedet wurde und für CSS3 bereits einige Module fertig sind, entwickelt das W3-Konsortium mit XSL parallel eine weitere Formatierungssprache. Auf die Frage nach dem Sinn dieser Entwicklung gibt das W3-Konsortium folgende Antwort: Das herausragendste Merkmal ist, dass CSS in HTML-Dokumenten verwendet werden kann. Auf der anderen Seite ist es mit XSL möglich, Dokumente mit XML-Daten in HTML- und CSS-Dokumente zu transformieren. Deshalb ergänzen sich die beiden Formatierungssprachen und können gemeinsam in Browsern benutzt werden. (Quelle: <http://www.w3.org/Style>)

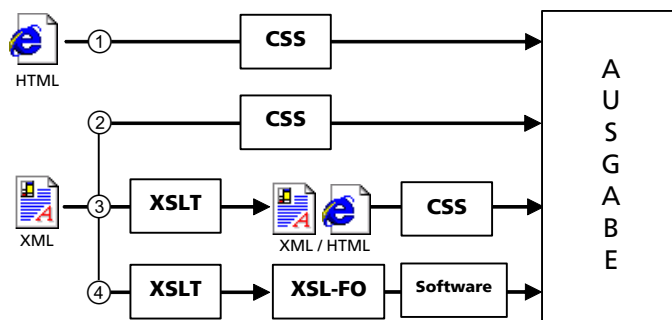
Verwendung in ...	CSS	XSL
HTML	ja	nein
XML	ja	ja
XSLT	nein	ja

Welche Formatierungssprache sollten Sie verwenden?

Benutzen Sie CSS, wenn Sie können, und XSL, wenn Sie müssen (siehe <http://www.w3.org/Style/CSS-vs-XSL>). Der Grund hierfür ist die einfache Funktionsweise von CSS. Außerdem existieren mehr Programme zur Erstellung von CSS-Formaten als für XSL-Formate.

Setzen Sie XSL ein, wenn Sie Daten sortieren, filtern oder ersetzen müssen. Beispielsweise erstellen Sie ein XML-Dokument und möchten daraus nur bestimmte Daten anzeigen lassen. Oder Sie möchten eine sortierte Liste der XML-Daten erstellen. Nutzen Sie XSL, wenn Sie die Inhalte mehrfach verwenden und vielleicht auch in einer anderen Reihenfolge anzeigen wollen bzw. in andere Formate transformieren möchten.

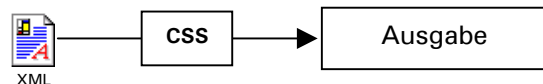
Folgende Übersicht soll Ihnen die Auswahl der entsprechenden Formatierungssprache erleichtern.



- ① Benutzen Sie zur Formatierung eines HTML-Dokuments CSS.
- ② Möchten Sie alle Daten eines XML-Dokuments in der angegebenen Reihenfolge anzeigen, benutzen Sie ebenfalls die CSS.
- ③ Erstellen Sie über XSLT ein XML- oder HTML-Dokument, das Sie über die CSS formatieren können.
- ④ Ansonsten greifen Sie auf die XSL-Transformation zurück und erstellen XSL-FO-Dokumente für die neu geordneten Daten. Diese werden dann über eine spezielle Software z. B. in ein PDF-Dokument transformiert.

9.3 Einbinden von CSS

Möchten Sie alle XML-Daten eines Dokuments in unveränderter Reihenfolge ausgeben, formatieren Sie die einzelnen Elemente über CSS.



Beispiel: *kap09\musik.css*

Sie können Ihre bisher erstellte Musiksammlung mithilfe von CSS so formatieren, dass der Browser die Daten wie ein HTML-Dokument darstellt. Erstellen Sie dazu die folgende Stylesheet-Datei zur Formatierung Ihrer XML-Daten der Musiksammlung.

```

ALBUM      { font-family:Verdana,helvetica,sans-serif; color:#000; }
INTERPRET, GRUPPE { font-size:14pt; font-weight:bold; color:#800000; }
TITEL      { font-size:12pt; font-weight:bold; color:#800000; }
LIED, AUTOR { font-size:8pt; text-indent:10pt; }
  
```

Innerhalb der CSS-Datei werden nicht wie bisher die jeweiligen HTML-Tags oder Klassennamen angegeben, sondern die jeweiligen Namen der Elemente. Ähnlich wie in HTML werden auch die Stylesheets in XML in das aktuelle Dokument eingebunden. Hierfür verwenden Sie die entsprechende Processing Instruction (PI).

```
<?xml-stylesheet type="text/css" href="CSS-Datei"?>
```

Es ist darauf zu achten, dass das Einbinden der CSS-Datei in die XML-Datei vor dem Hauptelement erfolgen muss. Die Typangabe `type="text/css"` teilt dem Browser mit, dass es sich hierbei um eine Stylesheet-Datei handelt.

Beispiel: *kap09\musik-css.xml*

In die bereits bestehende Musiksammlung soll die CSS-Datei eingebunden werden, um die Daten für die Ausgabe im Browser zu formatieren.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="musik.css"?>
<MUSIKSAMMLUNG>
  <ALBUM>
    <INTERPRET>Moby</INTERPRET>
    <TITEL>Play</TITEL>
    <LIED>Honey</LIED>
    <LIED>Find my baby</LIED>
    <LIED>Porcelain</LIED>
    <!-- usw. -->
  </ALBUM>
</MUSIKSAMMLUNG>
```

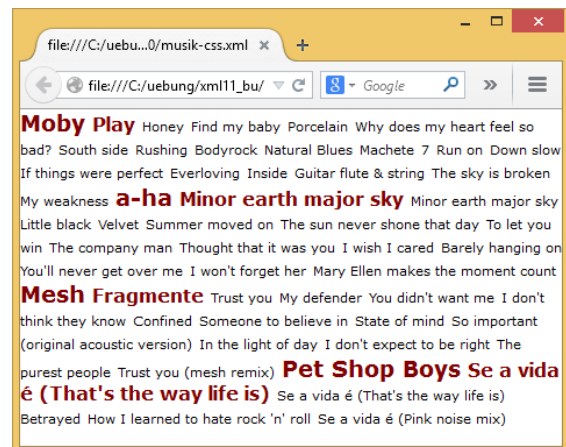
- Öffnen Sie diese Datei in Ihrem Browser.

Der Browser formatiert die Daten so, wie Sie dies in den Stylesheets für das entsprechende Element angegeben haben. Jedoch erscheinen alle Daten als Fließtext und werden hintereinander gesetzt.

Dies ist kein Fehler des Browsers, denn bisher haben Sie dem Browser noch nicht mitgeteilt, dass Sie diese Elemente als Absätze formatieren möchten.

In HTML erzwingen Sie einen neuen Absatz über das Element `p` oder das Element `div`. Einen Zeilenumbruch fügen Sie mit dem Element `br` ein.

Damit in XML die Elemente als Absätze formatiert werden können, steht Ihnen die CSS-Eigenschaft `display` zur Verfügung.



```
display:block|inline|list-item
```

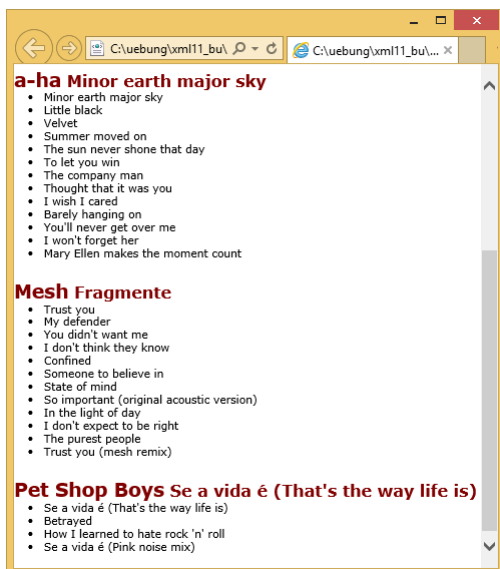
block	Erzwingt einen Block und somit einen neuen Absatz
inline	Erzwingt die Anzeige im Textfluss und somit keinen neuen Absatz
list-item	Wie block, jedoch mit einem Aufzählungszeichen

Beispiel: *kap09\musik2.css*

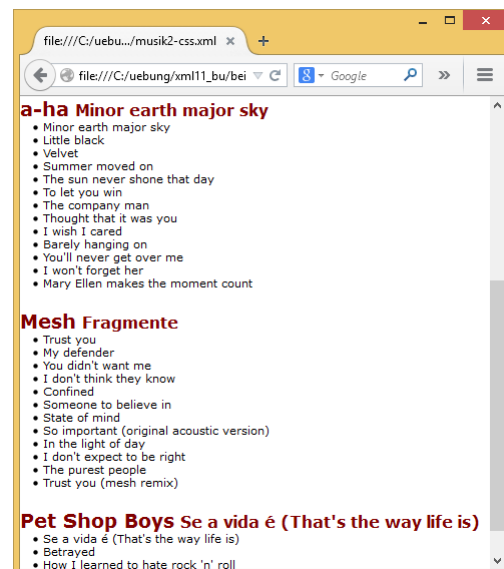
In der Auflistung der XML-Daten sollen der Name des Interpreten und der Titel des Albums in einer Zeile stehen. Jedes Lied soll als Aufzählung formatiert werden. Fügen Sie über die Stylesheet-Angabe `display` die Zeilenumbrüche ein. Damit die Alben voneinander getrennt dargestellt werden, setzen Sie einen unteren Abstand von 20 Pixel (`margin-bottom:20px;`). Die Box zur Darstellung der List-Marker wird unsichtbar, wenn sie sich außerhalb der Block-Box befindet. Über die Eigenschaft `list-style-position` kann sie entsprechend positioniert werden.

```
ALBUM { display:block; margin-bottom:20px;
        font-family:Verdana,helvetica,sans-serif; color:#000; }
INTERPRET, GRUPPE { font-size:14pt; font-weight:bold; color:#800000; }
TITEL { font-size:12pt; font-weight:bold; color:#800000; }
LIED, AUTOR { display:list-item; list-style-position: inside;
               font-size:8pt; text-indent:10pt; }
```

► Öffnen Sie die XML-Datei erneut in Ihrem Browser.



Internet Explorer 11.0



Mozilla Firefox 33.0



Bei Verwendung von älteren Browsern kann es hier zu unterschiedlichen Darstellungen kommen. Dies ist ein grundlegendes Problem, mit dem Web-Autoren konfrontiert werden. Nicht jeder Browser stellt immer alle Stylesheet-Formatierungen so dar, wie es vom W3-Konsortium in der CSS-Spezifikation vorgegeben wird. Aktuell betrifft dies vor allem die neuen Module der CSS3-Spezifikation, die noch nicht alle in jedem Browser vollumfänglich unterstützt werden. Sollen XML-Daten unabhängig vom Browser identisch dargestellt werden, sind Sie gezwungen, immer den kleinsten gemeinsamen Nenner in Bezug auf die Darstellung der Formatierung zu finden. Dies erreichen Sie, wenn Sie verschiedene Attribute der CSS-Eigenschaften ausprobieren und in unterschiedlichen Browsern testen.

9.4 Übung

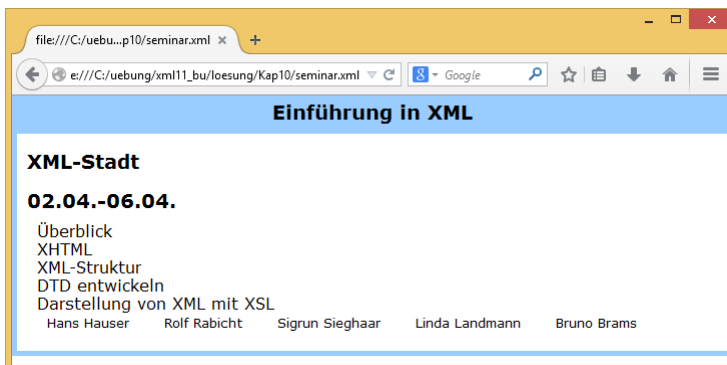
XML-Elemente mit CSS formatiert darstellen

Übungsdatei: --

Ergebnisdateien: *kap09\seminar.xml*,
kap09\seminar.css

Erstellen Sie mithilfe der CSS für die XML-Daten des Seminars aus den vorherigen Übungen die nachfolgende Formatierung.

- ✓ Alle XML-Elemente sollen in der Schriftart Verdana angezeigt werden.
- ✓ Das komplette XML-Dokument soll einen blauen Rahmen erhalten.
- ✓ Das Element `titel` soll zentriert auf blauem Hintergrund angezeigt werden.
- ✓ Die Elemente `ort` und `datum` sollen fett hervorgehoben werden.
- ✓ Variieren Sie mit oberen (`margin-top`) und unteren (`margin-bottom`) Abständen.
- ✓ Die Vor- und Nachnamen der Anwesenden sollen komplett auf einer Zeile angezeigt werden. Realisieren Sie dies mit der CSS-Eigenschaft `margin-left`.
- ✓ Binden Sie die externe CSS-Datei in die XML-Datei ein.



Das Ergebnis im Firefox

10 XPath

In diesem Kapitel erfahren Sie

- ✓ wie Sie mithilfe von XPath bestimmte Elemente selektieren können
- ✓ welche Rolle die Achsen, Knotenprüfungen und Prädikate spielen

Voraussetzungen

- ✓ Kenntnisse des Aufbaus von XML-Dokumenten

10.1 XPath-Grundlagen

Entwicklung

XPath ist keine XML-Anwendung, sondern eine Beschreibungssprache für Ausdrücke mit einer eigenen Syntax. Sie ermöglicht, einzelne Teile eines XML-Dokuments zu selektieren, anzusprechen und auszuwerten. Die offizielle Empfehlung des W3C für XPath 1.0 finden Sie unter <http://www.w3.org/TR/xpath/>. Verwendung findet XPath sowohl in XML Schema (vgl. Kapitel 7 und 8) als auch in XSL und XSLT Dokumenten (vgl. Kapitel 11 und 12).

Seit dem Jahr 2010 gibt es die Empfehlung des W3C für XPath 2.0 (<http://www.w3.org/TR/xpath20/>). Diese stellt eine Erweiterung der Version 1.0 dar, wird jedoch noch nicht von allen XML-Prozessoren unterstützt. Einen Überblick über einige Neuerungen enthält der entsprechende Abschnitt am Ende des Kapitels. XPath 2.0 ist die Grundlage von XSLT 2.0 und XQuery 1.0 (vgl. Kapitel 14) und wurde zeitgleich mit diesen Empfehlungen veröffentlicht.

Seit dem April des Jahres 2014 liegt mit der Version XPath 3.0 (<http://www.w3.org/TR/xpath-30/>) eine weitere Empfehlung vor, welche neben Fehlerkorrekturen einige weitere Neuerungen, wie beispielsweise dynamische Funktionsaufrufe umfasst. Anwendung findet XPath 3.0 aktuell vor allem in XQuery 3.0 Prozessoren.

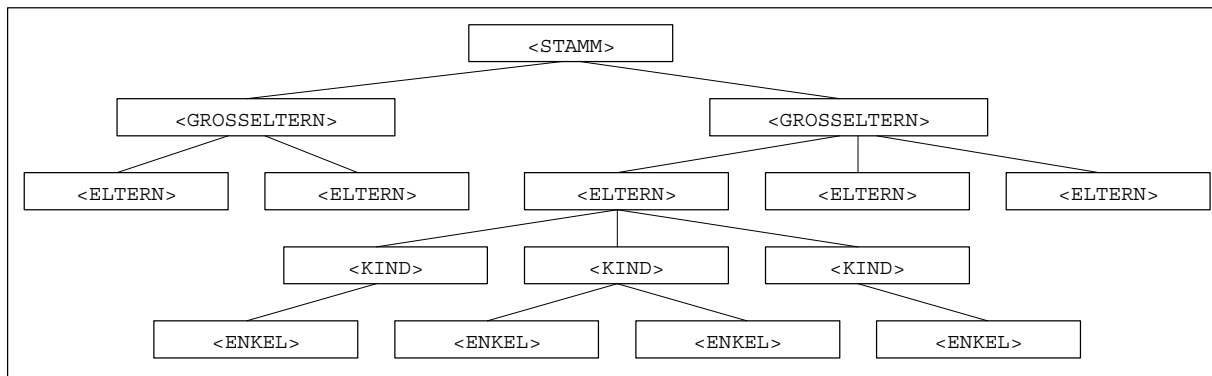
Baumstrukturen

XPath stellt ein XML-Dokument als einen Baum dar, der aus Knoten besteht. Dabei gibt es verschiedene Knotentypen:

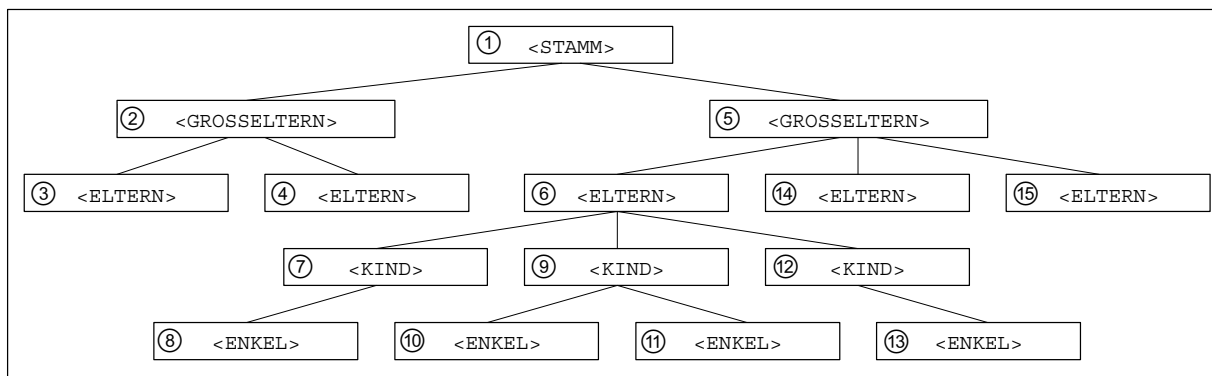
- | | | | | | | | |
|---|-----------------|---|------------------|---|-------------------------------|---|------------|
| ✓ | Wurzelknoten | ✓ | Elementknoten | ✓ | Attributknoten | ✓ | Textknoten |
| ✓ | Kommentarknoten | ✓ | Namensraumknoten | ✓ | Verarbeitungsanweisungsknoten | | |

Damit können Sie über festgelegte Kriterien ganz bestimmte Knoten eines XML-Dokuments selektieren und ausgeben.

Zum besseren Verständnis der nachfolgenden XPath-Prinzipien wurde die XML-Struktur in eine Art Familie aufgeteilt. Dies macht die Erklärung der Eltern-Kind-Beziehungen der Elemente verständlicher.



Die Anordnung der Komponenten des XML-Dokuments innerhalb der Baumstruktur erfolgt nach festen Regeln. Bei einem kompletten Durchlauf durchlaufen (traversieren, engl. to traverse) XML-Prozessoren den Baum entsprechend der Dokumentreihenfolge, wie sie im zugrundeliegenden XML-Dokument vorliegt. Dabei wird immer eine **Tiefensuche** angewendet. Bei der Tiefensuche werden immer zuerst die Teilbäume komplett durchlaufen:



XPath-Datentypen

Als Ergebnis eines XPath-Ausdrucks wird ein Objekt zurückgegeben, welches in XPath 1.0 einen von vier Datentypen besitzen kann:

node-set	Knotenmenge
string	Zeichenfolge
number	Fließkommazahl (nach IEEE 754)
boolean	boolescher Wert, entweder <code>true</code> oder <code>false</code>

Für Umwandlungen zwischen den einzelnen Typen enthält die Empfehlung feste Regeln.

Entitäten können von XPath nicht selektiert werden, da ein XPath-Ausdruck das vom Prozessor fertig geparste Dokument auswertet. Entitäten sind zu diesem Zeitpunkt jedoch schon durch ihre zugehörige Zeichenfolge ersetzt.



10.2 XML-Prinzipien

Lokalisierungsstufen

Eine Lokalisierung der unterschiedlichen Stufen innerhalb der XML-Struktur besteht aus drei Teilen:

- ✓ einer **Achse** (engl.: Axis Specifier) zum Navigieren in der XML-Baumstruktur,
- ✓ einer **Knotenprüfung** (engl.: Node Test), um über weitere Kriterien bestimmte Knoten zu selektieren,
- ✓ keinem oder mehreren **Prädikaten**, um die Auswahl der bisher selektierten Knoten genauer zu filtern.

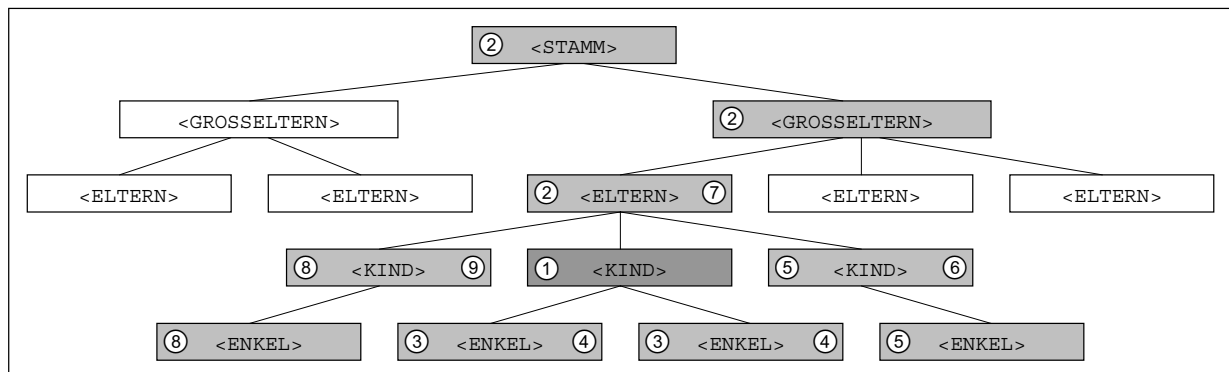
Die Angabe der Achse und die Knotenprüfung sind Pflichtangaben. Die Angabe des Prädikats ist optional.

```
achse::knotenprüfung[prädikat]
```

Die Achsen- und die Knotenprüfungsangabe werden durch zwei Doppelpunkte `::` voneinander getrennt angegeben. Das Prädikat wird in eckige Klammern `[]` eingeschlossen.

Achsen

XPath bietet sehr umfangreiche Möglichkeiten zum Selektieren bestimmter XML-Knoten. Zum Navigieren in einem Dokumentenbaum nutzt XPath verschiedene Achsen. In der nachfolgenden Abbildung und Tabelle finden Sie alle zur Verfügung stehenden Achsen, mit denen Sie eine Menge von Knoten bezüglich ihrer Position im XML-Baum relativ zum gegebenen Kontext-Knoten ① bestimmen können.



Achse	Erläuterung
self ①	Das aktuelle Element wird selektiert (Kontext-Knoten).
ancestor ②	Vorfahre; ein Vorfahre kann hierbei das Eltern-Element oder das Großeltern-Element sein, bis hin zur Wurzel.
ancestor-or-self ② ①	Vorfahre oder aktueller Kontext-Knoten; die Achse enthält auch immer das Hauptelement.
attribute	Hiermit werden die Attribute des Kontext-Knotens bestimmt.
child ③	Die Kind-Elemente des Kontext-Knotens werden angesprochen.
descendant ④	Nachfahre; ein Nachfahre kann hierbei ein Kind oder das Kind eines Kindes (Enkel-Element) sein.
descendant-or-self ④ ①	Nachfahre oder aktuelles Element; ein Nachfahre kann hierbei ein Kind oder das Kind eines Kindes (Enkel-Element) sein.

Achse	Erläuterung
following ⑤	Nachfolgende Knoten mit deren Kindern; hierbei werden innerhalb des Dokuments die nachfolgenden Knoten angesprochen, die sich auf derselben Ebene des Kontext-Knotens befinden, sowie deren entsprechende Kind-Elemente. Attribut- oder Namensraumknoten werden außer Acht gelassen.
following-sibling ⑥	Nachfolgende Knoten der gleichen Ebene desselben Eltern-Knotens; ist der Kontext-Knoten ein Attribut- oder Namensraumknoten, ist die Anzahl der folgenden Geschwister gleich null.
namespace	Der Namensraum des Kontext-Knotens wird angesprochen.
parent ⑦	Das Eltern-Element wird selektiert.
preceding ⑧	Vorherige Knoten mit deren Kindern; hierbei werden innerhalb des Dokuments die vorherigen Knoten des Kontext-Knotens angesprochen, wobei Attribut- oder Namensraumknoten außer Acht gelassen werden.
preceding-sibling ⑨	Vorherige Geschwister desselben Eltern-Knotens; ist der Kontext-Knoten ein Attribut- oder Namensraumknoten, ist die Anzahl der vorherigen Geschwister gleich null.

Die Achsen `ancestor`, `descendant`, `following`, `preceding` und `self` zerteilen ein Dokument. Sie überschneiden sich niemals und enthalten zusammen alle Knoten eines Dokuments, ohne dass ein Knoten doppelt vorkommt, wobei die Attribut- und Namensraumknoten nicht beachtet werden.



Selektierungspfad

Zum Selektieren bestimmter Baumknoten werden bei den Elementnamen bestimmte Zeichen vorangestellt. Sie stellen die sogenannten Stufen zum Selektieren der Elemente dar.

Die gekürzte Syntax eines Selektierungspfades (engl. Location Paths) lautet:

Zeichen	Bedeutung
(kein Zeichen)	Nachkomme eines Knotens
@	Attribut des selektierten Elements
/	Hauptelement
//	Selektierter Knoten oder dessen Nachkomme
.	Selektierter Knoten
..	Vorfahre des selektierten Knotens
*	Beliebige Knoten

Die in den nachfolgenden Beispielen verwendeten Angaben `xsl:for-each` und `xsl:value-of` sind Elemente der Formatierungssprache XSL und werden erst im nächsten Kapitel näher erläutert. Nehmen Sie daher die jetzige Angabe erst einmal als gegeben hin.



Beispiel: *kap10\xpath1.xsl*

Die Beispiele beziehen sich auf die bisher verwendete Musiksammlung. Es werden alle Lieder der Alben gesucht, die somit Kinder des Elements `ALBUM` sind.

```
<xsl:for-each select="//child::ALBUM/child::LIED">
  <xsl:value-of select="." />
</xsl:for-each>
```

Es werden die Albumtitel selektiert, die über das Attribut `stil` eine Stilrichtung besitzen.

```
<xsl:for-each select="//TITEL/attribute::stil">
  <xsl:value-of select="./parent::TITEL" />
</xsl:for-each>
```

Es werden die Elemente mit der Bezeichnung `TITEL` gefunden, die zusätzlich ein Attribut `stil` mit dem Wert `Pop` besitzen.

```
<xsl:for-each select="//TITEL[@stil='Pop']">
  <xsl:value-of select="." />
</xsl:for-each>
```

Es werden alle Jahrangaben der Alben ausgelesen, ausgehend vom Element `TITEL`, das ein Nachfahre des selektierten Elements ist.

```
<xsl:for-each select="//TITEL/ancestor::*">
  <xsl:value-of select="./JAHR" />
</xsl:for-each>
```

Abkürzungen

- ✓ Die Angabe der Achse `child::` kann weggelassen werden, da die Kind-Achse des selektierten Knotens standardmäßig immer angesprochen wird.
- ✓ Die Achse `attribute` zum Selektieren eines Attributknotens kann mit dem Zeichen `@` abgekürzt werden (z. B. `attribute::stil` wird zu `@stil`).

Zur besseren Veranschaulichung wird eine XML-Struktur aufgebaut, bei der jedes Element jeweils nur ein Unterelement besitzt.

①	<STAMM>
②	<GROSSELTERN>
③	<ELTERN>
④	<KINDER attr="m">
⑤	<ENKEL attr="w">
	</ENKEL>
	</KINDER>
	</ELTERN>
	</GROSSELTERN>
	</STAMM>

Für die nachfolgenden Beispiele eines Selektierungspfades wird angenommen, dass das Element `<ELTERN>` ③ selektiert ist.

Pfad	Erklärung
KINDER	Der Nachkomme von <code>ELTERN</code> ③ ist <code>KINDER</code> ④. Das Element <code>KINDER</code> wird gefunden.
* / ENKEL	Mit <code>*</code> wird der Knoten ④ übersprungen, sodass der <code>ENKEL</code> ⑤ gefunden wird.
/ ELTERN / KINDER	Das Hauptelement ist nicht das Element <code>ELTERN</code> , sodass die angegebenen Elemente nicht gefunden werden.
// KINDER [@attr='m']	Der selektierte Knoten ist <code>ELTERN</code> ③ und gleichzeitig der direkte Vorfahre von <code>Kinder</code> ④. Da dieses Element auch noch das Attribut <code>attr</code> mit dem Wert <code>m</code> aufweist, wird auch dieses Element angesprochen.
.. / ELTERN / KINDER	Der Vorfahre des Knotens ③ ist <code>GROSSELTERN</code> . Die nachfolgenden Unterelemente <code>ELTERN</code> und <code>KINDER</code> werden gefunden.
./ ENKEL	Der selektierte Knoten lautet <code>ELTERN</code> ③. Dieser besitzt keinen direkten Nachkommen <code>ENKEL</code> .

Knotenprüfung

Nachdem Sie mit der Angabe der Achse die Richtung der Auswahl eines Knotens gewählt haben, können Sie mit der Knotenprüfung ein weiteres Kriterium zur Vorauswahl eines Elements angeben.

Die Knotenprüfung trifft anhand der vorgegebenen Suchachse eine Auswahl der zu selektierenden Elemente. Die Angabe zur Prüfung kann hierbei der direkte Elementname sein.

Beispiel: *kap10\xpath2.xsl*

Diese Angabe selektiert im gesamten Dokument alle Elementknoten mit dem Namen LIED.

```
<xsl:for-each select="/descendant::LIED">
  <xsl:value-of select="." />
</xsl:for-each>
```

Das Zeichen `*` kennzeichnet, dass alle Elemente unterhalb des selektierten Knotens ausgewählt werden sollen. Im folgenden Beispiel sind es alle Kinder des Elements ALBUM.

```
<xsl:for-each select="//ALBUM/child::*">
  <xsl:value-of select="." />
</xsl:for-each>
```

Auch die Suche nach einem bestimmten Knotentyp ist möglich.

Knotentypen	Erläuterung
<code>node()</code>	Es werden die Knoten ausgewählt, die über die selektierte Achse erreichbar sind. Dabei werden auch die Namensraum- und Attributknoten einbezogen.
<code>comment()</code>	Einen Kommentar innerhalb einer XML-Struktur selektieren Sie mit diesem Knotentyp. Die Knoten werden auch Kommentarknoten genannt.
<code>text()</code>	Diese Anweisung ermöglicht entlang der Achse das Selektieren von Knoten, die einen Inhalt (Text) haben.
<code>processing-instruction()</code>	Damit wählen Sie die Prozessoranweisungen der XML-Datei aus. Beispielsweise ist die Angabe von <code><?xml-stylesheet ...?></code> eine solche Anweisung.
<code>processing-instruction("xyz")</code>	Mit der Angabe eines Namens können Sie spezielle Prozessoranweisungen der Form <code><?xyz ...?></code> auswählen.

Beispiel: *kap10\xpath3.xsl*

Alle Kinder des Elements ALBUM, die keine leeren Elemente sind, werden ausgewählt.

```
<xsl:for-each select="//ALBUM/child::*[text()]">
  <xsl:value-of select="." />
</xsl:for-each>
```

Es werden alle Elemente INTERPRET des selektierten Knotens ausgewählt.

```
<xsl:for-each select="/descendant-or-self::node()/child::INTERPRET">
  <xsl:value-of select="." />
</xsl:for-each>
```

Abkürzungen

- ✓ Den aktuellen Knoten `self::node()` können Sie mit `.` bezeichnen.
- ✓ Für den übergeordneten Knoten `parent::node()` können Sie zwei Punkte `..` angeben.
- ✓ Einen Knoten sowie alle seine Nachkommen wählen Sie mit den beiden Zeichen `/` aus, statt der Angabe von `/descendant-or-self::node()`.

Beispielsweise lässt sich die nachfolgende Angabe einfach abkürzen:

```
Aus
<xsl:for-each select="/descendant-or-self::node()/child::INTERPRET">
</xsl:for-each>
wird
<xsl:for-each select="//INTERPRET">
</xsl:for-each>
```

Prädikate

Für alle Knoten, die nach der Achsenbestimmung und der Knotenprüfung noch ausgewählt sind, können Sie spezielle Prädikate auswerten. Prädikate sind Ausdrücke, die einen Wert liefern und die selektierten Knoten weiter filtern.

Neben dem Vergleich von Zeichenketten können die Filter auch logische und arithmetische Abfragen beinhalten.

Abfragen	Mögliche Operanden
Logik	or, and, !, <, >, <=, >=, =, !=
Arithmetik	+, -, *, div, mod



Da in XML die Zeichen `<` und `>` als Einleitung und Abschluss eines Tags betrachtet werden, müssen Sie die beiden Zeichen jeweils als Entity `<` sowie `>` angeben.

Im folgenden Beispiel werden über die Abfrage des Knotentyps `text()` und den Logikausdruck `or` die Alben der Interpreten Moby und Philip Glass selektiert.

```
<xsl:for-each select="//ALBUM/INTERPRET[text()='Moby' or text()='Philip Glass']">
  <xsl:value-of select="..parent::node()/TITEL" />
</xsl:for-each>
```

Funktionen

XPath 1.0 umfasst eine Reihe von Funktionen. Analog zu den vorhandenen Datentypen gruppieren sich diese in die vier Bereiche

- ✓ node-set
- ✓ string
- ✓ boolean
- ✓ number

In der nachfolgenden Tabelle sind zwei von XPath definierte Funktionen zum Filtern von Knoten aufgelistet. Sie dienen dazu, die Position eines bestimmten Knotens sowie die Gesamtanzahl der Knoten einer Ebene zu ermitteln.

Eigenschaft	Erläuterung
<code>last()</code>	Liefert die Position des letzten Knotens der Ebene und somit gleichzeitig die Anzahl der Knoten der entsprechenden Ebene
<code>position()</code>	Gibt die aktuelle Position des Knotens in der selektierten Ebene aus

Beispiel: *kap10\xpath4.xsl*

Aus dem Dokument werden alle Elemente mit der Bezeichnung `ALBUM` selektiert und der Name des entsprechenden Interpreten wird ausgegeben. Das letzte Element `last()` wird hierbei außer Acht gelassen.

```
<xsl:for-each select="//ALBUM[position() != last()]">
  <br /><xsl:value-of select="./INTERPRET" />
</xsl:for-each>
```

Es wird, falls vorhanden, vom zweiten Element `BILD` des Oberelements `ALBUM` der Attributwert `quelle` aufgelistet.

```
<xsl:for-each select="//ALBUM/BILD[position() = 2]">
  <br /><xsl:value-of select="./@quelle" />
</xsl:for-each>
```

Von jedem Album werden die ersten drei Lieder selektiert und der Inhalt wird ausgegeben.

```
<xsl:for-each select="//LIED[position() <= 3]">
  <br /><xsl:value-of select="." />
</xsl:for-each>
```

Weitere Beispiele zur Selektierung von Elementen über XPath finden Sie auf der Informationsseite des W3-Konsortiums <http://www.w3.org/TR/xpath>, Abschnitt 2: Kapitel „Location Paths“.



10.3 XPath 2.0

Erweiterungen

Die Empfehlung XPath 2.0 ist eine Erweiterung von XPath 1.0. Ein Prozessor für XPath 2.0 verarbeitet über einen Kompatibilitätsmodus auch Ausdrücke der Version 1.0. Wichtige Neuerungen der Version 2.0 sind

- ✓ das erweiterte Datenmodell
- ✓ neue Operatoren
- ✓ neue Ausdrucksstrukturen
- ✓ die stark erweiterte Funktionsbibliothek
- ✓ neue Datentypen

Datenmodell

Im Unterschied zur Version 1.0 umfasst das Datenmodell von XPath 2.0 neben den Baumstrukturen auch **Einzelwerte** (sogenannte atomic values) und **Sequenzen** bzw. Listen. Bei den Einzelwerten kann es sich um verschiedene Daten handeln, beispielsweise um Zeichenfolgen, Zahlen, logische Werte oder Datums- bzw. Zeitwerte. Sequenzen bestehen aus einer geordneten Folge von Bezügen auf Knoten und einzelnen Werten. Eine Schachtelung der Sequenzen ist dabei nicht möglich, sie sind immer flach.

Aus der Erweiterung des Datenmodells ergeben sich viele Erweiterungen hinsichtlich der Operationen und Funktionen. So gibt es beispielsweise Möglichkeiten zur Erzeugung neuer Einzelwerte und Sequenzen.

Im Gegensatz zu XPath 1.0, bei dem das Ergebnis eines Ausdrucks eine ungeordnete Knotenmenge ist, ergibt die Auswertung in der Version 2.0 eine Sequenz.

Ausdrucksstrukturen

Mit der Erweiterung des Datenmodells um Sequenzen wird der effektive Umgang mit diesen notwendig. Der `for`-Ausdruck ermöglicht dies. Bei seiner Ausführung auf eine Sequenz wird die im Rückgabedruck angegebene Operation auf alle Datenelemente dieser angewendet.

```
for $Bereichsvariable in <Bindungssequenz> return <Rückgabedruck>
```

- ✓ Die Bereichsvariable wird durch das Zeichen `$` gekennzeichnet.
- ✓ Das Schlüsselwort `in` gibt die Bindungssequenz an. Die Bindungssequenz kann ein Zahlenbereich aber auch ein XPath Ausdruck zur Ermittlung von Knoten sein.
- ✓ Nach dem Schlüsselwort `return` steht der auszuwertende Ausdruck.

So wird der Ausdruck

```
for $i in 1 to 4 return $i*$i
```

in die Sequenz 1, 4, 9, 16 ausgewertet.

Um alle Kindknoten des aktuellen Knotens zu ermitteln, kann der Ausdruck `for` wie folgt eingesetzt werden:

```
for $k in child::* return name($k)
```

Eine weitere neue Möglichkeit der Ausdrucksstrukturen in XPath 2.0 stellen bedingte Ausdrücke mit dem Konstrukt `if ... then ... else` oder die Überprüfung auf das Vorhandensein von Knoten in Abhängigkeit von Bedingungen mit `some` oder `every` dar.

Datentypen

Das Modell der Datentypen in XPath 2.0 basiert auf dem XML-Schema. Dabei stehen die Datentypen zur Darstellung des eigentlichen XML Dokuments nicht im Vordergrund. Umfangreiche Neuerungen gibt es vor allem bei

- ✓ numerischen Datentypen (neben `number` stehen `integer`, `decimal`, `float` und `double` zur Verfügung),
- ✓ Datentypen für Datums- und Zeitwerte,
- ✓ der Möglichkeit der Erstellung benutzerdefinierter Datentypen.

Eine vollständige Übersicht aller Datentypen befindet sich unter <http://www.w3.org/TR/xpath-datamodel/>:

für die Arbeit mit ganzen Zahlen	
idiv	Division von ganzen Zahlen
to	Angabe ganzzahliger Bereiche

In der Version 2.0 stehen für Vergleichsoperationen alternative Operationen zur Verfügung:

XPath 1.0	XPath 2.0	XPath 1.0	XPath 2.0
=	eq	<=	le
!=	ne	>	gt
<	lt	>=	ge

Für die verschiedenartigen neuen Datentypen existiert ebenfalls eine Vielzahl neuer Operatoren, z. B. für den Umgang mit Datum und Zeit.

Funktionen

XPath 2.0 enthält eine sehr große Anzahl neuer Funktionen. Neben Funktionen, deren Existenz sich aus dem neuen Datenmodell ergibt, wie beispielsweise für den Umgang mit Sequenzen, sind dies auch viele Funktionen, die einen besseren Umgang mit den bereits in der Version 1.0 vorhandenen Datentypen ermöglichen. Dies betrifft zum Beispiel die Möglichkeiten der Zeichenkettenoperationen inklusive der Anwendung von regulären Ausdrücken.

Eine komplette Liste der in XPath 2.0 vorhandenen Funktionen und Operatoren finden Sie hier: (<http://www.w3.org/TR/xquery-operators/>).

10.4 Übung

Elemente mit XPath selektieren

Übungsdateien: *kap10\musiksammlung.xml*,
kap10\musiksammlung.dtd

Ergebnisdateien: *kap10\uebung1.xml*,
kap10\uebung1.xsl, *kap10\uebung2.xml*,
kap10\uebung2.xsl, *kap10\uebung3.xml*,
kap10\uebung3.xsl, *kap10\uebung4.xml*,
kap10\uebung4.xsl

Die vorgegebene Musiksammlung besitzt folgende Struktur:

```
<MUSIKSAMMLUNG>
  <ALBUM>
    <AUTOR> </AUTOR>
    <INTERPRET> </INTERPRET> <!-- oder <GRUPPE> </GRUPPE> -->
    <TITEL stil="" bewertung=""> </TITEL>
    <GESAMTZEIT> </GESAMTZEIT>
    <LABEL> </LABEL>
    <JAHR> </JAHR>
    <BILD typ="" quelle="" />
    <LIED> </LIED>
  </ALBUM>
</MUSIKSAMMLUNG>
```

- ① Geben Sie einen XPath-Ausdruck an, der die Namen aller Interpreten und Gruppen im XML-Dokument selektiert.

Moby
a-ha
Mesh
Ludwig van Beethoven
Philip Glass

- ② Ermitteln Sie die Titel aller Alben, die vom Stil **Klassik** sind.

Ludwig van Beethoven - Berühmte Klavier-Sonaten
Philip Glass - Koyaanisqatsi

- ③ Ermitteln Sie zusätzlich die Namen der Autoren, die die klassischen Alben in die Sammlung aufgenommen haben.

Ludwig van Beethoven - Berühmte Klavier-Sonaten (Max Mustermann)
Philip Glass - Koyaanisqatsi (Heiko Schröder)

- ④ Gesucht werden sollen die Alben, die vor dem Jahr 1990 erschienen sind.

1988 - Ludwig van Beethoven - Berühmte Klavier-Sonaten
1983 - Philip Glass - Koyaanisqatsi

11 XSL und XSLT

In diesem Kapitel erfahren Sie

- ✓ aus welchen Komponenten XSL besteht
- ✓ wie einzelne Elementinhalte mittels XSLT ausgelesen werden
- ✓ wie Sie durch HTML-Befehle in XSL-Dateien eine formatierte Ausgabe erhalten

Voraussetzungen

- ✓ XML Kenntnisse
- ✓ Grundlegende HTML-Kenntnisse

11.1 Einführung in XSL

XSL Komponenten

XSL (eXtensible Stylesheet Language) erweitert die Möglichkeiten von XML. Es dient sowohl zur Transformation von einem Datenformat in ein anderes als auch zur Präsentation der Daten. Für die Darstellung im Browser kann als Ausgabeformat HTML gewählt werden. Daneben bestehen Möglichkeiten, die Daten beispielsweise für die Darstellung in Form einer PDF- oder Postscript-Datei aufzubereiten.

XSL ist eine Zusammenfassung mehrerer Empfehlungen des W3C. Ein Dokument mit einer kurzen Einordnung der Thematik und mit Verweisen auf die konkreten Empfehlungen der einzelnen Bestandteile und deren aktuelle Versionen ist im Internet auf den Seiten des W3C verfügbar (<http://www.w3.org/Style/XSL/>).

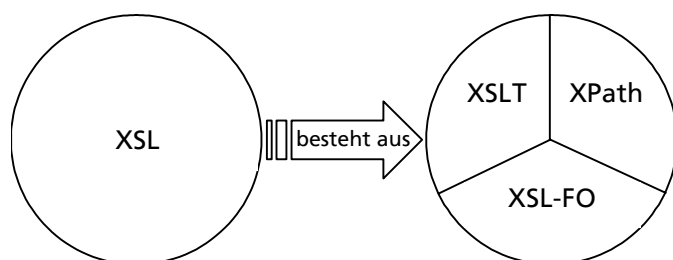
XSL untergliedert sich in drei Komponenten:

- ✓ **XSLT** (XSL Transformation) wird für die Auswertung und Umwandlung eines XML Ausgangsdokuments in ein neues Dokument verwendet. Für den Umwandlungsprozess können Regeln definiert werden. Diese ermöglichen sowohl die Auswahl von einzelnen Elementen und Attributen zur Übernahme in das Zieldokument als auch die Bearbeitung der Struktur des zu erstellenden Dokuments, bis hin zu einer im Vergleich zum Ausgangsdokument völligen Neugestaltung dieser.

Die Transformationsregeln werden in einem XSL Stylesheet festgelegt, welches dem zu bearbeitenden XML-Dokument zugeordnet wird. Dieses Stylesheet teilt dem verarbeitenden Prozessor mit, wie die logische Baumstruktur eines XML-Dokuments in eine repräsentative Struktur umgewandelt wird. Das XSL-Stylesheet selbst ist ebenfalls ein XML-Dokument und unterliegt damit den Regeln für deren Aufbau.

Die Empfehlung des W3C zu XSLT finden Sie unter <http://www.w3.org/TR/xslt20/>. Aktuell ist die Version XSLT 2.0 aus dem Jahr 2007.

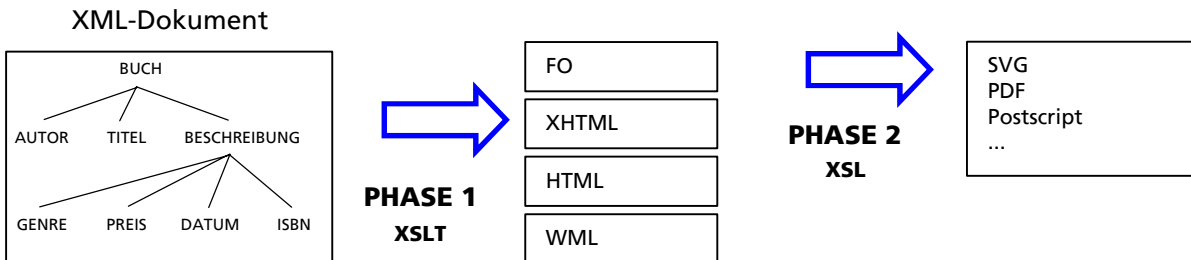
- ✓ **XPath** zur Selektion und zum Auswerten einzelner Teile eines XML Dokuments (vgl. Kapitel 10).
- ✓ **XSL-FO** ist die Formatierungssprache von XSL. Sie umfasst in einer XML-Syntax die wichtigsten Formatierungsanweisungen hinsichtlich Typographie und Layout zur Gestaltung von Dokumenten. Die Empfehlung des W3C zu XSL-FO 1.1 finden Sie unter <http://www.w3.org/TR/xsl/> (noch als Bestandteil der Empfehlung zu XSL) aus dem Jahr 2006. Des Weiteren gibt es einen Arbeitsentwurf für XSL-FO aus dem Jahr 2012 (<http://www.w3.org/TR/xslfo20/>).



Unterteilung von XSL

XSL-Phasen

Die Bearbeitung eines XML-Dokuments mit XSL besteht aus zwei Phasen.



Phase 1

Die erste Phase heißt **Baumtransformation**. In dieser Phase wird aus der XML-Baumstruktur unter Anwendung von XSLT und XPath eine weitere Baumstruktur erstellt. Wie die neue Struktur aufgebaut ist, wird durch Vorlagen bestimmt. Die Vorlagen (engl. Templates) bestehen aus verschiedenen Selektoren. Mit den Selektoren bestimmen Sie, welche Elemente angesprochen und in die neue Struktur übernommen werden sollen. Diese Struktur kann durch die Selektion der Daten sehr stark von der ursprünglichen Struktur abweichen. In der Transformationsphase werden die Formatierungen der Elemente noch nicht berücksichtigt.

Mithilfe von XSLT, den Vorlagen und einem XSLT-Prozessor können auch die XSL-FO Formatierungsobjekte wie Seiten, Absätze, Tabellen und die Formatierungseigenschaften wie Einrückungen, Zeilenabstände, Farbe, erzeugt werden.

Durch den Einsatz von HTML Befehlen in der `.xsl` Datei ist es möglich, als Ergebnis der Anwendung auf das XML-Dokument direkt ein HTML- oder XHTML-Dokument zu erzeugen. In diesem Fall entfällt die Phase zwei.

Phase 2

In der Phase 2 werden Formatierungsobjekte in eine druckbare Darstellung umgewandelt. Dazu ist eine weitere Software, der XSL-FO Prozessor notwendig. Apache-FOP ist eine kostenlose Software, die dies leistet. Sie basiert auf Java und ist somit plattformunabhängig. Sie finden die Software unter <http://xmlgraphics.apache.org/fop/>. Mit Apache-FOP sind Sie in der Lage, mithilfe von XSL-FO verschiedene Ausgabeformate, wie z. B. PDF, SVG, PostScript und Text zu erzeugen. Apache-FOP wurde hauptsächlich für die PDF-Ausgabe entwickelt.

Im Vergleich zur Formatierung eines Textes mit CSS ist XSL-FO deutlich komplexer. Daraus resultieren in der Anwendung sowohl an den Ersteller von XSL-Dokumenten als auch an die Entwickler von XSL-FO Prozessoren deutlich höhere Anforderungen, was eine Ursache für die noch nicht so weite Verbreitung von XSL-FO ist.

Für die Darstellung der erzeugten XML-Dokumente in Kapitel 11 und 12 werden HTML-Anweisungen in die XSL-Dateien integriert. Der XSL-Prozess beschränkt sich hier immer nur auf die erste Phase.



XSL als Namensraum

Ein XSL-Stylesheet ist formal ein wohlgeformtes XML-Dokument mit einem festgelegten Namensraum.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

Über das angelegte Präfix `xmlns:xsl` wird der Namensraum `xsl` definiert. Die Bezeichnung ist beliebig, es wird aber meistens das Kürzel `xsl` verwendet. XML-Elemente, die zu diesem Namensraum gehören, werden somit als Befehle interpretiert, die der XSLT-Prozessor verarbeiten soll. Alle anderen Tags, wie z. B. die bekannten HTML-Tags, werden in den nachfolgenden Beispielen unverändert an die Ausgabe im Browser weitergeleitet.

Im Umgangssprachegebrauch wird gelegentlich, obwohl dies nicht korrekt ist, XSLT mit XSL gleichgesetzt. Grund dafür ist u. a. einerseits die Verwendung des Namensraumes XSL bei der Anwendung von XSLT-Anweisungen und andererseits die Speicherform von XML-Dokumenten mit XSLT-Anweisungen als `.xsl` Dateien.



11.2 Einbinden einer XSL-Datei

Um den Inhalt eines XML-Dokuments über ein XSL-Stylesheet zu formatieren, fügen Sie unterhalb des XML-Prologs die Zeile für die Stylesheet-Deklaration ein. Die Vorlagen und Selektoren zum Auswählen bestimmter XML-Elemente werden in einer externen Datei definiert. Dem Dokument wird diese Datei, das externe Stylesheet, über eine Processing Instruction (PI) zugeordnet.

```
<?xml-stylesheet type="text/xsl" href="XSL-Datei"?>
```

Grundlegend erfolgt die Einbettung eines XSL-Stylesheets wie die Einbindung eines Cascading Stylesheets. Es muss lediglich als Wert des Attributs `href` der Name einer XSL-Datei und als Dateityp die Kennung `text/xsl` angegeben werden.

```
<?xml-stylesheet type="text/xsl" href="auswahl.xsl"?>
```

In diesem Beispiel wird in eine XML-Datei die XSL-Datei *auswahl.xsl* eingebunden, um die XML-Daten für die Ausgabe zu transformieren.

11.3 Templates

Template anlegen

Templates legen fest, welche Elemente einer XML-Struktur zur weiteren Verarbeitung genutzt werden sollen. Das XSL-Element zur Definition einer Vorlage lautet:

```
<xsl:template match="Pattern" name="foo">
  <!-- Aktionsteil -->
</xsl:template>
```

Diese Anweisung besagt, dass ein Element der Struktur selektiert werden soll, das dem angegebenen Muster (`Pattern`) entspricht. Der Aktionsteil legt fest, welche weiteren Arbeitsschritte folgen sollen.

Die Elemente in XSL sind die Vorlagen, welche über die Anweisung `xsl:template` erkennbar sind. Jedes Template beschreibt, für welches Element es in dem XML-Dokument gültig ist. Dieses Element, das auch Knoten genannt wird, wird über das Attribut `match` bestimmt. Der XSLT-Prozessor liest dabei die XML-Baumstruktur ein und vergleicht jedes XML-Element mit dem Attributwert `match`. Existiert ein solches Element, wird es über die Funktionen des Aktionsteils in die neue XML-Struktur übernommen. Die Adressierung erfolgt dabei über XPath.

Grundlegend kann diese Arbeitsweise folgendermaßen beschrieben werden:

- ✓ Wenn ein Element mit dem Namen, das zum Muster passt, im Dokument gefunden wird, dann führe den nachfolgenden Aktionsteil aus.
- ✓ Das Attribut `name` ist optional. Es legt eine eindeutige Bezeichnung für die Vorlage fest. Die Angabe von `foo` steht für einen zulässigen Namen.

Templates in Templates

Durch die Verschachtelung von XML-Elementen ist es notwendig, dass Sie innerhalb eines Templates auf andere Templates zugreifen können. Innerhalb des XSLT-Dokuments kann deshalb in andere Vorlagen verzweigt werden. Dies erreichen Sie im Aktionsteil einer Vorlage mit dem Aufruf des XSL-Elements `xsl:apply-templates`.

```
<xsl:apply-templates select="Pattern" />
```

Der Befehl selektiert alle Kind-Elemente des im Suchmuster angegebenen Dokumentknotens. Das Attribut `select` ist optional. Falls es nicht angegeben ist, werden alle Elemente des gewählten Suchmusters extrahiert. Der Befehl `xsl:apply-templates` ist immer nur unterhalb von `xsl:template` erlaubt.

Beispiel: *kap11/tutorial.xml*

Legen Sie die nachfolgende XML-Struktur an und binden Sie die XSL-Datei *tutorial.xsl* ein, um die Elemente `OBERELEMENT` und `SICHTBAR` anzusprechen. Das Element `UNSICHTBAR` soll im nachfolgenden Beispiel außer Acht gelassen werden.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="tutorial.xsl"?>

<OBERELEMENT>
  <SICHTBAR> Element, das angesprochen werden soll. </SICHTBAR>
  <UNSICHTBAR> Element, das NICHT angesprochen werden soll. </UNSICHTBAR>
</OBERELEMENT>
```

Beispiel: *kap11/tutorial.xsl*

Erstellen Sie die XSL-Datei, um die Daten der XML-Struktur zu selektieren und für jedes Element eine entsprechende Meldung auszugeben.

```
① <?xml version="1.0" encoding="iso-8859-1"?>
② <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
③   <xsl:template match="/">
    <html><head><title>XSL-Templates</title></head><body>
    <h3>Ansprechen einzelner XML-Elemente</h3>
④   <xsl:apply-templates />
    </body></html>
  </xsl:template>
⑤   <xsl:template match="OBERELEMENT">
⑥     <p>Die XML-Struktur enthält das Element OBERELEMENT.</p>
⑦     <xsl:apply-templates select="SICHTBAR" />
    </xsl:template>
⑧   <xsl:template match="UNSICHTBAR">
    <p>Die XML-Struktur enthält das Element UNSICHTBAR.</p>
    </xsl:template>
⑨   <xsl:template match="SICHTBAR">
⑩     <p>Die XML-Struktur enthält das Element SICHTBAR.</p>
⑪   </xsl:template>
</xsl:stylesheet>
```

- ① Der Prolog eines XML-Dokuments wird angegeben, da es sich bei einer XSL-Datei ebenfalls um ein XML-Dokument handelt.
- ② Die XSL-Datei wird als Stylesheet-Datei definiert und erhält zum eindeutigen Ansprechen der einzelnen Elemente den Namensraum `xsl`.
- ③ Über das erste Template wird das oberste Element der XML-Datei angesprochen.
- ④ Nach der Angabe einzelner HTML-Tags, die an den Browser weitergereicht werden, kontrolliert der XSLT-Prozessor, ob ein weiteres Element existiert. Findet er ein nächstes Element, verzweigt er in das Template, welches dem Namen des nächsten Elements entspricht. Der jeweilige Attributwert von `match` muss übereinstimmen. Das erste Element lautet `OBERELEMENT`, somit wird in das Template ⑤ verzweigt.

```
<OBERELEMENT>
  <SICHTBAR> Element, das angesprochen werden soll. </SICHTBAR>
  <UNSICHTBAR> Element, das NICHT angesprochen werden soll. </UNSICHTBAR>
</OBERELEMENT>
```

- ⑤ Da der Attributwert mit dem Namen des nächsten Elements übereinstimmt, wird der umschlossene Aktionsteil ausgeführt.
- ⑥ Der Aktionsteil des Templates besteht darin, das HTML-Tag `<p>` an den Browser zu übergeben. Mit dem hinterlegten Inhalt wird bekannt gegeben, dass das gesuchte Element `OBERELEMENT` gefunden wurde.
- ⑦ Es wird auf das nächste Element verwiesen. Durch die Angabe des Attributs `select` muss das nächste Element `SICHTBAR` heißen. Existiert es, verzweigt der XSLT-Prozessor in das entsprechende Template ⑨.
- ⑧ In dieses Template wird verzweigt, wenn nach dem XML-Knoten `UNSICHTBAR` gesucht wird. In diesem Beispiel wird das Template jedoch nicht angesprochen, sodass an dieser Stelle keine Ausgabe erfolgt.
- ⑨ Dieses Template wird ausgeführt, wenn nach dem Element `SICHTBAR` gesucht wird.

```
<OBERELEMENT>
  <SICHTBAR> Element, das angesprochen werden soll. </SICHTBAR>
  <UNSICHTBAR> Element, das NICHT angesprochen werden soll. </UNSICHTBAR>
</OBERELEMENT>
```

- ⑩ Durch die Ausgabe von `<p>` wird gezeigt, dass das gesuchte Element `SICHTBAR` gefunden wurde.
- ⑪ Die Vorlage wird geschlossen. Der Prozessor kehrt über das Template ⑤ zur Stelle ④ zurück. Da keine weiteren Anweisungen folgen, wird die Transformation der Daten beendet.

Ansprechen einzelner XML-Elemente

Die XML-Struktur enthält das Element `OBERELEMENT`.

Die XML-Struktur enthält das Element `SICHTBAR`.

Ausgabe der XML-Daten über die XSL-Transformation im Browser

Spezielle Vorlage ansprechen

Standardmäßig wird über die XSL-Anweisung `<xsl:apply-templates />` die nächste Vorlage aufgerufen und ausgeführt. Mit der optionalen Vergabe eines Template-Namens über die Anweisung `<xsl:template match="Pattern" name="foo">` haben Sie die Möglichkeit, gezielt eine bestimmte Vorlage aufzurufen. Das XSL-Element zum Ansprechen des Templates lautet:

```
<xsl:call-template name="foo" />
</xsl:call-template>
```

Mit der zusätzlichen Angabe des Attributs `name` legen Sie fest, dass nicht die nächste, sondern eine unter diesem Namen angelegte Vorlage aufgerufen werden soll. Die Angabe von `foo` steht hier für einen zulässigen Namen.

Beispiel: *kap11\call_template.xsl*

Das gezielte Ausführen von Vorlagen wird über die Funktion `xsl:call-template` gesteuert, indem mit dem Attribut `name` bezeichnete Vorlagen direkt aufgerufen werden.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><head><title>call-template</title></head>
    <body>
      ① <xsl:call-template name="vorlage2">
      ② </xsl:call-template>
    </body></html>
  </xsl:template>
```

```

③ <xsl:template match="OBERELEMENT" name="vorlage1">
    <!-- Aktionsteil -->
</xsl:template>

④ <xsl:template match="ELEMENT" name="vorlage2">
    <!-- Aktionsteil -->
</xsl:template>
</xsl:stylesheet>

```

- ① Diese Anweisung ruft die Vorlage mit der Bezeichnung `vorlage2` auf.
- ② Der Aufruf des Templates wird abgeschlossen. Es kann auch die verkürzte Schreibweise für ein leeres Element verwendet werden (`<xsl:call-template name="vorlage2" />`).
- ③ Dieses Template erhält den Namen `vorlage1`. Innerhalb dieses Beispiels wird es nicht aufgerufen.
- ④ Dies ist das Template `vorlage2`, das von ① aufgerufen und ausgeführt wird.

11.4 Selektion mit Filter in XPath

Der Mustervergleich von Elementen beruht auf der Sprache XPath. Sie stellt die verschiedenen Abfragemechanismen zur Verfügung und steuert somit den Zugriff auf die XML-Elemente. Mithilfe von XPath können Sie nicht nur den Namen eines Elements vergleichen, sondern auch speziellere Abfragen zu Elementkombinationen, Attributnamen und Attributwerten gestalten.

Beispiel: *kap11xpath.xml*

Für die Auflistung der möglichen Mustervergleiche wird die Struktur der erweiterten Musiksammlung herangezogen.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<MUSIKSAMMLUNG>
① <ALBUM typ="CD">
② <INTERPRET>Moby</INTERPRET>
③ <TITEL stil="Pop">Play</TITEL>
<GESAMTZEIT>63:03</GESAMTZEIT>
<LABEL>Mute Records Limited</LABEL>
<ERSCHEINUNGSJAHR>1999</ERSCHEINUNGSJAHR>
④ <LIED>
<LIEDTITEL>Honey</LIEDTITEL>
⑤ <TITELZEIT>3:28</TITELZEIT>
</LIED>
<LIED>
<LIEDTITEL>Find my baby</LIEDTITEL>
⑥ <TITELZEIT>3:59</TITELZEIT>
</LIED>
</ALBUM>
⑦ <ALBUM typ="Vinyl">
<!-- ... -->
</ALBUM>
</MUSIKSAMMLUNG>

```

<code><xsl:template match="..."></code>	findet
ALBUM	die Elemente <code><ALBUM></code> a ① ⑦
ALBUM/INTERPRET	die Elemente <code><INTERPRET></code> , die direkte Unterknoten (Kind, Child) des Elements <code><ALBUM></code> sind ②
ALBUM//TITELZEIT	alle Elemente <code><TITELZEIT></code> , die mittelbare oder unmittelbare Kinder des Elements <code><ALBUM></code> sind ⑤ ⑥
ALBUM/LIED[1]	das erste Element <code><LIED></code> , das ein Unterelement des Elements <code><ALBUM></code> ist ④. In eckigen Klammern wird die Nummer des Elements angegeben.
//LIED	die Elemente <code><LIED></code> in beliebiger Tiefe im Dokument
./LIED	die Elemente <code><LIED></code> in beliebiger Tiefe von der vorherigen Selektion aus gesehen (nicht im ganzen Dokument)
ALBUM/*/TITELZEIT	die Elemente <code><TITELZEIT></code> , die einen Vorfahren vom Typ <code><ALBUM></code> haben ⑤ ⑥. Das Zeichen <code>*</code> steht hierbei für ein beliebiges dazwischen befindliches Element.
ALBUM/TITEL[@stil]	die Elemente <code><TITEL></code> mit dem Attribut <code>stil</code> , die gleichzeitig Kinder des Elements <code><ALBUM></code> sind ③
ALBUM/TITEL[@stil='Pop']	die Elemente <code><TITEL></code> mit dem Attribut <code>stil</code> und dem Attributwert <code>Pop</code> , die gleichzeitig Kinder des Elements <code><ALBUM></code> sind ③
ALBUM/INTERPRET GRUPPE	die Unterelemente <code><INTERPRET></code> oder <code><GRUPPE></code> des Elements <code><ALBUM></code> ②

11.5 Inhalte der Elemente ausgeben

Mithilfe der Templates und der Filter von XPath haben Sie eine bestimmte XML-Struktur selektiert. Die entsprechenden Knoten des Baumes werden zwar selektiert, jedoch wird kein Wert eines Elements ausgegeben. Sie müssen dem XSL-Prozessor mitteilen, welche Daten der gewählten Struktur verwendet werden sollen.

Der Befehl zum Einfügen eines Elementinhalts lautet:

```
<xsl:value-of select="Pattern" />
```

Der mit `xsl:template` selektierte Knoten der XML-Baumstruktur wird mit dem angegebenen Pattern verglichen. Existiert ein Element, auf das das Muster zutrifft, wird dessen Inhalt an das zu verarbeitende Programm weitergegeben.

Beispiel: *kap11/tutorial2.xml*

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="tutorial2.xsl"?>

<TEXT>
  <RED>XSL machts möglich...</RED>
  <BOLD>Die Daten werden fett dargestellt oder</BOLD>
  <ITALIC> kursiv, je nach angelegter Transformation.</ITALIC>
</TEXT>
```

Die XML-Datei besteht aus der Struktur mit dem Hauptelement `TEXT` und dessen drei Unterelementen `RED`, `BOLD` und `ITALIC`. Die eingebundene XSL-Datei *tutorial2.xsl* enthält die Befehle zum Auswählen der jeweiligen Elemente.

Beispiel: *kap11/tutorial2.xsl*

Das Beispiel zeigt Ihnen die Wirkungsweise des XSL-Stylesheets `xsl:value-of`. Als Ergebnis wird hier der Inhalt der Elemente `RED` und `ITALIC` ausgegeben.

```

① <?xml version="1.0" encoding="iso-8859-1"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
②    <xsl:template>
      <html><head><title>Elementinhalte ausgeben</title></head><body>
③      <xsl:value-of select="//RED" /> <br />
④      <xsl:value-of select="//ITALIC" /> <br />
      </body></html>
⑤    </xsl:template>
  </xsl:stylesheet>

```

- ① Der Namensraum `xsl`, der im gesamten Dokument Gültigkeit haben soll, wird definiert.
- ② Wenn kein Pattern für die Selektion eines Elements angegeben ist, wird standardmäßig das Hauptelement der XML-Datei *tutorial2.xml* selektiert.
- ③ Der Inhalt des Elements `RED`, das sich unterhalb des selektierten Elements `TEXT` befindet, wird an das Programm, in diesem Fall an den Browser, weitergegeben.
- ④ Ebenso wird der Inhalt des Elements `ITALIC` ausgegeben.
- ⑤ Die Vorlage wird geschlossen und somit die Auswahl der Elemente beendet.

XSL macht's möglich...
kursiv, je nach angelegter Transformation.

Ergebnis der Datenselektion im Browser

Zur Formatierung der Dateninhalte im Browser können Sie die üblichen HTML-Tags sowie Stylesheet-Angaben verwenden. Ebenso ist es möglich, die Reihenfolge der ausgegebenen XML-Daten zu beeinflussen, indem Sie die Elemente in unterschiedlicher Reihenfolge selektieren.

Beispiel: *kap11/tutorial3.xsl*

```

① <?xml version="1.0" encoding="iso-8859-1"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
②    <xsl:template match="/">
      <html><head><title>Elementinhalte ausgeben</title></head><body>
③      <p><b><xsl:value-of select="//BOLD" /></b></p>
④      <p><i><xsl:value-of select="//ITALIC" /></i></p>
⑤      <h3 style="color:red"><xsl:value-of select="//RED" /></h3>
      </body></html>
⑥    </xsl:template>
  </xsl:stylesheet>

```

- ① Der Namensraum `xsl`, der im gesamten Dokument Gültigkeit haben soll, wird definiert.
- ② Durch die Angabe des Pfads `/` wird das Hauptelement `TEXT` der XML-Datei *tutorial3.xml* selektiert.
- ③ Der Inhalt des Elements `BOLD`, das sich unterhalb des selektierten Elements `TEXT` befindet, wird an den Browser weitergegeben. Durch die Angabe der HTML-Elemente `P` und `B` wird der Dateninhalt als fett formatierter Absatz angezeigt.
- ④ Der Inhalt des Unterelements `ITALIC` wird kursiv dargestellt.
- ⑤ Dem Dateninhalt des Elements `RED` wird eine Überschrift der dritten Ordnung zugewiesen, die über die Stylesheet-Angabe `style="color:red"` in roter Farbe dargestellt wird.
- ⑥ Die Vorlage wird geschlossen und somit die Auswahl der Elemente beendet.

Die Daten werden fett dargestellt oder
kursiv, je nach angelegter Transformation.

XSL machts möglich...

Anzeige der formatierten Daten im Browser

11.6 Reihenfolge der Template-Aufrufe

Anhand eines Beispiels sollen die Auswirkungen der verschiedenen Template-Aufrufe gezeigt werden.

Beispiel: *kap11/test.xml*

Zur besseren Visualisierung der Ergebnisse werden die einzelnen Elemente mit Buchstaben und die Inhalte mit einer entsprechenden Zahl angegeben. So steht [b1] für das erste Element b, [b2] für das zweite Element b, [b1-c1] für das erste Element c, das ein Kind des ersten Elements b ist, usw.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="test.xsl"?>

<a>
  <b> [b1] </b>
  <bc>
    <c> [b1-c1] </c>
    <c> [b1-c2] </c>
  </bc>
  <b> [b2] </b>
  <d> [d1] </d>
  <d> [d2] </d>
</a>
```

Beispiel: *kap11/test.xsl*

Mit dieser XSL-Datei sollen verschiedene Templates ausgeführt und somit verschiedene Elemente ausgegeben werden.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

①  <xsl:template match="a">
    <html><head><title>Template-Test</title></head>
    <body>
②    <xsl:apply-templates />
    </body></html>
  </xsl:template>

③  <xsl:template match="b">
    <p>b = <xsl:value-of select="." /> </p>
    <xsl:apply-templates select="c" />
  </xsl:template>

④  <xsl:template match="bc">
    <p>bc = <xsl:value-of select="." /> </p>
    <xsl:apply-templates />
  </xsl:template>
```

⑤	<pre> </xsl:template> <xsl:template match="c"> <p>c = <xsl:value-of select="." /> </p> </xsl:template> </xsl:stylesheet> </pre>
---	---

- ① Das Hauptelement *a* der XML-Datei wird selektiert.
- ② Mit `apply-templates` wird in die anderen Vorlagen verzweigt. Wenn Sie diesen Aufruf nicht angeben, werden die anderen Vorlagen übersprungen. Im weiteren Verlauf soll dieser Aufruf abgewandelt und das entsprechende Ergebnis sichtbar gemacht werden.
- ③ Dies ist das erste Template, mit dem die Unterelemente *b* selektiert werden, ausgehend vom aktuellen Element. Die Inhalte der Elemente werden ausgegeben und es wird in das nächste Template *c* verzweigt.
- ④ Dieses Template selektiert das Element *bc* und gibt dessen Inhalt aus. Da in der bereits festgelegten XML-Datei zwei Unterelemente *c* festgelegt sind, werden deren Inhalte ausgegeben.
- ⑤ Mit dem Template werden die Unterelemente nochmals separat selektiert und deren Inhalte einzeln ausgegeben.

```

b = [b1]
bc = [b1-c1] [b1-c2]
c = [b1-c1]
c = [b1-c2]
b = [b2]
[d1] [d2]

```

Ausgabe der XSL-Anweisungen

Durch den nicht namentlich angegebenen Template-Aufruf in Zeile ② werden sämtliche Unterelemente von dem Element *a* aufgelistet. Es werden zum Schluss die Inhalte der beiden Elemente *d* ausgegeben, obwohl diese nicht speziell über ein Template angesprochen werden.

In der nachfolgenden Tabelle wird die jeweilige Ausgabe im Browser dargestellt, wenn Sie den Template-Aufruf in Zeile ② ändern.

Template-Aufruf	Ausgabe im Browser
<code><xsl:apply-templates select="b" /></code>	<pre> b = [b1] b = [b2] </pre>

Die Unterelemente *b* werden selektiert und deren Inhalte ausgegeben. Es werden auch noch die anderen Templates aufgerufen. Da die aktuellen Elemente *b* jedoch keine weiteren Unterelemente besitzen, wird nicht in diese Templates verzweigt.

Template-Aufruf	Ausgabe im Browser
<code><xsl:apply-templates select="bc" /></code>	<pre> bc = [b1-c1] [b1-c2] c = [b1-c1] c = [b1-c2] </pre>

Ausgehend vom Hauptelement *a* werden die Unterelemente *bc* selektiert und die Werte ausgegeben. Danach wird das nächste Template aufgerufen. Da das Element *bc* die Unterelemente *c* besitzt, wird dieses Template ausgeführt.

Template-Aufruf	Ausgabe im Browser
<code><xsl:apply-templates select="c" /></code>	--nichts--

Da sich im Hauptelement kein direktes Unterelement *c* befindet, wird nichts ausgegeben.

Template-Aufruf	Ausgabe im Browser
<code><xsl:apply-templates select="*/c" /></code>	<pre> c = [b1-c1] c = [b1-c2] </pre>

Die Angabe des Zeichens \square steht für ein beliebiges Oberelement von c . Die Inhalte der beiden Elemente c werden ausgegeben, weil b als Unterelement von a die Unterlemente c besitzt.

11.7 Übungen

Übung 1: Theoretische Frage zu XSL

Übungsdatei: --

Ergebnisdatei: *kap11\uebung1-3.html*

- ① Nennen Sie die Processing Instruction zum Einbinden einer XSL-Datei in eine XML-Datei.
- ② Erklären Sie, wie ein XSL-Namensraum innerhalb der XSL-Datei erstellt wird.
- ③ Nachfolgend sind verschiedene Selektionsfilter aufgelistet. Erläutern Sie, welche Elemente hiermit selektiert werden.

`//ELEMENT1`

`.//ELEMENT2`

`ELEMENT1/ELEMENT2 [1]`

`ELEMENT1/ELEMENT2 [@attribut1]`

Übung 2: Daten transformieren und formatieren

Übungsdateien: *kap11\ms.dtd*,
kap11\uebung4.xml

Ergebnisdatei: *kap11\uebung4.xsl*

Erstellen Sie mithilfe der XSL-Transformation die nachfolgende HTML-Ansicht, indem Sie die entsprechenden Daten aus der bereits bestehenden Musiksammlung auslesen.

Interpret Album	Label Jahr	Musikstil	Aufgenommen von
Moby Play	Mute Records Limited 1999	Pop	Heiko Schröder
a-ha Minor earth major sky	WEA Records 2000	Pop	Max Mustermann
Mesh Fragmente	Jarrett Records 1999	Independent	Heiko Schröder
Ludwig van Beethoven Berühmte Klavier-Sonaten	Pilz Compact Disc 1988	Klassik	Max Mustermann
Philip Glass Koyaanisqatsi	Island Records Limited 1983	Klassik	Heiko Schröder

Übung 3: Daten transformieren und formatieren

Übungsdateien: *kap11\kfz.dtd*,
kap11\uebung5.xml

Ergebnisdatei: *kap11\uebung5.xsl*

Lassen Sie sich aus der Fahrzeugverwaltung mithilfe von XSL die Hersteller mit ihren jeweiligen Fahrzeugmodellen auflisten. Stellen Sie die jeweiligen Daten Modell, Hubraum und Leistung übersichtlich in einer HTML-Tabelle dar.

Hersteller	Modell	Hubraum	Leistung
BMW	BMW Z8	4941	294kW
MERCEDES	SLK 32 AMG	3199	260kW
AUDI	1.8 T quattro	1781	165kW

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

12 XSLT-Elemente

In diesem Kapitel erfahren Sie

- ✓ wie Sie zur Mehrfachauswahl von Elementen Schleifen durchführen können
- ✓ wie Elemente auf bestimmte Eigenschaften getestet werden können
- ✓ wie Sie auf verschiedene Rückgabewerte reagieren können

Voraussetzungen

- ✓ Grundlegender Aufbau von XSLT-Befehlen

12.1 Schleifen und Fallunterscheidungen

Bisher konnten Sie mit dem Aktionsteil `<xsl:value-of select="Pattern">` jeweils nur ein Element innerhalb einer Vorlage ansprechen. Um mehrere Elemente mit gleichem Namen zu selektieren, benötigen Sie die Schleifenfunktion. Diese führt einen Aktionsteil so lange aus, bis der angegebene Mustervergleich nicht mehr übereinstimmt. Dies ist beispielsweise nützlich, wenn Sie in der Musiksammlung alle vorhandenen Interpreten auslesen möchten.

12.2 Schleifenbildung

Die Schleife `xsl:for-each` ermöglicht es, einen ganzen Knotensatz innerhalb einer einzigen Regel zu verarbeiten.

```
<xsl:for-each select="Pattern">
  <!-- Aktionen -->
</xsl:for-each>
```

`xsl:for-each` weist den Parser an, nach allen Elementen zu suchen, die mit dem Wert des Attributs `select` übereinstimmen.

Beispiel

Aus einer XML-Struktur sollen die Elemente `ELEMENT1` selektiert und verschiedene Aktionen ausgeführt werden.

```
① <xsl:for-each select="//ELEMENT1">
②   <!-- Aktionsteil -->
③ </xsl:for-each>
```

- ① Diese Anweisung sucht nach dem `ELEMENT1`.
- ② Nach dem Auffinden des gesuchten Elements wird der Aktionsteil ausgeführt.
- ③ Die Schleife wird beendet. Der Prozessor springt zurück zum Anfang der Schleife ①. Es wird erneut geprüft, ob ein weiteres Element `ELEMENT1` vorhanden ist. Ist dies der Fall, wird die Schleife wiederholt ausgeführt. Dies geschieht so lange, bis kein Element mehr gefunden wurde. Die Schleife wird daraufhin verlassen.

Beispiel: *kap12|for-each1.xsl*

Die nachfolgenden Beispiele werden mit den Daten der Musiksammlung realisiert. Es sollen in einer Schleife alle Lieder eines Albums ausgegeben werden.

	<pre> <?xml version="1.0" encoding="iso-8859-1"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:template match="/"> <html> <head><title>Liedübersicht</title></head> <body> ① <xsl:for-each select="//ALBUM"> ② <xsl:value-of select="./INTERPRET ./GRUPPE" /> ③ <xsl:value-of select="./TITEL" /> ④ <xsl:for-each select="./LIED"> ⑤ <xsl:value-of select="." /> ⑥ </xsl:for-each> ⑦ </xsl:for-each> </body> </html> </xsl:template> </xsl:stylesheet> </pre>	
--	---	--

- ① Innerhalb des Templates wird nach dem Element `ALBUM` gesucht, das sich direkt unterhalb des Wurzelements befinden muss. Ist es vorhanden, wird Punkt ② ausgeführt, ansonsten wird zu Punkt ⑦ verzweigt und die Transformation beendet.
- ② Mithilfe des Oder-Operators `|` werden die Inhalte der Elemente `INTERPRET` oder `GRUPPE` ausgegeben.
- ③ Ebenso wird der Inhalt des Elements `TITEL` an das Programm übergeben.
- ④ Eine weitere Schleife wird geöffnet, in der am aktuellen Knoten nach dem Element `LIED` gesucht wird.
- ⑤ Ist ein weiteres Element vorhanden, wird sein Inhalt an das Programm weitergegeben.
- ⑥ Die Schleife zum Selektieren der Elemente `LIED` wird beendet und es wird zum Punkt ④ zurückgekehrt.
- ⑦ Die in Punkt ① geöffnete Schleife wird beendet und wieder zum Anfang verzweigt. Dies geschieht so lange, bis kein Element `ALBUM` mehr gefunden wird.

Beispiel: *kap12|for-each1.xml*

Die festgelegte XSL-Datei wird in die XML-Datei der Musiksammlung eingebunden, um somit die Selektierung der einzelnen Daten zu ermöglichen.

	<pre> <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?> ① <?xml-stylesheet type="text/xsl" href="for-each1.xsl"?> ② <MUSIKSAMMLUNG> <!-- ... --> </MUSIKSAMMLUNG> </pre>	
--	--	--

- ① Nach dem Prolog wird die XSL-Datei über die Processing Instruction `<?xml-stylesheet...?>` eingebunden und auf die Daten der XML-Datei angewendet.
- ② Hier beginnt die Auflistung der einzelnen Elemente.

► Öffnen Sie die Musiksammlung in Ihrem Browser.

Der Interpret, der Name des Albums sowie die jeweiligen Titel werden im Browser wie in der folgenden Abbildung angezeigt. Diese Darstellung ist jedoch keine zufriedenstellende Lösung, weil die Übersicht der Daten verloren geht.

MobyPlayHoneyFind my babyPorcelainWhy does my heart feel so bad?South sideRushingBodyrockNatural BluesMachete7Run onDown slowIf things were perfectEverlovingInsideGuitar flute & stringThe sky is brokenMy weakness-a-haMinor earth major skyMinor earth major skyLittle blackVelvetSummer moved onThe sun never shone that dayTo let you winThe company manThought that it was youI wish I caredBarely hanging onYou'll never get over meI won't forget herMary Ellen makes the moment countMeshFragmenteTrust youMy defenderYou didn't want meI don't think they knowConfinedSomeone to believe inState of mindSo important (original acoustic version)In the light of dayI don't expect to be rightThe purest peopleTrust you (mesh remix)Ludwig van BeethovenBerühmte Klavier-SonatenMondschein SonatePathétiqueWaldstein-SonatePhilip GlassKoyaanisqatsiKoyaanisqatsiVesselCloudscapePruit IgoeThe GridProphecies

Anzeige der selektierten Daten im Browser



Beispiel: *kap12\for-each2.xsl*

Eine mögliche visuelle Gestaltung der Daten kann beispielsweise mithilfe einer Tabelle in HTML realisiert werden. In die entsprechenden Zellen werden dann die XSL-Befehle zum Selektieren der Daten eingefügt.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>Liedübersicht</title>
<style type="text/css">
table { border:3px solid #000; width:400px;}
td { border:1px dotted #000; padding:3px; }
.bg { background-color: #DDD;}
.big { font-size:16pt; font-weight:bold; }
</style>
</head>
<body>
<table>
<tr valign="top" class="bg">
<td><span class="big">Musiksammlung - Auflistung der Lieder</span></td>
</tr>
<xsl:for-each select="//ALBUM">
<tr><td style="color:red; font-weight:bold;">
<xsl:value-of select="./INTERPRET|./GRUPPE" /> -
<xsl:value-of select="./TITEL" /> <br />
</td></tr>
<tr valign="top"> <td>
<xsl:for-each select="./LIED">
<xsl:value-of select="." /> <br />
</xsl:for-each>
</td> </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```


Musiksammlung - Auflistung der Lieder
Moby - Play
Honey
Find my baby
Porcelain
Why does my heart feel so bad?
South side
Rushing
Bodyrock
Natural Blues
Machete
7
Run on
Down slow
If things were perfect
Everloving
Inside
<small>© 1999 Moby & others</small>

Formatierte Anzeige der selektierten Daten im Browser

12.3 Elemente sortieren

Standardmäßig werden die Elemente in der Reihenfolge ihres Auftretens ausgegeben. Mit dem XSLT-Element `xsl:sort` haben Sie die Möglichkeit, die Sortierung der XML-Elemente zu beeinflussen. Dieses XSLT-Element kann als Kind-Element von `for-each` und `apply-templates` verwendet werden.

```
<xsl:sort select="Pattern" [order=" " data-type=" " case-order=" "]>
```

- ✓ `xsl:sort` weist den Parser an, die mit `select` angegebenen Elemente zu sortieren. Die übrigen Attribute sind optional.
 - ✓ Über das Attribut `order` geben Sie die Sortierreihenfolge an. Der Standardwert `ascending` erzwingt eine aufsteigende und der Wert `descending` eine absteigende Sortierung.
 - ✓ Das Attribut `data-type` legt den Datentyp fest. Mit dem Wert `text` (Standardwert) und dem Wert `number` können Sie die Sortierreihenfolge bei Zahlenwerten ändern.
 - ✓ Wenn Sie als Datentyp den Wert `text` festgelegt haben, können Sie mit `case-order` die Einordnung der Buchstaben beeinflussen. Der Wert `upper-first` gibt an, dass auftretende Großbuchstaben vor den entsprechenden Kleinbuchstaben einsortiert werden sollen. Mit dem Standardwert `lower-first` wäre es umgekehrt.
- Erweitern Sie die bisherige Datei *for-each2.xsl* und speichern Sie diese unter dem Namen *sort.xsl*.

Beispiel: *kap12\sort.xsl*

Die Interpreten sollen alphabetisch absteigend sortiert werden. Die Lieder jedes Interpreten sollen hingegen alphabetisch aufsteigend sortiert werden. Die eingefügten Zeilen sind im Beispiel farblich hervorgehoben.

```

<!-- ... -->
<xsl:for-each select="//ALBUM">
①  <xsl:sort select="./INTERPRET|./GRUPPE" order="descending" />
    <tr><td style="color:red; font-weight:bold;">
        <xsl:value-of select="./INTERPRET|./GRUPPE" /> -
        <xsl:value-of select="./TITEL" /> <br />
    </td></tr>

    <tr valign="top"> <td>
        <xsl:for-each select="./LIED">
②  <xsl:sort select="." data-type="text" case-order="upper-first" />
        <xsl:value-of select="." /> <br />
        
```

```

    </xsl:for-each>
  </td></tr>
</xsl:for-each>
<!-- ... -->

```

- ① Innerhalb des Elements `ALBUM` werden die Elemente `INTERPRET` oder `GRUPPE` sortiert. Die alphabetische Sortierung wird über das Attribut `order` umgekehrt.
 - ② Nach dem Selektieren der Elemente `LIED` sollen diese über den Standardwert aufsteigend sortiert werden. Den Datentyp und die veränderte Einordnung von groß- und kleingeschriebenen Buchstaben passen Sie über `data-type` und `case-order` an. Der Wert `upper-first` hat zur Folge, dass große Buchstaben vor den kleinen einsortiert werden, z. B. A, a, B, b, C, c usw.
- Binden Sie die Datei *sort.xsl* in die Datei *for-each2.xml* ein, damit die Sortierung der Daten durchgeführt werden kann.
 - Speichern Sie die Datei unter dem Namen *sort.xml* und laden Sie sie in Ihren Browser.

12.4 Einfache Fallunterscheidung

Mit einer einfachen Fallunterscheidung haben Sie die Möglichkeit, ein Element daraufhin zu testen, ob es selbst oder dessen Attribute bestimmte Bedingungen erfüllen.

Sie können ein Element auf dessen Inhalt überprüfen, indem Sie den Inhalt hinter dem Elementnamen in einfachen Anführungszeichen `' '` angeben.

```

<xsl:if test="Elementname='Inhalt'">
</xsl:if>

```

Soll ein Attribut auf einen bestimmten Wert überprüft werden, geben Sie hinter dem Namen des Elements das entsprechende Attribut und in einfachen Anführungszeichen dessen Wert an. Beachten Sie, dass ein Attribut mithilfe des Zeichens `@` angesprochen wird.

```

<xsl:if test="Elementname/@Attributname='Wert'">
</xsl:if>

```

Element finden

Es soll kontrolliert werden, ob innerhalb des aktuellen Knotens das Element `INTERPRET` mit dem Inhalt `Moby` existiert.

```

① <xsl:if test="INTERPRET='Moby'">
②   <!-- Aktionsteil -->
③ </xsl:if>

```

- ① Mit dieser Abfrage wird im aktuellen Knoten nach dem Interpreten `Moby` gesucht.
- ② Wurde ein Element mit dem gesuchten Interpreten gefunden, wird der Aktionsteil ausgeführt.
- ③ Die Fallunterscheidung wird beendet.

Attributwert finden

In diesem Beispiel wird getestet, ob im aktuellen Knoten ein Element `TITEL` existiert, das zusätzlich ein Attribut `stil` mit dem Wert `Pop` bzw. `p` besitzt.

```

① <xsl:if test="TITEL/@stil='Pop' or TITEL/@stil='p'">
②   <!-- Aktionsteil -->
③ </xsl:if>

```

- ① Mit dieser Abfrage wird im aktuellen Knoten nach einem Album gesucht, das in die Stilrichtung `Pop` eingeordnet wurde. Die Suche können Sie mit dem Schlüsselwort `or` verknüpfen. Es wird im aktuellen Knoten nach dem Element und dessen Attribut gesucht.
- ② Wurde ein Attribut `stil` mit dem Wert `Pop` oder `p` gefunden, wird der Aktionsteil ausgeführt.
- ③ Die Fallunterscheidung wird beendet.

Beispiel: *kap12lif.xsl*

In dem nächsten Beispiel sollen nur die Alben ausgegeben werden, die der Stilrichtung `Pop` zugeordnet werden können.

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
① <xsl:template match="/">
    <html>
    <head>
        <title>Pop-Alben</title>
    </head>
    <body>
        <h2>Musiksammlung - Auflisten der Pop-Alben</h2>
        <div>
②      <xsl:for-each select="//ALBUM">
③        <xsl:sort select="./INTERPRET|./GRUPPE" />
④        <xsl:if test="TITEL/@stil='Pop' or TITEL/@stil='p'">
⑤          <xsl:value-of select="./INTERPRET|./GRUPPE" /> -
⑥          <xsl:value-of select="./TITEL" />
          <br />
⑦        </xsl:if>
⑧      </xsl:for-each>
        </div>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>

```

- ① Die Vorlage wird eingeleitet und das Hauptelement selektiert.
- ② Mit einer Schleife werden alle Alben durchsucht.
- ③ Die Ausgabe soll nach dem Namen des Interpreten bzw. der Gruppe sortiert werden.
- ④ Jedes Element `TITEL` innerhalb des Elements `ALBUM` wird geprüft, ob es das Attribut `stil` mit dem Wert `Pop` oder `p` enthält.
- ⑤ Der Wert des Elements `INTERPRET` wird an das Programm übergeben.
- ⑥ Ebenso wird der Inhalt des Elements `TITEL` zurückgegeben.
- ⑦ Die Fallunterscheidung wird beendet.
- ⑧ Die Schleife zum Durchsuchen der Elemente `ALBUM` wird geschlossen.

Beispiel: *kap12lif.xml*

Nach der Einbindung der XSL-Datei in die bisherige Musiksammlung werden nur die Alben aufgelistet, die dem Musikstil `Pop` zugeordnet worden sind.

Musiksammlung - Auflisten der Pop-Alben

Moby - Play
a-ha - Minor earth major sky

Ausgabe von Daten nach einer einfachen Fallunterscheidung



In XSLT gibt es keine Fallunterscheidung in der Form `if-else`. Hierfür müssen Sie die nachfolgende, komplexe Fallunterscheidung `choose-when-otherwise` einsetzen.

12.5 Komplexe Fallunterscheidung

Die einfache Fallunterscheidung ermöglicht es Ihnen, eine Aktion nur dann auszuführen, wenn das Ergebnis des Tests wahr (true) ist. In komplexeren Fallunterscheidungen haben Sie zusätzlich die Möglichkeit, auch den Fall zu betrachten, dass das Ergebnis der Prüfung falsch (false) ist. Beispielsweise wäre es möglich, die Daten, die einem Mustervergleich entsprechen, in einer anderen Farbe darzustellen.

```
<xsl:choose>
  <xsl:when test="Elementname='Inhalt'">
    <!-- Aktionsteil -->
  </xsl:when>
  <xsl:otherwise>
    <!-- alternativer Aktionsteil -->
  </xsl:otherwise>
</xsl:choose>
```

oder

```
<xsl:choose>
  <xsl:when test="Elementname/@Attributname='Wert'">
    <!-- Aktionsteil -->
  </xsl:when>
  <xsl:otherwise>
    <!-- alternativer Aktionsteil -->
  </xsl:otherwise>
</xsl:choose>
```

Das XSLT-Element `xsl:choose` (deutsch: wählen) leitet eine komplexe Fallunterscheidung ein. Nachfolgend wird über `xsl:when` (deutsch: wenn) die erste Fallunterscheidung eingeleitet. Hierbei wird getestet, ob die angegebene Prüfung des Elements erfolgreich ist. Sollte dies der Fall sein, wird der Aktionsteil ausgeführt. Ansonsten springt der Prozessor zu der Stelle, an welcher der alternative Zweig `xsl:otherwise` (deutsch: ansonsten) beginnt. Es werden die Aktionen ausgeführt, die von `xsl:otherwise` umschlossen werden.

Sie können beliebig viele Fallunterscheidungen definieren. Zu diesem Zweck geben Sie für jeden gewünschten Mustervergleich das XSLT-Element `xsl:when` an.



Der Wert, der geprüft werden soll, wird in einfache Anführungszeichen `' '` gesetzt. Sie können jedoch auch Werte als Zahlen behandeln. In diesem Fall geben Sie die Zahl ohne die einfachen Anführungszeichen an. Um Zahlen zu vergleichen, verwenden Sie die Zeichen `=`, `<` und `>`. Da die beiden letzten Zeichen jedoch als Einleitung und Abschluss eines Elements betrachtet werden, müssen sie als Entities `>` (greater than) und `<` (lower than) angegeben werden.



Es wird immer nur eine Möglichkeit bearbeitet. Wenn ein Mustervergleich zutrifft, wird die Aktion ausgeführt und die Anweisung `xsl:choose` verlassen. Die restlichen Unterscheidungen werden nicht beachtet. Es kann also immer nur ein möglicher Aktionsteil innerhalb einer Fallunterscheidung ausgeführt werden.

Beispiel

In diesem Beispiel wird die grundlegende Anwendung einer komplexen Fallunterscheidung beschrieben. Zum besseren Verständnis werden die Mustervergleiche mit einer möglichen Abfrage zum Namen einer Person und deren Alter durchgeführt.

```
① <xsl:choose>
  ② <xsl:when test="//PERSON/NAME='Meier'">
    <!-- Aktionsteil -->
```

```

③ </xsl:when>
④ <xsl:when test="//PERSON/ALTER &gt; 65">
    <!-- Aktionsteil -->
⑤ </xsl:when>
⑥ <xsl:otherwise>
    <!-- alternativer Aktionsteil -->
⑦ </xsl:otherwise>
⑧ </xsl:choose>

```

- ① Die komplexe Fallunterscheidung wird eingeleitet.
- ② Im ersten Fall wird kontrolliert, ob es in der XML-Struktur eine Person mit dem Namen Meier gibt. Der folgende Aktionsteil wird bei Übereinstimmung ausgeführt.
- ③ Die Ausführung des ersten Aktionsteils wird beendet und die Anweisung `xsl:choose` wird verlassen. Der Prozessor verzweigt zur Stelle ⑥.
- ④ Trifft der erste Fall ② nicht zu, wird kontrolliert, ob das Alter der Person größer als 65 ist. Liefert der Mustervergleich das Ergebnis `true`, wird dieser Aktionsteil ausgeführt.
- ⑤ Der zweite Aktionsteil wird beendet. Der Prozessor springt zur Stelle ⑧.
- ⑥ Trifft kein Mustervergleich zu, wird dieser alternative Aktionsteil ausgeführt.
- ⑦ Die Anweisung `xsl:otherwise` wird beendet.
- ⑧ Die Fallunterscheidung ist abgeschlossen.

Beispiel: *kap12\choose.xsl*

In dem nachfolgenden Beispiel werden die einzelnen Alben der Musiksammlung je nach Bewertung unterschiedlich farbig hinterlegt. Dabei erhalten Alben mit einer Bewertung von 4 und 5 einen hellgrünen und Alben mit einer geringeren Bewertung einen hellroten Hintergrund. Ist keine Bewertung hinterlegt, bleibt der Hintergrund weiß.

```

① <?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html><head>
    <title>Komplexe Fallunterscheidung</title>
    <style type="text/css">
        div { border:1px solid white; padding:3px; }
    </style>
</head>
<body>
<h2>Musiksammlung - Bewertungen auswerten</h2>

    <xsl:for-each select="//ALBUM">
        <xsl:sort select="TITEL/@bewertung" order="descending"/>
        ② <xsl:choose>
            ③ <xsl:when test="TITEL/@bewertung &gt; 3">
                <div style="background-color:#90EE90;">
                    <b><xsl:value-of select="INTERPRET|GRUPPE" /> -
                        <xsl:value-of select="TITEL" /> <br /></b>
                    Bewertung: <xsl:value-of select="TITEL/@bewertung" />
                </div>
            ④ </xsl:when>
            ⑤ <xsl:when test="TITEL/@bewertung &gt;= 0">
                <div style="background-color:#FF6347;">
                    <b><xsl:value-of select="INTERPRET|GRUPPE" /> -
                        <xsl:value-of select="TITEL" /> <br /></b>
                    Bewertung: <xsl:value-of select="TITEL/@bewertung" />
                </div>
            </xsl:when>

```

⑥	<pre> <xsl:otherwise> <div> <xsl:value-of select="INTERPRET GRUPPE" /> - <xsl:value-of select="TITEL" />
 Bewertung: - </div> </xsl:otherwise> </xsl:choose> </xsl:for-each> </body></html> </pre>
⑦	
⑧	<pre> </xsl:template> </xsl:stylesheet> </pre>

- ① Die Vorlage zum Ansprechen von Elementen einer XML-Struktur wird eingeleitet.
- ② Die Fallunterscheidung wird mit `<xsl:choose>` gestartet und ist bis zum schließenden Element ⑦ gültig.
- ③ Mit `xsl:when` wird die erste Fallunterscheidung durchgeführt. Hier wird getestet, ob der Wert des Attributs `bewertung` innerhalb des Elements `TITEL` größer als 3 ist (`>3`). Trifft dies zu, werden die Aktionen zur Formatierung der Ausgabe durchgeführt. In diesem Fall werden die Elemente mit einer hellgrünen Farbe hinterlegt.
- ④ Die erste Fallunterscheidung wird beendet.
- ⑤ Der zweite Mustervergleich testet, ob eine Bewertung `>= 0` vorliegt. Der Hintergrund wird rötlich eingefärbt.
- ⑥ Wenn keiner der vorherigen Mustervergleiche übereinstimmt, wird der alternative Fall `xsl:otherwise` eingeleitet.
- ⑦ Die komplexe Fallunterscheidung `xsl:choose` wird beendet.
- ⑧ Die Vorlage wird beendet.

Musiksammlung - Bewertungen auswerten

Mesh - Fragmente
Bewertung: 5

Moby - Play
Bewertung: 4

Philip Glass - Koyaanisqatsi
Bewertung: 4

a-ha - Minor earth major sky
Bewertung: 3

Ludwig van Beethoven - Berühmte Klavier-Sonaten
Bewertung: nicht vorhanden

Nach Bewertung sortierte Anzeige der Alben

12.6 Übungen

Übung 1: Daten selektieren und ausgeben

Übungsdateien: *kap12\ms.dtd*,
kap12\uebung1.xml

Ergebnisdatei: *kap12\uebung1.xsl*

Stellen Sie die Daten der Musiksammlung in einer Tabelle dar. Der Interpret und der Albumtitel sollen dabei aufsteigend sortiert angezeigt werden. Wählen Sie dazu den Interpreten und den Titel des Albums und listen Sie alle enthaltenen Lieder auf.

Musiksammlung - Aufsteigend sortiert nach Interpret und Titel

a-ha Minor earth major sky	Ludwig van Beethoven Berühmte Klavier-Sonaten	Mesh Fragmente	Moby Play	Philip Glass Koyaanisqatsi
Minor earth major sky Little black Velvet Summer moved on The sun never shone that day To let you win The company man Thought that it was you I wish I cared Barely hanging on You'll never get over me I won't forget her Mary Ellen makes the moment count	Mondschein Sonate Pathétique Waldstein-Sonate	Trust you My defender You didn't want me I don't think they know Confined Someone to believe in State of mind So important (original acoustic version) In the light of day I don't expect to be right The purest people Trust you (mesh remix)	Honey Find my baby Porcelain Why does my heart feel so bad? South side Rushing Bodyrock Natural Blues Machete 7 Run on Down slow If things were perfect Everloving Inside Guitar flute & string The sky is broken My weakness	Koyaanisqatsi Vessel Cloudscape Pruit Igoe The Grid Prophecies

Nach Interpreten sortierte Ausgabe mit den Liedern des jeweiligen Albums

Übung 2: Mit Fallunterscheidungen arbeiten

Übungsdateien: *kap12\ms.dtd*,
kap12\uebung2.xml

Ergebnisdatei: *kap12\uebung2.xsl*

Geben Sie alle Alben mit den Informationen zu den Interpreten aus. Stellen Sie über die Fallunterscheidung `<xsl:choose>` die Stilrichtung der jeweiligen Alben dar.

Übung 3: Mit Schleifen arbeiten

Übungsdateien: *kap12\kfz.dtd*,
kap12\uebung3.xml

Ergebnisdatei: *kap12\uebung3.xsl*

Erstellen Sie über XSL-Schleifen eine formatierte Ausgabe der Daten aus der Fahrzeugverwaltung. Achten Sie darauf, auch die Attributwerte der Elemente als Maßeinheiten auszugeben.

Fahrzeugverwaltung - Technische Daten auflisten

Hersteller	BMW	MERCEDES	AUDI
Modell	BMW Z8	SLK 32 AMG	1.8 T quattro
Leergewicht	1690kg	1495kg	1395kg
Zylinder	8	6	4
Hubraum	4941ccm	3199ccm	1781ccm
Leistung	294kW	260kW	165kW
Höchstgeschwindigkeit	250kmh	250kmh	243kmh
von 0 auf 100 km/h	4,7s	5,2s	6,4s

Formatierte Ausgabe der technischen Daten

13 Links in XML

In diesem Kapitel erfahren Sie

- ✓ wie Sie einfache Links in XML verwenden
- ✓ wie erweiterte Links erstellt und genutzt werden

Voraussetzungen

- ✓ Erstellen von Namensräumen

13.1 Einführung in XLink

HTML bietet die Möglichkeit, zwischen verschiedenen Dokumenten oder innerhalb eines Dokuments zu verzweigen. Die vorhandene Lösung ist mit einigen Einschränkungen verbunden:

- ✓ HTML-Hyperlinks sind statisch, d. h., wenn sie einmal erstellt worden sind, können sie nicht ohne Änderung des Quelltextes geändert werden.
- ✓ Ein Link in HTML kann immer nur ein Ziel besitzen, die Angabe mehrerer Zieladressen ist nicht möglich.
- ✓ HTML-Hyperlinks führen immer vom Link im Quelldokument zum angegebenen Ziel, sie besitzen also nur diese eine Richtung.
- ✓ In HTML können nur die Elemente `<a>` und – mit Einschränkungen `` als Ausgangspunkt eines Links genutzt werden.

Das Ziel von XLink (eXtended Linking Language) ist, eine Sprache ohne die genannten Einschränkungen bereitzustellen. Seit Juni 2010 gibt es eine Empfehlung für XLink, welche im Jahr 2011 nochmals leicht überarbeitet wurde (vgl. <http://www.w3.org/TR/xlink11>).

XLink gibt Ihnen die Möglichkeit, Verweise dynamisch so zu verändern, dass sie Verbindungen zwischen verschiedenen XML-Dokumenten oder Elementen herstellen.



Die nachfolgend aufgeführten Beispiele können mit den zum Zeitpunkt der Bucherstellung (Oktober 2014) aktuellen Browsern nur sehr eingeschränkt angezeigt werden, da die XLink-Funktionen kaum in den Browsern implementiert sind. Lediglich der Firefox unterstützt XLink rudimentär. Die gezeigten Beispiele beziehen sich auf die vorgegebene Spezifikation des W3-Konsortiums und werden vermutlich mit den Browsern der nächsten Generation genutzt werden können.



Mit XPointer gibt es eine XPath-Erweiterung, die anstelle auf ein ganzes Dokument genau wie bei Ankern in HTML auf einzelne Dokumentteile verweist. Unter dem Link http://www.w3.org/TR/#tr_XPointer finden Sie die Empfehlungen des W3C aus dem Jahr 2003 zu diesem Thema. Gegenwärtig wird XPointer (mit Ausnahme des Amaya-Browsers (vgl. <http://www.w3.org/Amaya/>, einer Referenzanwendung des W3C) von keinem Browser unterstützt, jedoch von anderen XML-Sprachen verwendet.

13.2 XLink

Verweise innerhalb eines Dokuments werden interne Verknüpfungen genannt. Ein Beispiel in HTML ist die Sprungmarke, die innerhalb einer Webseite zu einer bestimmten Stelle führt. In XML werden die internen Verweise mit XPath realisiert.

Eine weitere Art von Links sind externe Verknüpfungen, die Grafiken einbinden oder auf andere Webseiten verweisen. In XML werden die externen Verweise mittels XLink realisiert. XLink ist ein Teil der eXtended Linking Language und wird verwendet, um Verknüpfungen zwischen verschiedenen Ressourcen zu erstellen und zu beschreiben. Diese Verknüpfungen sind mit den Hyperlinks in HTML vergleichbar, besitzen jedoch erweiterte Funktionen.

Namensraum festlegen

Zum Anlegen eines XLinks werden Namensräume für die entsprechenden Elemente angelegt. Zur Namensraum-Definition wird standardmäßig die URI <http://www.w3.org/1999/xlink> benutzt:

Es gibt zwei Möglichkeiten, ein XLink-Element zu erstellen. Entweder Sie definieren den Namensraum direkt am Element

```
<ELEMENT xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="datei.xml">
  Inhalt des Elements
</ELEMENT>
```

oder Sie legen einen Bereich fest, in dem der Namensraum verwendet werden soll.

```
<xlink:simple xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:href="datei.xml">
  <ELEMENT>Inhalt des Elements</ELEMENT>
</xlink:simple>
```

Attribute

Zur speziellen Definition eines XLinks stehen Ihnen verschiedene Attribute zur Verfügung.

Attribut	Bedeutung
type	Bezeichnet beliebige XML-Elemente, die wie XLink-Elemente behandelt werden. Dieses Attribut kann die Werte <code>simple</code> , <code>extended</code> , <code>locator</code> , <code>arc</code> , <code>resource</code> oder <code>title</code> annehmen.
href	Stellt eine Adresse oder eine externe Ressource dar. Die Werte müssen URI-Referenzen (relative oder absolute URLs) sein. Besitzt das Attribut <code>type</code> den Wert <code>locator</code> , muss das Attribut <code>href</code> angegeben werden.
role	Das Attribut bezeichnet die Quelle, von der aus verlinkt wird.
arcrole	Hiermit legen Sie eine Bemerkung zu einem XLink-Element an.
title	Dieses Attribut beschreibt für den Nutzer die Funktion eines Elements.
show	Das Attribut legt die Art der Anzeige des Links fest. Der Attributwert <code>embed</code> stellt im Dokument die eingebundene Zielressource, z. B. ein Bild, sofort dar. Der Wert <code>new</code> zeigt das Ziel (z. B. ein Bild) in einem neuen Fenster an. Der Wert <code>replace</code> ersetzt die aktuell angezeigte Ressource mit der Zielressource des Links.
actuate	Legt die Art der Aktivierung des Links fest. Gültige Werte sind: <code>onLoad</code> , <code>onRequest</code> , <code>other</code> und <code>none</code> .
label, from, to	Hiermit legen Sie eine eindeutige Bezeichnung für ein XLink-Element fest, auf das Sie über die Angabe von <code>from</code> und <code>to</code> direkt verweisen können.

Ein Element ist dann ein XLink-Element, wenn es ...

- ✓ mindestens ein Attribut `type` mit einem gültigen Wert besitzt,
- ✓ die für den Attributtyp erforderlichen weiteren Attribute enthält.

Der Elementtyp und seine Attribute

Die nachfolgende Tabelle stellt die optionalen Angaben (o) und Pflichtangaben (x) entsprechend dem gewählten Elementtyp in der Übersicht dar (Quelle: <http://www.w3.org/TR/xlink>).

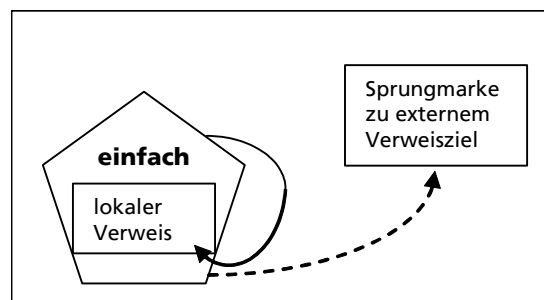
	simple	extended	locator	arc	resource	title
type	x	x	x	x	x	x
href	o	-	x	-	-	-
role	o	o	o	-	o	-
arcrole	o	-	-	o	-	-
title	o	o	o	o	o	-
show	o	-	-	o	-	-
actuate	o	-	-	o	-	-
label	-	-	o	-	o	-
from	-	-	-	o	-	-
to	-	-	-	o	-	-

Die Angaben besagen:

x	Pflichtangabe: Sie müssen das Attribut angeben.
o	Optional: Ihnen ist es freigestellt, ob Sie dieses Attribut angeben.
-	Keine Angabe: Dieses Attribut kann in Verbindung mit dem Elementtyp nicht genutzt werden.

13.3 Einfache Links

Ein einfacher Link (engl. simple link) besteht aus dem lokalen oder dem externen Verweisziel. XLink definiert kein neues Element innerhalb eines XML-Dokuments, sondern legt globale Attribute fest. Dies bedeutet, dass jedes XML-Element, das diese globalen Attribute verwendet, auch ein XLink-Element darstellen kann.



Die Document Type Definition eines einfachen Links in XLink lautet (Quelle: <http://www.w3.org/TR/xlink>):

```
<!ELEMENT ElementName ANY>
<!ATTLIST ElementName
  xlink:type      (simple)          #FIXED "simple"
  xlink:href      CDATA             #IMPLIED
  xlink:role      CDATA             #IMPLIED
  xlink:arcrole   CDATA             #IMPLIED
  xlink:title     CDATA             #IMPLIED
  xlink:show      (new|replace|embed|other|none) #IMPLIED
  xlink:actuate   (onLoad|onRequest|other|none)  #IMPLIED>
```

Beispiel: kap13\link_a.xml

Im Folgenden wird das HTML-Element a durch einen XLink nachgebildet.

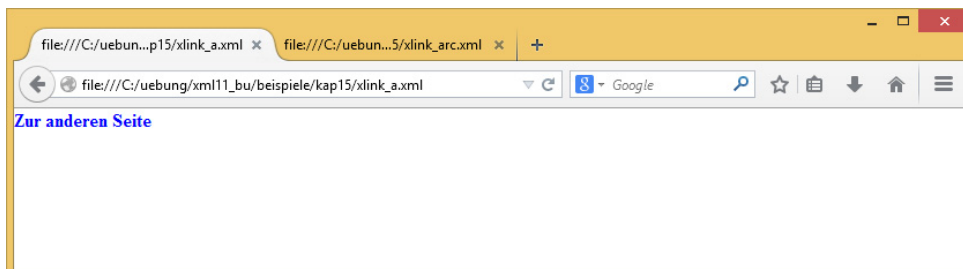
```

① <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
② <?xml-stylesheet type="text/css" href="link.css"?>
③ <xlink:simple xmlns:xlink="http://www.w3.org/1999/xlink">
④   <a xlink:type="simple"
⑤     xlink:actuate="onRequest"
⑥     xlink:show="replace"
⑦     xlink:href="xlink_arc.xml">
      Zur anderen Seite
⑧   </a>
⑨ </xlink:simple>

```

- ① Zur Gestaltung der Seite wird eine einfache CSS-Datei eingebunden.
- ② Für das Element a wird ein einfacher Link definiert.
- ③ Dieser soll erst auf Anfrage (onRequest) ausgelöst werden.
- ④ Nach dem Auslösen soll der derzeitige Inhalt durch den neuen Inhalt ersetzt werden.
- ⑤ Der Link verweist auf die Datei *xlink_arc.xml*.
- ⑥ Der Link wird mit dem angegebenen Text dargestellt.
- ⑦ Die Definition des Links wird beendet.

Von den bekannten Browsern unter Windows (Firefox, Internet Explorer, Google, Opera) unterstützt aktuell nur der Firefox die Umsetzung von XLink-Befehlen. Dies gilt allerdings auch nur für einfache Links wie in dem Beispiel. Des Weiteren funktioniert der Link nur beim gleichzeitigen Betätigen der **Strg**- oder der **⇧**-Taste während des Ausführens des Mausklicks. Dabei bewirkt – unabhängig vom Wert des Attributs `xlink:show` – die Taste **⇧** das Öffnen der Zielseite in einem neuen Browserfenster und die Taste **Strg** das Öffnen der Zielseite in einem zusätzlichen (verdeckten) Register.



Darstellung im Firefox nach gleichzeitigem Betätigen von **Strg** und Klick auf den Link

Beispiel: kap13\link_img.xml

Ein Link, der bereits beim Laden der Seite in das Dokument eingebunden werden soll, lässt sich ebenfalls realisieren. Daher ist es auch möglich, mit XLink das HTML-Element `img` zu realisieren, denn Bilder sind Objekte, die bereits beim Laden der Seite eingebunden werden.

```

① <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
② <xlink:simple xmlns:xlink="http://www.w3.org/1999/xlink">
③   <img xlink:type="simple"
④     xlink:actuate="onLoad"
⑤     xlink:show="embed"
⑥     xlink:href="bild.gif">
      Ein eingebettetes Bild
⑦   </img>
⑧ </xlink:simple>

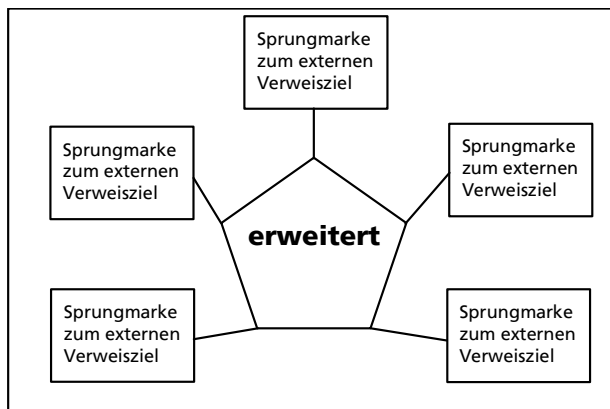
```

- ① Das Element `img` wird als einfacher Link definiert.
- ② Dieser soll bereits beim Laden des Dokuments (`onLoad`) ausgelöst werden.
- ③ Das Bild soll in das Dokument eingebettet werden (`embed`).
- ④ Der Link verweist auf die angegebene Datei `bild.gif`.
- ⑤ Die Grafik erhält die angegebene Bildunterschrift.
- ⑥ Die Festlegung des eingebetteten Links wird beendet.

13.4 Erweiterte und multidirektionale Links

Erweiterte Links können mehr als zwei Verweisziele haben und somit komplizierte Verbindungen zwischen beliebig vielen Ressourcen herstellen. Keiner der Endpunkte eines erweiterten Links muss dabei ein Teil des Dokuments sein, in dem der Link definiert wird.

Die Linien in der Abbildung stellen die Verbindungen eines erweiterten Links zu verschiedenen externen Verweiszielen (externe Ressourcen) dar. Diese Beziehungen besitzen jedoch keine Richtung und werden durch sogenannte traversal rules (deutsch: Verbindungsregeln) bestimmt. Ohne diese Verbindungen hätten die Ressourcen keinerlei Beziehung untereinander, sie könnten beispielsweise fünf getrennte Dokumente sein.



Auch ein Element vom Typ `extended` ist in der Lage, einen XLink zu definieren. Alle nachfolgenden XLink-Elemente (`locator`, `arc` usw.) geben lediglich weitere Informationen zum Link an und werden innerhalb eines `extended`-Elements notiert. Treten solche Elemente nicht innerhalb eines `extended`-Elements auf, werden sie ignoriert. (Quelle: <http://www.w3.org/TR/xlink>)

```
<!ELEMENT ElementName ((title|resource|locator|arc)*)>
<!-- ATTLIST ElementName
  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
  xlink:type (extended) #FIXED "extended"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED-->
```

Die Definition des Namensraumes, in dem die weiteren XLink-Elemente eingefügt werden, lautet wie folgt:

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink">
  <!-- weitere Kindknoten -->
</xlink:extended>
```

Der erweiterte Link kann, wie bereits erwähnt, weitere Elemente enthalten.

Elementtyp	Erläuterung
<code>locator</code>	Der Elementtyp adressiert das externe Verweisziel, das innerhalb des erweiterten Links verwendet wird.
<code>arc</code>	<code>arc</code> bestimmt die Art und Weise, wie ein XLink benutzt wird. Die Angabe besteht aus einer Quelle und dem Ziel.
<code>title</code>	Dieser Typ ermöglicht es, für den Menschen lesbare Informationen zu einem Link zur Verfügung zu stellen.
<code>resource</code>	Dieses Element liefert die lokale Ressource, die am Link beteiligt ist.

Elementtyp locator

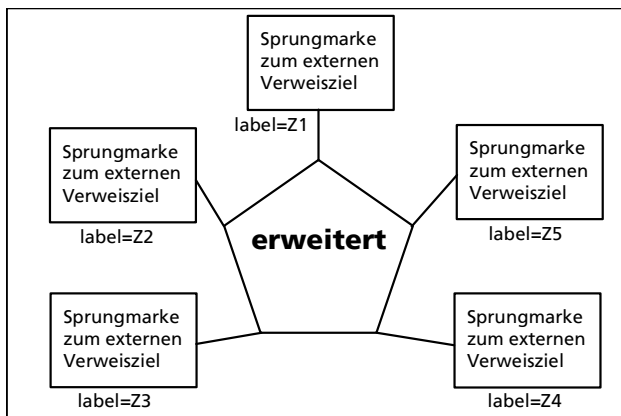
Innerhalb des `extended-Links` wird das `locator-Element` festgelegt. Es beinhaltet über das `href`-Attribut die notwendige Information für die einzubindende Ressource. Über das Attribut `label` wird diesem Element ein Bezeichner zugewiesen. Durch die Mehrfachangabe dieses Elements innerhalb eines `extended-Elements` werden Mehrfachverweise erstellt (Quelle: <http://www.w3.org/TR/xlink>).

```
<!ELEMENT ElementName (title*)>
<!--ATTLIST ElementName
  xlink:type (locator) #FIXED "locator"
  xlink:href CDATA #REQUIRED
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:label NMTOKEN #IMPLIED-->
```

Beispiel: *kap13|xlink_locator.xml*

Innerhalb des erweiterten XLinks wird einer Sprungmarke `href` eine Bezeichnung `label` zugewiesen.

In dem nachfolgenden Beispiel sind den Links keine Werte zugewiesen worden. Die drei Leerzeichen stehen für eine beliebige URI.

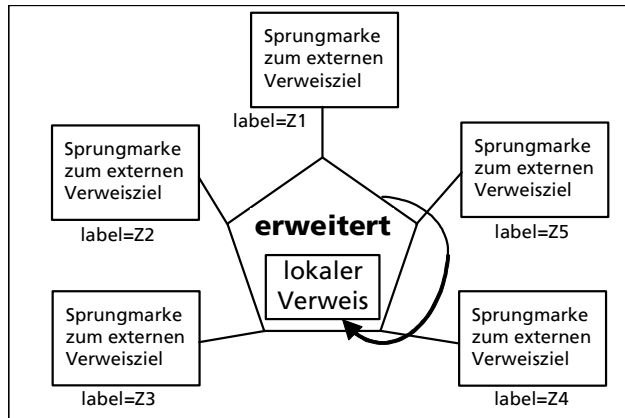


```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink">
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z1" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z2" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z3" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z4" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z5" />
</xlink:extended>
```

Elementtyp resource

Dieses Element legt die lokale Ressource fest, die auf ein spezielles XML-Element verweist und die innerhalb des erweiterten Links als Unterelement erscheinen soll. Ein vollständiges Unterelement mit seinem gesamten Inhalt wird als lokale Ressource bezeichnet. Im Grunde kann dies jede adressierbare Informationseinheit sein, z. B. eine XML-Datei, eine Grafik, ein Wert oder Ähnliches. Die W3-Spezifikation des `resource-Elements` lautet (Quelle: <http://www.w3.org/TR/xlink>):

```
<!ELEMENT ElementName ANY>
<!--ATTLIST ElementName
  xlink:type (resource) #FIXED "resource"
  xlink:role CDATA #IMPLIED
  xlink:title CDATA #IMPLIED
  xlink:label NMTOKEN #IMPLIED-->
```

Beispiel: *kap13\xlink_resource.xml*

Die Abbildung stellt einen erweiterten Link mit fünf externen und einem lokalen Verweisziel dar. Der lokale Verweis kann beispielsweise einen lokal gespeicherten Wert enthalten.

```

① <?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
  <xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink">
    <wo xlink:type="locator" xlink:href="..." xlink:label="Z1" />
    <wo xlink:type="locator" xlink:href="..." xlink:label="Z2" />
    <wo xlink:type="locator" xlink:href="..." xlink:label="Z3" />
    <wo xlink:type="locator" xlink:href="..." xlink:label="Z4" />
    <wo xlink:type="locator" xlink:href="..." xlink:label="Z5" />
  ② <was xlink:type="resource" xlink:label="wert">lokale Ressource</was>
  ③ <was xlink:type="resource" xlink:label="pic">/images/bild.jpg</was>
  </xlink:extended>

```

- ① Innerhalb des erweiterten XLinks werden den Sprungmarken href die Bezeichnungen label zugewiesen.
- ② Als lokale Ressource wird der Verweis hinzugefügt, der den Bezeichner wert mit dem Text lokale Ressource enthält.
- ③ Der zweite Verweis pic besitzt den Inhalt /images/bild.jpg, der beispielsweise zum Anzeigen eines Bildes genutzt werden kann.

Elementtyp arc

Das Element arc (deutsch: Bogen) definiert die Richtung und das Verhalten eines Links innerhalb einer Ressource. Dabei werden die Richtungen durch die Angaben der mit locator definierten Bezeichner festgelegt (Quelle: <http://www.w3.org/TR/xlink>).

```

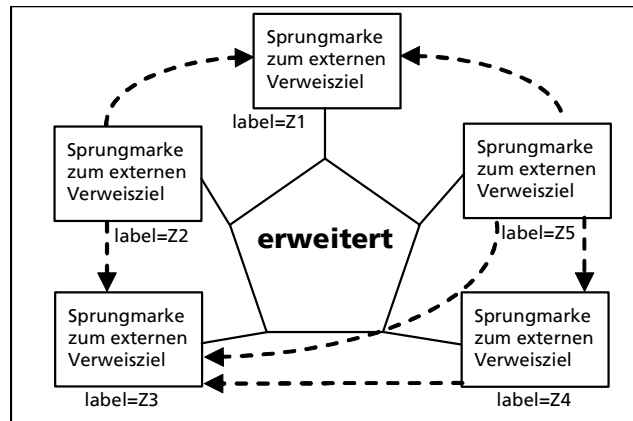
<!ELEMENT ElementName (title*)>
<!ATTLIST ElementName
  xlink:type      (arc)                #FIXED "arc"
  xlink:arcrole   CDATA                 #IMPLIED
  xlink:title     CDATA                 #IMPLIED
  xlink:show      (new|replace|embed|other|none) #IMPLIED
  xlink:actuate   (onLoad|onRequest|other|none)  #IMPLIED
  xlink:from      NMTOKEN               #IMPLIED
  xlink:to        NMTOKEN               #IMPLIED>

```

Beispiel: *kap13\link_arc.xml*

Mit dem XLink-Element `arc`, welches ebenfalls innerhalb des `extended`-Elements angegeben werden muss, definieren Sie die Verbindungsregel, mit der von einer zur anderen Ressource verwiesen wird.

Werden mehrere Verbindungsregeln angegeben, welche die gleichen Ressourcen beschreiben, bei denen jedoch der Quell- und der Ziellink vertauscht sind, wird dieser Link als multidirektional bezeichnet.



```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink">
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z1" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z2" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z3" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z4" />
  <wo xlink:type="locator" xlink:href="..." xlink:label="Z5" />
  <was xlink:type="resource" xlink:label="wert">lokale Ressource</was>
  <was xlink:type="resource" xlink:label="pic">/images/bild.jpg</was>
  ① <wie xlink:type="arc" xlink:from="Z2" xlink:to="Z3" />
  <wie xlink:type="arc" xlink:from="Z2" xlink:to="Z1" />
  ② <wie xlink:type="arc" xlink:from="Z5" xlink:to="Z3" />
  <wie xlink:type="arc" xlink:from="Z5" xlink:to="Z1" />
  <wie xlink:type="arc" xlink:from="Z5" xlink:to="Z4" />
  <wie xlink:type="arc" xlink:from="Z4" xlink:to="Z3" />
</xlink:extended>
```

- ① Mit dem `arc`-Element wird festgelegt, dass die bereits mit `locator` angelegten Sprungmarken mit der Bezeichnung `Z2` und `Z3` verbunden werden.
- ② Hier wird der Link von `Z5` nach `Z3` festgelegt.

Elementtyp `title`

Die XLink-Elemente `extended`, `locator` und `arc` können ein Attribut `title` besitzen. Zusätzlich können diese Elemente auch mehrere Elemente des Elementtyps `title` enthalten. Dies ist beispielsweise nützlich, wenn für den Menschen lesbare Hinweise zu einem Link notwendig sind. Ein wichtiger Aspekt ist hierbei die Verwendung von mehrsprachigen Hinweisen, da verschiedensprachige Texte eingefügt werden können. Dieses Element besitzt in XLink keine weiteren Attribute.

```
<!ELEMENT ElementName ANY>
<ATTLIST ElementName
  xlink:type (title) #FIXED "title"
  xlink:lang CDATA #IMPLIED>
```

13.5 XBase

Eine Zielsetzung bei der Definition von XLink war, die in HTML enthaltenen Verknüpfungsmöglichkeiten vollständig abzubilden. Mit der Einführung von HTML 4.01 ergab sich daraus die Anforderung, das in dieser Version neue Element `<base>` zur Spezifizierung einer Basisadresse von URIs entsprechend in XLink nachzubilden.

Im Ergebnis wurde mit XBase eine Spezifikation geschaffen, die diese Anforderung umsetzt. Sie stellt das Attribut `xml:base` zur Verfügung. Der dem Attribut zugewiesene Wert wird bei der Dokumentverarbeitung als Basisadresse interpretiert. Die gesamte als Linkziel verwendete URI ergibt sich damit aus dieser Basisadresse in Verbindung mit dem im Attribut `xlink:href` angegebenen Wert.

In Anlehnung an das Beispiel *kap13xlink_a.xml* würde aus

```
<a xml:base="http://www.herdtd.com/files"
  xlink:type="simple"
  xlink:href="xlink_arc.xml">
  Zur anderen Seite
</a>
```

die folgende URI entstehen:

http://www.herdtd.com/files/xlink_arc.xml

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

14 XQuery

In diesem Kapitel erfahren Sie

- ✓ wie Sie Daten mit XQuery selektieren
- ✓ wie Sie mit XPath Ausdrücke in XQuery nutzen
- ✓ was FLWOR Ausdrücke sind und wie sie diese anwenden
- ✓ welche erweiterten Möglichkeiten XQuery bietet

Voraussetzungen

- ✓ Kenntnisse im Aufbau eines XML-Dokuments
- ✓ XPath Kenntnisse

14.1 XQuery-Grundlagen

Entwicklung

Die Mengen an in XML gespeicherten Daten werden immer größer. Wesentliche Ursachen dafür sind z.B.:

- ✓ die sich aus dem XML-Format ergebenden Möglichkeiten zum Datenaustausch zwischen Unternehmen,
- ✓ die effektive Bereitstellung von Daten zur Darstellung auf Webseiten und deren Verarbeitung in Webshops,
- ✓ die Erweiterung von relationalen Datenbanksystemen um Möglichkeiten zur Speicherung von Daten im XML-Format,
- ✓ die Umstellung von vielen Anwenderprogrammen, zum Beispiel der Office Programme von Microsoft, auf XML als Datenformat für erstellte Dokumente.

Mit der zunehmenden Datenmenge entstand der Bedarf nach einer effektiven und allgemeinverwendbaren Abfragesprache analog zu SQL (**S**tructured **Q**uery **L**anguage) für die Auswertung von relationalen Datenbanken. Im Jahr 2007 veröffentlichte das W3C die Empfehlung XQuery 1.0, um dieser Anforderung nachzukommen. Während die Version 2.0 aus dem Jahr 2010 sich im Wesentlichen auf Fehlerkorrekturen beschränkte, liegt mit der aktuellen Version 3.0 vom April 2014 (<http://www.w3.org/TR/xquery-30/>) eine Version vor, die einige Spracherweiterungen enthält. Dabei handelt es sich unter anderem um Erweiterungen bei den sogenannten FLWOR-Ausdrücken (vgl. Abschnitt 14.3) sowie um programmtechnische Erweiterungen wie beispielsweise die Unterstützung dynamischer Funktionsaufrufe, die Verwendung privater Funktionen und die Einführung von in vielen Programmiersprachen typischen `try/catch` Ausdrücken.

Datenmodell und Syntax

Bei XQuery handelt es sich um eine Erweiterung von XPath 2.0. Genau wie XPath ist auch XQuery keine XML-Anwendung, sondern besitzt eine eigene Syntax.



Mit der Empfehlung XQueryX des W3C (<http://www.w3.org/TR/xqueryx/>) existiert eine Variante zur Formulierung von XQuery-Abfragen in Form eines XML-Dokuments.

Beide Abfragetechniken basieren auf demselben Datenmodell (vgl. Abschnitt 10.3). XPath 2.0 ist eine Teilmenge von XQuery 1.0. Jeder gültige XPath 2.0 Ausdruck ist zugleich ein gültiger XQuery 1.0 Ausdruck. Alle Möglichkeiten von XPath 2.0 zur Datenselektion können in XQuery genutzt werden.

Syntaktisch handelt es sich bei XQuery um eine funktionale Sprache, deren einzelne Ausdrücke ausgewertet werden. Die Ausdrücke können geschachtelt sein, sodass ein Ausdruck weitere Ausdrücke beinhalten kann.

Für Schlüsselwörter wird die Kleinschreibung genutzt, es existieren jedoch (mit Ausnahme einiger Funktionsnamen) keine reservierten Schlüsselwörter.

XQuery-Abfragen bestehen aus Modulen. Es wird zwischen zwei Modultypen unterschieden: dem **Hauptmodul** sowie den **Bibliotheksmodulen**.

Das Hauptmodul ist in zwei Bereiche eingeteilt, den **Prolog** für Einstellungen und Deklarationen und den **Query-Body**, welcher die eigentliche Abfrage als Ausdruck enthält.

Hauptmodul	
Prolog (Deklarationen und Importklauseln werden jeweils mit einem Semikolon <code>;</code> abgeschlossen)	Teil 1: Einstellungen für ✓ Setter für den XQuery-Prozessor (Steuerung der Abfrage) ✓ Namensraumdeklarationen ✓ Importklauseln (für Deklarationen und Schemas)
	Teil 2: Deklaration von ✓ Variablen, ✓ Funktionen und ✓ Optionen
Query-Body	Abfrageausdruck

Bibliotheksmodule bestehen aus einer **Moduldeklaration** und dem **Prolog**. Alle Module können eine einleitende Versionsdeklaration enthalten. In dieser können Sie eine Deklaration der Codierung vornehmen:

```
xquery version "1.0" encoding "utf-8";
```

Ob und wie diese berücksichtigt wird, ist jedoch von der Implementierung des jeweiligen XQuery-Prozessors abhängig.

XQuery-Ausdrücke können **Variablen** enthalten. Diesen kann – entgegen dem eigentlichen Wortsinn und im Unterschied zu anderen Programmiersprachen – in einem XQuery-Ausdruck nur **einmal** ein Wert zugewiesen werden. Eine nachträgliche Änderung ist nicht möglich. Anwendung finden Variablen deshalb in erster Linie als Referenz auf einen ihnen zugewiesenen Code. Sie dienen somit der besseren Lesbarkeit und Übersichtlichkeit der Ausdrücke.

Abfrageausdrücke

Abfrageausdrücke beziehen sich auf ein Datenmodell, welches zum Zeitpunkt der Abfrage bekannt sein muss. Mögliche Abfrageziele sind zum einen lokale oder über eine URL zugängliche XML-Dokumente. Die Zuordnung der Abfrageziele erfolgt dabei über die Funktionen `doc()` bzw. `collection()`. Zum anderen ist XQuery vom Ansatz her so konzipiert, dass Abfrageausdrücke auch zur Abfrage anderer Datenquellen, wie einer die Speicherung von XML-Daten unterstützenden Datenbank oder den Inhalt einer Webseite, genutzt werden können.

Der Abfrageausdruck besteht aus einer beliebigen Anzahl von Einzelausdrücken. Jeder Einzelausdruck wird mit einem Komma abgeschlossen. Die Abfolge der Einzelausdrücke wird auch als **Sequenz** bezeichnet. Sequenzen werden in Klammern eingeschlossen. Einzelausdrücke besitzen verschiedene Formen. Dabei können im Unterschied zu den reinen Abfragemöglichkeiten in XPath 2.0 mit einem XQuery-Ausdruck auch neue Elemente einer Datenstruktur erzeugt werden:

Ausdruck	Beispiele	Hinweis
Literale	"Produkt von 2 * 2", "aktuelle Zeit"	
Arithmetische Ausdrücke	2*2	
Funktionen	true(), count (...), current-time()	
Konstrukoren	, element a { attribute x { 1 } }	
XPath 2.0-Ausdruck	doc("musiksammlung.xml")//GRUPPE	vgl. Abschnitt 14.2
FLWOR-Ausdruck	<pre> let \$doc := doc("musiksammlung.xml") for \$x in \$doc//ALBUM where \$x/JAHR < 1989 order by \$x/INTERPRET return <ALBUM> { \$x/INTERPRET } </ALBUM> </pre>	vgl. Abschnitt 14.3



Implementierungen

Die Auswertung der Ausdrücke erfolgt durch den XQuery-Prozessor. Mit der zunehmenden Verbreitung von XQuery wächst auch die Anzahl von verfügbaren Implementierungen:

- ✓ Viele professionelle XML-Werkzeuge, wie beispielsweise von Altavo (<http://www.altavo.com>) und DataDirect (<http://www.xquery.com>) besitzen mittlerweile eine XQuery-Komponente.
- ✓ Für die Integration in Java-Anwendungen existiert die XQuery API for Java (XQJ, siehe <https://jcp.org/en/jsr/detail?id=225>), welche analog zur JDBC API für die Auswertung relationaler Datenbanken für XML-Dokumentbestände genutzt werden kann.
- ✓ Das .Net Framework von Microsoft enthält Klassen in dem speziellen Namensraum System.Xml.XPath zur Arbeit mit XPath und XQuery-Ausdrücken.
- ✓ Der Datenbankstandard SQL enthält eine Erweiterung mit dem Datentyp XML. Dieser wird in zunehmenden Maße von den einzelnen Datenbanksystemen unterstützt, beispielsweise von IBM DB2 9.5 und von PostgreSQL (vgl. HERDT-Buch *SQL - Grundlagen und Datenbankdesign*, Anhang A „SQL/XML“).

Auf Grund der Leistungsfähigkeit von XQuery, beispielsweise der Datenauswertung unterschiedlicher Quellen (vgl. Abschnitt 17.4) und der Zusammenführung der ermittelten Daten, ist zukünftig eine weiter zunehmende Unterstützung und Bedeutung von XQuery zu erwarten. Dies ergibt sich auch aus den zunehmenden Anforderungen der Verwaltung großer Datenmengen (Stichwort: Big Data). Die Datenmengen liegen oft im XML-Format vor.



Auf der Webseite des W3C zum Thema (<http://www.w3.org/XML/Query/>) finden Sie Hinweise auf verfügbare Implementierungen und Testumgebungen.

14.2 XPath basierte XQuery-Abfragen

Zur Datenselektion können Sie in XQuery XPath 2.0 Ausdrücke verwenden. In Verbindung mit den in XQuery vorhandenen vordefinierten Funktionen ermöglicht dies kurze Abfragen.

Beispiel *kap14/xquery1_xsl.xql*

Im Beispiel werden mit einem XPath-Ausdruck alle Gruppen in der Musiksammlung ermittelt und als Unter-elemente in den neuen Wurzelknoten `GRUPPEN` eingefügt. Das Ergebnis enthält eine Sequenz ausgewählter Elemente des Ausgangsdokuments.

```

① xquery version "1.0";
② <GRUPPEN>
③ {
④ doc("musiksammlung.xml") //GRUPPE
  }
</GRUPPEN>

```

- ① Die optionale Versionsdeklaration des Dokuments.
- ② Die selektierten Elemente sollen in ein XML-Fragment mit dem Wurzelknoten `Gruppen` eingefügt werden. Dieses wird hier definiert. Die Definition stellt einen gültigen XQuery-Ausdruck dar und wird in die Ausgabe so wie definiert übernommen.
- ③ Die geschweiften Klammern schließen den Ausdruck zur Selektion der Daten ein.
- ④ Das auszuwertende XML-Dokument wird über die XQuery-Funktion `doc()` geöffnet. Im direkten Anschluss auf den Funktionsaufruf folgt ein XPath-Ausdruck, der die zu selektierenden Elemente – im Beispiel den Inhalt des Elements `GRUPPE` aller Alben, die das Element besitzen – bestimmt. Die beiden `//` vor dem zu selektierenden Element geben an, dass sich dieses auf einer, vom Wurzelknoten aus gesehen, beliebigen Hierarchieebene befinden kann.

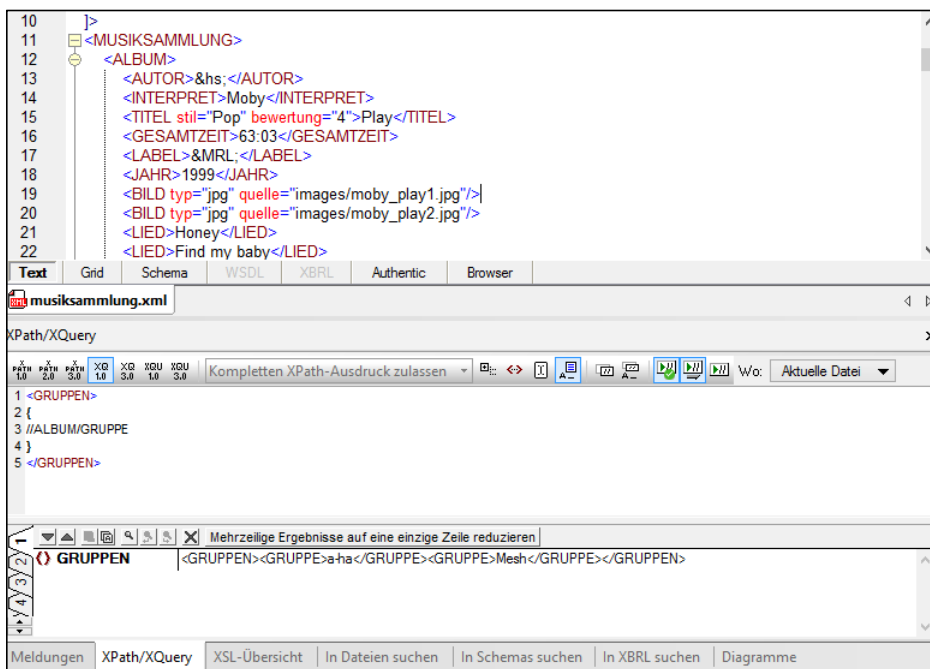
```

<?xml version="1.0"?>
- <GRUPPEN>
  <GRUPPE>a-ha</GRUPPE>
  <GRUPPE>Mesh</GRUPPE>
</GRUPPEN>

```

Ergebnis in der Browserdarstellung

Einige XML-Umgebungen unterstützen die direkte Auswertung von XPath- und XQuery-Ausdrücken. Die Abbildung zeigt die direkte Auswertung des im Beispiel verwendeten XQuery-Ausdrucks in Altova XMLSpy.



Auswertung eines XML-Ausdrucks auf die aktuelle Datei in Altova XMLSpy

14.3 Abfragen mit FLWOR-Ausdrücken

Bestandteile von FLWOR Ausdrücken

Ohne gute Kenntnisse in XPath 2.0 sind die darauf basierenden Abfragen bei zunehmender Komplexität der Selektion nicht immer leicht lesbar und sofort verständlich. Alternativ bietet XQuery mit den sogenannten FLWOR-Ausdrücken eine zweite Möglichkeit zur Datenselektion.



Die Aussprache von FLWOR entspricht dem englischen Flower (Blume).

Der Name der Ausdrücke FLWOR resultiert aus den Anfangsbuchstaben der **Klauseln**, aus denen sich ein typischer nach dieser Syntax gebildeter Ausdruck zusammensetzt.

F	for	dient der Ausführung einer Iteration auf eine Sequenz
L	let	definiert eine in dem Ausdruck gültige Variable
W	where	formuliert die Bedingung der Knotenauswahl
O	order by	legt die Sortierung der Ergebnisse fest
R	return	bestimmt das vom Ausdruck gelieferte Ergebnis

Die FLWOR-Syntax hat starke Ähnlichkeit mit der SQL-Syntax (Structured Query Language) zur Abfrage von relationalen Datenbanken. Für die einzelnen Klauseln gelten folgende Regeln:

Klausel	Regeln	Häufigkeit im Ausdruck
for	✓ Operation wird auf alle Elemente der Sequenz ausgeführt	✓ mindestens eine for oder eine let Klausel pro Ausdruck
let	✓ Variablennamen müssen immer mit dem Zeichen \$ beginnen ✓ Wertzuweisung erfolgt über den Operator :=	
where	✓ steht nach allen for oder let Klauseln ✓ die Verknüpfung mehrerer Bedingungen über die logischen Operatoren and bzw. or ✓ zur Strukturierung der Bedingungen können Klammern verwendet werden	✓ optional ✓ maximal eine Klausel pro Ausdruck
order by	✓ die Sortierreihenfolge kann über die Schlüsselwörter ascending (aufsteigend, Standardwert) und descending (absteigend) gesteuert werden ✓ fehlt die order by Klausel, wird das Ergebnis entsprechend der Reihenfolge der Sequenz in der for Klausel geliefert	✓ optional
return	✓ steht immer am Ende des FLWOR-Ausdrucks ✓ liefert eine Einheit oder eine Sequenz	✓ mindestens einmal im Ausdruck

Ausdrücke in einer Anweisung können verschachtelt werden. So kann eine for Klausel eine innere for Klausel enthalten:

```
for $i in ('a', 'b' ) for $j in (1, 2 ) return element { $i } { $j }
```

Als Ergebnis liefert der Ausdruck <a>1 <a>2 1 2.

Das Ergebnis eines Ausdrucks kann wiederum Bestandteil eines anderen Ausdrucks sein. Zum Beispiel berechnet der Ausdruck

```
avg( for $i in (1 to 4) return $i * $i )
```

den Durchschnitt (7,5) auf Basis der Sequenz, die von dem in den Funktionsklammern stehenden FLWOR-Ausdruck geliefert wird.

Beispiel *kap14/xquery2_xsl.xql*

Im Beispiel werden mit einem FLWOR-Ausdruck alle Einzelinterpreten ermittelt, deren Album vor dem Jahr 1989 erschienen und damit mindestens 25 Jahre alt ist. Als Ergebnis wird eine XML-Struktur erstellt, welche unter dem neuen Wurzelement `EINZELINTERPRETEN_OLDIES` alle Alben mit den Elementen `INTERPRET`, `TITEL` und `JAHR` enthält.

```

xquery version "1.0";
① <EINZELINTERPRETEN_OLDIES>
{
②   let $doc := doc("musiksammlung.xml")
③   for $x in $doc//ALBUM
④   where $x/JAHR < 1989
⑤   order by $x/INTERPRET
⑥   return
⑦     <ALBUM>
⑧       { $x/INTERPRET }
⑨       { $x/TITEL }
       { $x/JAHR }
     </ALBUM>
}
</EINZELINTERPRETEN_OLDIES>

```

- ① Die Definition des Wurzelknotens `EINZELINTERPRETEN_OLDIES` für das zu erstellenden XML-Fragment.
- ② Der Variablen `$doc` wird über die Funktion `doc` das XML-Dokument *musiksammlung.xml* als Datenmodell zugewiesen.
- ③ Die Iteration wird über alle Knoten `ALBUM` des festgelegten Datenmodells ausgeführt. Über die Variable `$x` kann auf den im Durchlauf aktuellen Knoten zugegriffen werden.
- ④ Als Bedingung für zu selektierende Knoten wird bestimmt, dass der Inhalt des Elements `JAHR` kleiner als 1989 sein muss.
- ⑤ Die Ausgabe der Ergebnisknoten soll sortiert nach dem Inhalt des Elements `INTERPRET` erfolgen.
- ⑥ - ⑨ Das Ergebnis des Ausdrucks wird in der `return` Klausel bestimmt. Für jeden gefundenen Wert soll ein Element `ALBUM` mit den Unter-elementen `INTERPRET`, `TITEL` und `JAHR` geliefert werden. Der Zugriff auf den Inhalt der Unterknoten erfolgt unter Verwendung des in der Variablen `$x` gespeicherten aktuellen Knotens.

```

<?xml version="1.0"?>
- <EINZELINTERPRETEN_OLDIES>
  - <ALBUM>
    <INTERPRET>Ludwig van Beethoven</INTERPRET>
    <TITEL stil="Klassik">Berühmte Klavier-Sonaten</TITEL>
    <JAHR>1988</JAHR>
  </ALBUM>
  - <ALBUM>
    <INTERPRET>Philip Glass</INTERPRET>
    <TITEL stil="Klassik" bewertung="4">Koyaanisqatsi</TITEL>
    <JAHR>1983</JAHR>
  </ALBUM>
</EINZELINTERPRETEN_OLDIES>

```

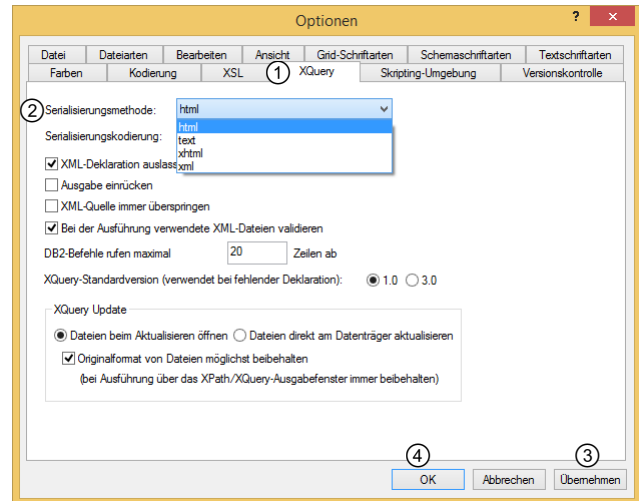
Ergebnis in der Browserdarstellung

14.4 Gestaltung der Ausgabe der Auswertungsergebnisse

Eine XQuery-Abfrage muss im Ergebnis nicht zwangsläufig ein XML-Fragment liefern. Je nach verwendetem XQuery-Prozessor kann dieser auf Anweisung hin auch reines Textdokument oder ein HTML- bzw. XHTML-Dokument erzeugen. Festgelegt wird dies über die Serialisierungsmethode des Prozessors.

Um im Programm XMLSpy diese Einstellung vorzunehmen, gehen Sie wie folgt vor:

- ▶ Öffnen Sie über *Extras - Optionen* das Dialogfenster *Optionen*.
- ▶ Wechseln Sie auf das Register *XQuery* ①.
- ▶ Wählen Sie unter *Serialisierungsmethode* die gewünschte Einstellung *html*, *text*, *xhtml* oder *xml* ②.
- ▶ Bestätigen Sie die Einstellung mit *Übernehmen* ③ und schließen das Dialogfenster über *OK* ④.



Beispiel *kap14/xquery3_xsl.xql*

Das Beispiel aus dem letzten Abschnitt wird um eine HTML-Grundstruktur und um Tags zur Erzeugung einer Liste ergänzt. Die Ausführung der Abfrage erzeugt ein gültiges HTML-Dokument.

```

① xquery version "1.0";
   <html encoding="utf-8">
     <head>
       <title>EINZELINTERPRETEN OLDIES</title>
     </head>
     <body>
       <ul>
         {
           let $doc := doc("musiksammlung.xml")
           for $x in $doc//ALBUM
           where $x/JAHR < 1989
           order by $x/INTERPRET
           return
           ② <li> Interpret: {data( $x/INTERPRET )},
              Titel:      {data( $x/TITEL )},
              Jahr:       {data( $x/JAHR )}, </li>
         }
       </ul>
     </body>
   </html>

```

- ① Die benötigten HTML-Tags werden an den notwendigen Stellen in die XQuery-Abfrage eingefügt. Die verwendete HTML-Grundstruktur ist nicht zwingend notwendig. Die Ausgabe könnte sich auch lediglich auf den Bereich der Listentags `` und `` beschränken.
- ② Die Funktion `data()` bewirkt, dass lediglich der Inhalt der Knoten und nicht die Knoten selbst im Ergebnis geliefert werden.

- Interpret: Ludwig van Beethoven, Titel: Berühmte Klavier-Sonaten, Jahr: 1988,
- Interpret: Philip Glass, Titel: Koyaanisqatsi, Jahr: 1983,

Darstellung des Ergebnisses im Browser

14.5 Auswertung verbundener Dokumente

Eine besondere Stärke von XQuery besteht darin, mehrere Datenquellen bei einer Auswertung heranzuziehen. Dabei können über Vergleichsoperationen miteinander verbundene Inhalte kombiniert und so im Rückgabergebnis Teile der unterschiedlichen Datenquellen gemeinsam geliefert werden.

Eine weitere Möglichkeit zur Auswertung mehrerer Datenquellen stellt die Funktion `collection()` dar, der eine Liste von mehreren Datenquellen übergeben werden kann.



Beispiel *kap14/labels.xml* und *kap14/labels.dtd*

Zur Demonstration wird zusätzlich zur Musiksammlung ein XML-Dokument *labels.xml*, welches auf der DTD *labels.dtd* basiert, verwendet.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE LABELS SYSTEM "labels.dtd" >
<LABELS>
  <LABEL>
    <NAME>Mute Records Limited</NAME>
    <MAINFOFFICE>Detroit</MAINFOFFICE>
    <WEB>www.muterec.com</WEB>
  </LABEL>
  <LABEL>
    ...
  </LABEL>
  ...
</LABELS>
<!ELEMENT LABELS (LABEL+)>
<!ELEMENT LABEL (NAME, MAINFOFFICE, WEB)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT MAINFOFFICE (#PCDATA)>
<!ELEMENT WEB (#PCDATA)>
```

Beispiel *kap14/xquery4_xsl.xql*

In der XQuery-Abfrage werden zum Inhalt des Knotens `LABEL` von jedem Album die zugehörigen Zusatzinformationen aus der Datei *labels.xml* ermittelt.

```
xquery version "1.0";
<LABELLISTE>
{
  ① for $a in doc("musiksammlung.xml")//ALBUM
  ② order by $a//LABEL
  ③ return
  ④ <COMPANY>
  ⑤ {
  ⑥   $a/LABEL,
   for $b in doc("labels.xml")//LABEL[NAME=$a/LABEL]
   return
   <DETAILS>
     { $b/MAINFOFFICE }
     { $b/WEB }
   </DETAILS>
 }
  </COMPANY>
}
</LABELLISTE>
```

- ① In der äußeren `for` Schleife wird die Musiksammlung `musiksammlung.xml` geöffnet. Für alle Knoten des Elements `ALBUM` wird der mit `return` definierte Ausdruck ermittelt. Der Zugriff auf den aktuellen Knoten kann über die Variablen `$a` erfolgen.
- ② Die Ausgabe erfolgt aufsteigend sortiert nach dem Inhalt des Elements `LABEL`.
- ③ Unterhalb des Wurzelknotens `LABELLISTE` des Ergebnisses wird im Rückgabewert das Unterelement `COMPANY` definiert.
- ④ Für den aktuellen Knoten in der Iteration werden der Tag `LABEL` und sein Inhalt ausgegeben.
- ⑤ In der inneren `for` Schleife wird die Datei `labels.xml` geöffnet. Der Inhalt des Knotens `Name` wird jeweils mit dem aktuellen Knoten `LABEL` der Musiksammlung verglichen. Der Vergleich erfolgt dabei über den kompletten Namen, da die Entity in `musiksammlung.xml` zum Zeitpunkt der Durchführung der XQuery-Abfrage bereits aufgelöst ist. Für den aktuellen Knoten der Iteration der inneren `for` Schleife wird die Variablen `$b` definiert.
- ⑥ Der Rückgabewert der inneren `for` Schleife setzt sich aus dem Unterelement `DETAILS` sowie den Tags `MAINFOFFICE` und `WEB` sowie deren aktuellen Inhalten zusammen.

```
<?xml version="1.0"?>
- <LABELLISTE>
  - <COMPANY>
    <LABEL>Island Records Limited</LABEL>
    - <DETAILS>
      <MAINFOFFICE>New York</MAINFOFFICE>
      <WEB>www.islandrecords.com</WEB>
    </DETAILS>
  </COMPANY>
  - <COMPANY>
    <LABEL>Jarrett Records</LABEL>
    - <DETAILS>
      <MAINFOFFICE>Boston</MAINFOFFICE>
      <WEB>www.jrm.com</WEB>
    </DETAILS>
  </COMPANY>
+ <COMPANY>
+ <COMPANY>
+ <COMPANY>
</LABELLISTE>
```

Darstellung des Ergebnisses mit den verknüpften Informationen im Browser

14.6 Übungen

Übung 1: Mit FLWOR-Ausdrücken arbeiten

Übungsdatei: *kap14/kfz.dtd*,
kap14/uebung1.xml

Ergebnisdatei: *kap14/uebung1.xql*

- ① Ermitteln Sie mit einem FLWOR-Ausdruck in einer XQuery-Abfrage alle Fahrzeuge, deren Höchstgeschwindigkeit mindestens 250 km/h beträgt.
- ② Geben Sie für alle ermittelten Fahrzeuge den Hersteller und die Modellbezeichnung aus. Die Ausgabe soll nach dem Hersteller in absteigender Sortierung erfolgen.

```
<?xml version="1.0"?>
- <FAHRZEUGE>
  - <KFZ>
    <HERSTELLER>MERCEDES</HERSTELLER>
    <MODELL>SLK 32 AMG</MODELL>
  </KFZ>
  - <KFZ>
    <HERSTELLER>BMW</HERSTELLER>
    <MODELL>BMW Z8</MODELL>
  </KFZ>
</FAHRZEUGE>
```

Ergebnisdarstellung im Browser

15 DOM und SAX

In diesem Kapitel erfahren Sie

- ✓ wie Sie ein DOM-Objekt anlegen und damit auf die XML-Struktur zugreifen
- ✓ welcher Unterschied zwischen DOM und SAX besteht
- ✓ wie Sie über JavaScript Daten eines XML-Dokuments auslesen
- ✓ wozu Dateninseln genutzt werden

Voraussetzungen

- ✓ Kenntnisse in JavaScript

15.1 DOM

Mit der Entwicklung von XML hat das W3-Konsortium das **Document Object Model**, kurz DOM, verabschiedet. Dieses Objektmodell bildet die Struktur eines XML-Dokuments im Arbeitsspeicher ab und erlaubt den Zugriff auf die Elementstruktur eines XML-Dokuments über eine API-Schnittstelle (API = Application Programming Interface). Über bestimmte Methoden und Eigenschaften können somit die einzelnen Elemente, Attribute und Inhalte angesprochen werden. Dazu muss die Anwendung in einer objektorientierten Programmier- oder Skriptsprache verfasst sein, beispielsweise in C++, Java, Delphi, VBScript, Perl, PHP oder JavaScript. Über diese Schnittstelle haben Sie die Möglichkeit, die Daten eines XML-Dokuments in einer Anwendung zu verarbeiten oder zu verändern. In diesem Abschnitt lernen Sie, wie Sie mithilfe von JavaScript in den aktuellen Browsern direkt auf einzelne XML-Daten zugreifen und diese verändern können. Dieses Modell kann beispielsweise über Java oder PHP auf ähnliche Weise verwendet werden.

Der Nachteil von DOM gegenüber dem später in diesem Kapitel erläuterten Modell SAX ist, dass bei umfangreichen XML-Dokumenten das Erzeugen des Dokumentenbaums sehr zeitintensiv sein kann. Zudem besteht ein hoher Bedarf an Arbeitsspeicher, da die gesamte XML-Struktur während der Bearbeitung im Speicher gehalten werden muss.

HTML und DOM

Eine HTML-Datei kann als XML-Dokument betrachtet werden, da sie eine XML-ähnliche Struktur der verschiedenen Elemente besitzt. Daher kann auch das DOM auf HTML-Dokumente angewandt werden, solange die Browser die entsprechenden JavaScript-Anweisungen zum Ansprechen des DOMs verarbeiten können.

Sie können die Elemente, die Attribute und den Inhalt jeder HTML-Datei auslesen und löschen, aber auch neue Elemente erstellen und in das Dokument einfügen. Somit haben Sie die Möglichkeit, Dokumente **dynamisch** zu generieren, ohne dass der Browser erneut eine Verbindung zum Webserver herstellen muss. Dies ist möglich, da das DOM den Zugriff auf jedes Element eines XML-Dokuments erlaubt.



Die Technik der dynamischen Änderung von Webseiten, ohne dass dabei eine Kommunikation mit dem Server erfolgen muss, wird als **Ajax** (Asynchrones JavaScript und XML) bezeichnet. Ajax beruht auf den vier Komponenten JavaScript, CSS, DOM und dem XMLHttpRequest-Objekt (vgl. den folgenden Abschnitt). Obwohl in Ajax-Anwendungen auf Grund der immer noch nicht vorhandenen einheitlichen XML-Unterstützung der einzelnen Browser aktuell in stärkerem Maße **JSON** (JavaScript Object Notation) statt XML für die Darstellung der Daten zum Einsatz kommt, ist XML dennoch weiterhin ein wichtiger Bestandteil von Ajax.

15.2 Erzeugen eines XML-DOM

Das Erzeugen eines XML-DOM-Objekts erfolgt über zwei verschiedene Wege. Der erste ist das Erzeugen eines Document Object Models über die API `XMLHttpRequest`.

Erzeugen des Objekts über `XMLHttpRequest`

`XMLHttpRequest` ist eine API zum Laden von Daten über das HTTP-Protokoll. Die Art der Daten ist hierbei egal. Es kann sich um reinen Text oder formatierte XML-Elemente handeln.

Um mit dem Browser per JavaScript auf ein XML-Objekt zuzugreifen, muss dieses Objekt als Instanz von `XMLHttpRequest` angelegt werden.

```
<script type="text/javascript">
  ObjektName = new XMLHttpRequest();
</script>
```

Zum Laden und Verarbeiten der Daten sind zwei Methoden notwendig.

```
open(Methode, URL[, Asynchron, Benutzername, Passwort])
send(Zeichenkette)
```

Mit der Methode `open` laden Sie die zu verarbeitenden Daten. Über den Parameter `Methode` geben Sie die Art und Weise des Datenzugriffs über HTTP an. Meist wird die Methode `GET` oder `POST` verwendet. Mit `URL` geben Sie ein Skript oder eine Datei an, die vom Server geladen werden soll.

Lokale XML-Dateien lassen sich aus Sicherheitsgründen im Browser nicht über die Methode `open` laden. Es ist zwingend notwendig, dass die XML- und HTML-Dateien auf einem Webserver abgelegt werden. Mit Ausnahme des Mozilla Firefox funktionieren deshalb die Beispiele des folgenden Abschnitts bei einer lokalen Ausführung nicht. Wenn Sie einen anderen Browser nutzen, laden Sie die Beispieldateien auf einen Ihnen verfügbaren freizugänglichen Webserver. Alternativ können Sie auch einen lokalen Webserver nutzen. Eine Möglichkeit zu dessen Einrichtung bietet die Installation von XAMPP, einer Web-Entwicklungsumgebung (<https://www.apachefriends.org/de/index.html>).



Der optionale Wert von `Asynchron` legt fest, ob das Skript weiter ausgeführt werden soll, solange noch keine Daten zurückgeliefert worden sind (`true`), oder ob der Browser so lange warten soll, bis alle Daten in das Objekt geladen worden sind (`false`). In unseren Beispielen ist es sinnvoll, auf die komplette Datenrückgabe zu warten. Wenn notwendig können der Benutzername und das Passwort für den Zugriff auf die URL hinterlegt werden.

Mit der Methode `send` wird das Laden der Daten gestartet.

```
responseText()
responseXML()
```

Je nachdem, welche Datenstruktur von dem aufgerufenen Skript erwartet wird, können Sie die Daten dem Objekt über die Methode `responseText` als Text übergeben. Mit `responseXML` wird die XML-Struktur dem Objekt als Document Object Model übergeben, auf das Sie mit den nachfolgenden Methoden und Eigenschaften zugreifen können.

Browserweiche zum Erzeugen des Objekts

Das `XMLHttpRequest` wird von allen aktuellen Browsern unterstützt. Damit auch immer noch verwendete ältere Browser wie der Internet Explorer in den Versionen 6 und 7 das entsprechende Objekt erzeugen können und mit den nachfolgenden Eigenschaften und Methoden auf die einzelnen Elemente zugegriffen werden kann, wird eine sogenannte Browserweiche verwendet.

Beispiel: *kap15\loadxmldoc.js*

```

① function loadXMLDoc(filename) {
②   if (window.XMLHttpRequest) {
      xhttp=new XMLHttpRequest();
③   } else {
      xhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
④   xhttp.open("GET", filename, false);
⑤   xhttp.send();
⑥   return xhttp.responseXML;
}

```

- ① Die Browserweiche wird als JavaScript-Funktion `loadXMLDoc` hinterlegt.
- ② Mit `window.XMLHttpRequest` wird überprüft, ob der aktuell verwendete Browser diese API und deren Methoden kennt. Ist dies der Fall, wird das XML-Objekt über diese Methode erzeugt. Dieser Zweig wird von den aktuellen Browsern ausgeführt. Das Objekt `xhttp` wird erstellt.
- ③ Ältere Versionen des Internet Explorers benötigen das entsprechende ActiveX-Objekt.
- ④ Hier werden die Parameter zum Senden der Anfrage festgelegt. Die Daten sollen über die Methode `GET` übertragen werden. Die Variable `filename`, deren Wert mit der Funktion `loadXMLDoc` übergeben wird, beinhaltet die URL des aufzurufenden Skripts oder der XML-Datei. Die Datenübertragung soll nicht asynchron erfolgen (`false`). Der Browser soll auf die Rückmeldung des Servers warten.
- ⑤ Die Methode `send` startet die Datenabfrage.
- ⑥ Die Rückgabe der Daten wird im XML-Format erwartet und entsprechend im Objekt `xhttp` abgelegt und als Ergebnis der Funktion zurückgeliefert.

15.3 Ansprechen der Knotenelemente

Das DOM ist ein Modell, das die Verbindung der verschiedenen Objekte mit anderen Objekten eines Dokuments anzeigt. Im DOM stellt jeder Knoten ein Objekt in der Baumstruktur des Dokuments dar. So besteht beispielsweise die HTML-Angabe

```
<div>Inhalt des Blocks</div>
```

aus zwei Knoten, dem umschließenden Element `div` und dem Text "Inhalt des Blocks". Da sich der Textknoten innerhalb des Elements `div` befindet, wird er auch als Kind-Knoten (child node) des Elements `div` bezeichnet. Das umschließende Element wird Eltern-Knoten (parent node) genannt.

```

<div>
  Inhalt des Blocks
</div>

```

Befindet sich innerhalb des Elements `div` ein weiteres Element, wie beispielsweise

```
<div>Inhalt <i>des Blocks</i></div>
```

enthält das Element `div` zwei Kind-Knoten, das Wort `Inhalt` sowie das Element `i`. Der Text "des Blocks" ist nun nicht mehr das Kind des Elements `div`, sondern das Kind des Elements `i`.

```

<div>
  Inhalt
  <i>
    des Blocks
  </i>
</div>

```

Außerdem gibt es noch die Attributknoten (attribute nodes). Diese entstehen, wenn bei einem Element ein Attribut angegeben wird.

```
<div style="text-align:center;">Inhalt <i>des Blocks</i></div>
```

Das Element `div` enthält nun drei Knoten, denn das Attribut `style` wird als Attributknoten betrachtet.

```
<div
  style="text-align:center;">
  Inhalt
  <i>
    des Blocks
  </i>
</div>
```

Knoten durchlaufen

Nach dem Festlegen des XML-Objekts benötigen Sie die Methoden und Eigenschaften zum Ansprechen der einzelnen Knotenpunkte und deren Inhalte. Die nachfolgende Tabelle stellt die Methoden und Eigenschaften des W3C DOM Level 3 dar, mit denen Sie in der DOM-Baumstruktur die Elemente durchlaufen können.

Methode/Eigenschaft	Erläuterung
<code>childNodes</code>	Alle Unterknoten (Kind-Knoten) des selektierten Knotens werden ausgelesen.
<code>parentNode</code>	Der übergeordnete Knoten (Eltern-Knoten) des selektierten Knotens wird angesprochen.
<code>hasChildNodes</code>	Damit können Sie prüfen, ob der selektierte Knoten weitere Unterknoten besitzt.
<code>firstChild</code>	Der erste untergeordnete Knoten wird selektiert.
<code>lastChild</code>	Der letzte untergeordnete Knoten wird selektiert.
<code>previousSibling</code> <code>nextSibling</code>	Hiermit erhalten Sie den vorherigen bzw. folgenden Schwester-Knoten des selektierten Knotens.
<code>getAttribute("Name")</code>	Diese Eigenschaft liefert vom selektierten Knoten den Wert des mit Name angegebenen Attributs.
<code>Attributes()</code>	Der Attributwert eines selektierten Knotens wird über den Index ausgelesen (0 = erster Attributwert; 1 = zweiter Attributwert; 2 = dritter Attributwert usw.).
<code>nodeType</code>	Diese Eigenschaft liefert die Art des selektierten Knotens als Zahl zurück. Die wichtigsten sind: 1 = Elementknoten, 2 = Attributknoten, 3 = Textknoten.
<code>nodeName</code> <code>nodeValue</code>	Der Name bzw. der Wert oder Inhalt eines selektierten Knotens wird ausgegeben.
<code>tagName</code>	Hiermit erhalten Sie den Namen des Elements.
<code>textNode</code> <code>text</code>	Mit <code>textNode</code> ermitteln Sie den Inhalt eines selektierten Elements. Die Methode <code>text</code> ist die entsprechende Methode für den Internet Explorer und entspricht dem veralteten W3C DOM Level 2.

Zusätzlich haben Sie die Möglichkeit, über zwei Methoden bestimmte Elemente direkt anzusprechen.

Methode	Erläuterung
<code>getElementById("id")</code>	Diese Methode liefert ein spezielles Element, das mit dem Attribut <code>id</code> eine eindeutige Kennung aufweist.
<code>getElementsByTagName("name")</code>	Hiermit erhalten Sie alle Elementknoten, die dem gesuchten Namen entsprechen, als Array. Die einzelnen Knoten sprechen Sie über den Index an, z. B. <code>getElementsByTagName("table")[0]</code> .

Beispiel: *kap15\knoten.html*

Die Datenstruktur einer XML-Datei wird als Objekt erstellt. Über die Methoden und Eigenschaften des Objekts können die einzelnen XML-Knoten angesprochen werden.

	<code><html lang="de"></code>
	<code><head></code>
①	<code><title>Musiksammlung</title></code>
	<code><script type="text/javascript" src="loadxmldoc.js"></script></code>
	<code></head></code>
	<code><body></code>
	<code><script type="text/javascript"></code>
②	<code>XMLObjekt = loadXMLDoc("musiksammlung.xml");</code>
③	<code>Knoten = XMLObjekt.firstChild;</code>
④	<code>document.write('Hauptelement: ' + Knoten.nodeName + '<br \/>');</code>
⑤	<code>KnotenElement = XMLObjekt.getElementsByTagName('TITEL')[0];</code>
⑥	<code>document.write('Knotenelement: ' + KnotenElement.nodeName + '<br \/>');</code>
⑦	<code>document.write('Attribut ' + KnotenElement.attributes[0].nodeName +</code>
	<code>' mit Wert ' + KnotenElement.attributes[0].nodeValue + '<br \/>');</code>
⑧	<code>document.write('Attribut ' + KnotenElement.attributes[1].nodeName +</code>
	<code>' mit Wert ' + KnotenElement.attributes[1].nodeValue + '<br \/>');</code>
	<code></script></code>
	<code></body></code>
	<code></html></code>

- ① Die JavaScript-Datei *loadxmldoc.js* mit der Funktion zum Laden der XML-Struktur wird geladen.
- ② Dem Objekt `XMLObjekt` werden über die JavaScript-Funktion `loadXMLDoc()` die XML-Daten der Datei *musiksammlung.xml* übergeben.
- ③ Der Variablen `Knoten` wird das erste Kind der XML-Struktur zugewiesen.
- ④ Der Name des Knotens wird über die Eigenschaft `nodeName` im Browser ausgegeben.
- ⑤ Des Weiteren werden mit der Methode `getElementsByTagName` im Objekt `XMLObjekt` die Knoten mit dem Namen `TITEL` selektiert. Über die Angabe des Index 0 wird der Variablen `KnotenElement` der erste Knoten zugewiesen.
- ⑥ Der Name des Knotens wird ausgegeben. In dem Fall lautet er `TITEL`.
- ⑦ Über die Eigenschaft `attributes` erhalten Sie Zugriff auf alle Attribute des selektierten Knotens. Über die Angabe von `attributes[0]` sprechen Sie das erste Attribut an und können dessen Namen und Wert über die entsprechenden Eigenschaften ausgeben lassen.
- ⑧ Den Namen und Wert des zweiten Attributs erhalten Sie durch die Angabe des nächsten Index `attributes[1]`.



Die Methoden beim Zugriff auf die Elemente einer XML-Struktur müssen Sie nicht zwingend in einer Variablen speichern (`Knoten = XMLObjekt.firstChild`); Sie können auch die Eigenschaft eines Elements direkt aufrufen, z. B. `XMLObjekt.firstChild.nodeName`.

Zur Veranschaulichung des Aufrufs der Eigenschaften werden die Methoden noch einmal in Tabellenform dargestellt. In der entsprechenden Spalte finden Sie den Teil der XML-Struktur, der durch die Methode oder Eigenschaft angesprochen wird.

firstChild	childNodes	attributes	nodeValue	
<HAUPTELEMENT>				
	<ELEMENT1	attr="xyz">	Inhalt1	</ELEMENT1>
	<ELEMENT2	attr="uvw">	Inhalt2	</ELEMENT2>
</HAUPTELEMENT>				

Um beispielsweise den Inhalt des Elements `ELEMENT2` auszulesen, verwenden Sie die Anweisung `XMLObjekt.childNodes[1].nodeValue`; Den Attributwert des ersten Elements `Element1` erhalten Sie, wenn Sie die Anweisung `XMLObjekt.childNodes[0].attributes[0].nodeValue` verwenden.

Beispiel: *kap15\person.xml*

Die folgende XML-Struktur wird als Beispiel verwendet. Sie enthält zwei Elemente mit dem Attribut `geschlecht` und unterschiedlichen Attributwerten.

	<code><?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?></code>
①	<code><DATEN></code>
②	<code><PERSON geschlecht="männlich">Max Mustermann</PERSON></code>
	<code><PERSON geschlecht="weiblich">Sabine Salzmänn</PERSON></code>
	<code></DATEN></code>

- ① In der XML-Datei wird das Hauptelement `DATEN` definiert.
- ② Die beiden Elemente `PERSON` enthalten das Attribut zum Bestimmen des Geschlechts sowie den Namen der entsprechenden Person.

Beispiel: *kap15\DOM.html*

In dem nachfolgenden Beispiel sollen die Daten und Attributwerte der einzelnen Elemente der XML-Datei angesprochen und im Browser angezeigt werden.

	<code><html lang="de"></code>
	<code><head></code>
	<code><title>Musiksammlung</title></code>
①	<code><script type="text/javascript" src="loadxml.doc.js"></script></code>
	<code></head></code>
	<code><body></code>
	<code><script type="text/javascript"></code>
②	<code>XMLObjekt = loadXMLDoc("person.xml");</code>
③	<code>xml_daten = XMLObjekt.getElementsByTagName('PERSON')[0];</code>
④	<code>document.write('<p>')</code>
	<code>document.write(xml_daten.childNodes[0].nodeValue + '<br \/>');</code>
	<code>document.write(xml_daten.attributes[0].nodeName + ' = ' +</code>
	<code>xml_daten.attributes[0].nodeValue);</code>
⑤	<code>xml_daten = XMLObjekt.getElementsByTagName('PERSON')[1];</code>
⑥	<code>document.write('<p>')</code>
	<code>document.write(xml_daten.childNodes[0].nodeValue + '<br \/>');</code>
	<code>document.write(xml_daten.attributes[0].nodeName + ' = ' +</code>
	<code>xml_daten.attributes[0].nodeValue);</code>
	<code></script></code>
	<code></body></code>
	<code></html></code>

- ① Die JavaScript-Datei *loadxml/doc.js* wird geladen.
- ② Die zuvor erstellte XML-Datei *person.xml* wird in das Objekt geladen.
- ③ Mit `getElementsByTagName` wird auf den ersten Knoten mit dem Namen `PERSON` zugegriffen.
- ④ Über die JavaScript-Methode `document.write()` werden im Browser der Wert des Knotens sowie der Name und Wert des ersten Attributs ausgegeben.
- ⑤ Auf das nächste Element `PERSON` erhalten Sie Zugriff über die Angabe des Index `[1]`.
- ⑥ Auch hier werden der Knoteninhalt sowie der Attributsname und dessen Wert angezeigt.

Max Mustermann
geschlecht = männlich

Sabine Salzmänn
geschlecht = weiblich

*Ausgabe der XML-Daten
im Browser*

Knoten automatisiert auslesen

Damit Sie alle Unterelemente eines XML-Dokuments ansprechen können, muss eine Schleife verwendet werden. Diese Schleife wird dann so lange durchlaufen, bis alle Unterelemente ausgelesen sind.

Oberste Ebene	1. Ebene	2. Ebene
<MUSIKSAMMLUNG>	<ALBUM>	<AUTOR></AUTOR> <INTERPRET></INTERPRET> <TITEL stil="" bewertung=""></TITEL> <GESAMTZEIT></GESAMTZEIT> <LABEL></LABEL> <JAHR></JAHR> <BILD typ="" quelle="" /> <LIED></LIED>
</MUSIKSAMMLUNG>	</ALBUM>	<AUTOR></AUTOR> <INTERPRET></INTERPRET> <TITEL stil="" bewertung=""></TITEL> <GESAMTZEIT></GESAMTZEIT> <LABEL></LABEL> <JAHR></JAHR> <BILD typ="" quelle="" /> <LIED></LIED>

Diese Tabelle stellt die grundlegende Struktur der Musiksammlung dar. Die erste Ebene kann sich beliebig oft wiederholen, wobei auch die zweite Ebene dementsprechend oft angelegt wird.

Die Anzahl der Schleifendurchläufe richtet sich nach der Anzahl der Elemente einer Ebene. So soll die Menge der Elemente der ersten und zweiten Ebene bestimmt werden. Es wird das erste Element der ersten Ebene selektiert und alle Unterelemente (zweite Ebene) werden ausgegeben. Danach werden das zweite Element der ersten Ebene und seine Unterelemente angesprochen, dann das dritte Element usw.

Die Anzahl der Elemente einer Ebene wird über die Eigenschaft `length` ermittelt:

```
selektierterKnoten.length
```



Bei der Abfrage, wie viele Unterknoten ein Element besitzt, müssen Sie beachten, dass immer auch der Textknoten mitgezählt wird.

Es sollen die Inhalte aller Elemente ausgegeben werden. Dazu sind die folgenden Schritte notwendig:

- ✓ Selektieren der obersten Ebene;
- ✓ Anzahl der Elemente der ersten Ebene ermitteln (`childNodes.length`);
- ✓ Schleife durchlaufen, in der alle Elemente der obersten Ebene ausgelesen werden;
- ✓ innerhalb der Schleife die Anzahl der Elemente der zweiten Ebene ermitteln (`Kind(i).childNodes.length`);
- ✓ weitere Schleife bilden, um die Inhalte aller Elemente der zweiten Ebene auszulesen.

Damit Sie die Inhalte der Elemente ausgeben können, müssen Sie in der Datei *loadxml.doc.js* eine weitere Funktion hinterlegen. Diese ist notwendig, weil in den Browsern, die den W3C DOM Level 3 unterstützen, die Inhalte der Elemente über `textContent` angesprochen werden. In den Browsern, die nur den W3C DOM Level 2 unterstützen, benötigen Sie die Eigenschaft `text`.

Beispiel: *kap15\loadxml.doc.js*

```

① function getNodeText(xmlNode) {
    if (xmlNode != null) {
        // W3C DOM Level 3
②     if (typeof xmlNode.textContent != 'undefined') {
            return(xmlNode.textContent);
        }
        // W3C DOM Level 2
③     else if (typeof xmlNode.text != 'undefined') {
            return(xmlNode.text);
        }
    }
}

```

- ① Die Funktion wird `getNodeText` genannt und erwartet als Parameter den ausgewählten XML-Knoten, der innerhalb der Funktion über den Namen `xmlNode` angesprochen werden kann.
- ② Ist dem Browser die Knoten-Eigenschaft `textContent` bekannt und liefert er somit nicht den Wert `undefined` zurück, dann wird der entsprechende Wert zurückgeliefert.
- ③ Ansonsten wird, wenn der Knoten `xmlNode` die Eigenschaft `text` besitzt, dessen Wert zurückgeliefert.

Beispiel: *kap15\musik1.html*

```

<script type="text/javascript">
① XMLObjekt = loadXMLDoc("musiksammlung.xml");
② Kind = XMLObjekt.getElementsByTagName('ALBUM');
  document.write('<br \/>Anzahl der Knoten: ' + Kind.length);

  /* Schleife für die 1.Ebene */
③ for (i=0; i < Kind.length; i++) {
    document.write('<br \/><b>Anzahl der unteren Knoten in Knoten[' + i + ']: ' +
      + Kind[i].childNodes.length + '<\b>');

    /* Schleife für die 2.Ebene */
④ for (j=0; j < Kind[i].childNodes.length; j++) {
⑤     if (Kind[i].childNodes[j].nodeType == 1) {
⑥     document.write('<br \/>Inhalt des Knotens[' + i + ',' + j + ']: ' +
      Kind[i].childNodes[j].nodeName + ' - ' +
      getNodeText(Kind[i].childNodes[j]));
    }
  }
}
</script>

```

- ① Die Elemente der XML-Datei *musiksammlung.xml* werden in das Objekt geladen.
- ② Die Variable `Kind` enthält die Kind-Knoten der Elemente `ALBUM`. Dies ist notwendig, damit auf die weiteren Unterelemente zugegriffen werden kann. Wie viele Elemente vorhanden sind, wird über die Eigenschaft `length` angezeigt.
- ③ Die `for`-Schleife wird initialisiert und für alle Elemente der ersten Ebene (`Kind.length`) durchlaufen. Ausgegeben wird hierbei die Anzahl der jeweiligen Unterknoten. Hier werden Sie feststellen, dass alle Browser doppelt so viele wie der Internet Explorer ausgeben. Dies ist darauf zurückzuführen, dass der Internet Explorer nur die tatsächlichen Knoten zählt, die anderen Browser aber auch die Whitespace-Zeichen, z. B. Zeilenumbrüche, zählen.
- ④ In einer zweiten Schleife werden genau diese jeweiligen Unterknoten näher untersucht.

- ⑤ Um auf das tatsächliche Knotenelement zugreifen zu können, fügen Sie eine Abfrage ein, ob es sich um einen Elementknoten (Knotentyp vom Wert 1) handelt.
- ⑥ Ist dies der Fall, können Sie den Namen des Knotens über die Eigenschaften `nodeName` ausgeben lassen. Den Inhalt lassen Sie über die definierte JavaScript-Funktion `getNodeText()` ermitteln, indem Sie den entsprechenden Knoten übergeben.

Als Ergebnis erhalten Sie die Anzahl der Knoten der ersten Ebene, die Anzahl der jeweiligen Unterelemente sowie die Bezeichnung jedes einzelnen Elements und dessen Inhalt.

```
Anzahl der Knoten: 5
Anzahl der unteren Knoten in Knoten[0]: 53
Inhalt des Knotens[0,1]: AUTOR - Heiko Schröder
Inhalt des Knotens[0,3]: INTERPRET - Moby
Inhalt des Knotens[0,5]: TITEL - Play
Inhalt des Knotens[0,7]: GESAMTZEIT - 63:03
Inhalt des Knotens[0,9]: LABEL - Mute Records Limited
Inhalt des Knotens[0,11]: JAHR - 1999
Inhalt des Knotens[0,13]: BILD -
Inhalt des Knotens[0,15]: BILD -
Inhalt des Knotens[0,17]: LIED - Honey
Inhalt des Knotens[0,19]: LIED - Find my baby
Inhalt des Knotens[0,21]: LIED - Porcelain
Inhalt des Knotens[0,23]: LIED - Why does my heart feel so bad?
Inhalt des Knotens[0,25]: LIED - South side
Inhalt des Knotens[0,27]: LIED - Ruckling
```

Anzeige der gefundenen Knoten

Knoten gezielt ansprechen

Die Möglichkeit, per Skript durch das Dokument zu laufen und jedes Element einzeln anzusprechen, ist umständlich. Daher haben Sie auch die Möglichkeit, ein Element direkt anzusprechen, ohne vorher dessen übergeordnete Elemente durchlaufen zu müssen.

Beispiel: *kap15\musik2.html*

Die Interpreten und Gruppen, die in der Datei *musiksammlung.xml* hinterlegt sind, sollen mit ihren bisher aufgenommenen Alben in einer übersichtlichen Tabelle aufgelistet werden. In der Spalte *Bemerkung* soll die jeweilige Anzahl der Lieder des Albums angezeigt werden.

Folgende Schritte sind beim Selektieren der Daten notwendig:

- ✓ Ermitteln der Anzahl der Elemente der ersten Ebene;
- ✓ Schleife festlegen, in der alle Elemente der ersten Ebene durchlaufen werden;
- ✓ Ausgeben der Elemente *INTERPRET* bzw. *GRUPPE* und *TITEL* aus der zweiten Ebene;
- ✓ Ermitteln der Anzahl der Elemente *LIED* im jeweiligen *ALBUM*.

```
<body>
  <h2>Musiksammlung</h2>
  ① <p>Datenumfang: <span id="counter"></span> Einträge</p>
  <div id="tabelle"></div>
  <script type="text/javascript">
    ② XMLObjekt = loadXMLDoc("musiksammlung.xml");
    ③ counter.innerHTML = XMLObjekt.getElementsByTagName('ALBUM').length;
    ④ output = '<table><tr class="firstline"><td><b>Interpret</b></td>
      <td><b>Album</b></td><td><b>Bemerkung</b></td></tr>';
    ⑤ Album = XMLObjekt.getElementsByTagName('ALBUM');
    ⑥ for (i=0; i < Album.length; i++) {
    ⑦   output = output + '<tr>';
```

```

    output = output + '<td>';
    ⑧ Interpret = Album[i].getElementsByTagName('INTERPRET');
    ⑨ if (Interpret.length == 0) {
        Interpret = Album[i].getElementsByTagName('GRUPPE');
    }
    ⑩ output = output + getNodeText(Interpret[0]);
    output = output + '</td>';

    output = output + '<td>';
    ⑪ Titel = Album[i].getElementsByTagName('TITEL');
    output = output + getNodeText(Titel[0]);
    output = output + '</td>';

    output = output + '<td>';
    ⑫ Lieder = Album[i].getElementsByTagName('LIED');
    ⑬ output = output + 'Lieder:' + Lieder.length;
    output = output + '</td>';

    output = output + '</tr>';
    ⑭ };
    output = output + '</table>';
    ⑮ tabelle.innerHTML = output;</script>
</body>

```

- ① Die Stellen zur Ausgabe von Werten werden definiert. Die Elemente `span` und `table`, die beide mit einer jeweiligen ID versehen werden, sollen per JavaScript mit Daten gefüllt werden.
- ② Die Elemente der XML-Datei *musiksammlung.xml* werden in das Objekt geladen.
- ③ Die Anzahl der Elemente `ALBUM` im XML-Dokument wird über die Suche nach dem Elementnamen (`getElementsByTagName`) ermittelt und über die Eigenschaft `innerHTML` an dem HTML-Tag ausgegeben, das mit der ID `counter` definiert wurde.
- ④ Es folgt der Aufbau der Tabelle, in der die Informationen angezeigt werden sollen. Zur besseren Übersicht werden diese Informationen in der Variablen `output` abgelegt.
- ⑤ Der Variablen `Album` wird das Objekt mit den gefundenen Alben zugewiesen. Das Ergebnis ist hierbei ein Array mit der entsprechenden Anzahl der Alben.
- ⑥ Mit der `for`-Schleife soll das Ansprechen der einzelnen Indizes des Arrays `Album` ermöglicht werden.
- ⑦ HTML-Anweisungen zum Aufbau der Tabelle werden in das Dokument geschrieben.
- ⑧ In der ersten Spalte der Tabelle wird der Name des Interpreten ausgegeben. Mit der Zählvariablen `i` ist es möglich, das jeweilige Element des Objekts `Album` anzusprechen. Hat die Variable `i` den Wert 0, wird das erste Album der Liste angesprochen, ist der Wert 1, das zweite Album usw. Die Methode `getElementsByTagName` liefert den Unterknoten des selektierten Albums, der den Namen des Interpreten enthält.
- ⑨ Sollte innerhalb des selektierten Albums kein Element `INTERPRET` vorhanden sein, wird nach dem Element `GRUPPE` gesucht und das Objekt der Variablen `Interpret` zugewiesen.
- ⑩ Da innerhalb des Elements `ALBUM` das Element `INTERPRET` bzw. `GRUPPE` nur einmal vorkommt, kann dessen Inhalt direkt über den Index 0 angesprochen werden. Die Funktion `getNodeText()` ist die selbst erstellte JavaScript-Funktion, die über die Datei *loadxml/doc.js* eingebunden ist.
- ⑪ Um den Titel des Albums herauszufiltern, wird innerhalb des Elements `ALBUM` nach den möglichen Elementen `TITEL` gesucht und diese in der Variablen `Titel` abgelegt. Der Knotentext des Elements `TITEL` wird über die JavaScript-Funktion ausgegeben.
- ⑫ Auch hier wird nach möglichen Elementen innerhalb des Elements `ALBUM` gesucht. In diesem Fall nach den Elementen `LIED`.
- ⑬ Statt des Namens eines jeden einzelnen Liedes wird über die Eigenschaft `length` die Anzahl der Lieder in die Variable `output` geschrieben.
- ⑭ Solange noch nicht alle Alben des XML-Dokuments ausgegeben sind, kehrt der Interpreter zurück zur Stelle ⑥ und führt die `for`-Schleife für das nächste Element `ALBUM` aus.
- ⑮ Ganz am Ende des Skripts wird der Inhalt der Variablen `output` an dem Element mit der ID `tabelle` als HTML-Text ausgegeben.

Musiksammlung

Datenumfang: 5 Einträge

Interpret	Album	Bemerkung
Moby	Play	Lieder:18
a-ha	Minor earth major sky	Lieder:13
Mesh	Fragmente	Lieder:12
Ludwig van Beethoven	Berühmte Klavier-Sonaten	Lieder:3
Philip Glass	Koyaanisqatsi	Lieder:6

*Formatierte Ausgabe der Musiksammlung***Daten separieren**

In diesem Abschnitt wird ein Beispiel vorgestellt, bei dem spezielle Informationen erst dann angezeigt werden, wenn diese mithilfe von JavaScript-Anweisungen angefordert werden.

Beispiel: *kap15/loadxml.doc.js*

In diesem Abschnitt wird bei Alben der Musiksammlung ein Hyperlink hinterlegt. Erst bei einem Klick auf den Namen des Albums soll sich ein weiteres Fenster öffnen, in dem die Daten des Albums aufgelistet werden. Dazu erstellen Sie die nachfolgende JavaScript-Funktion, die Sie in der Datei *loadxml.doc.js* einfügen.

```

① function OeffneInfoFenster(AlbumName) {
②     Fenster = open("", "Individuell", "resizable=yes, screenX=0, screenY=0,
        height=400, width=300");
        Fenster.focus();

③     with (Fenster) {
④         document.write('<html><head><title>' + AlbumName + '</title>');
        document.write('<link type="text/css" rel="stylesheet"
            href="musik_link.css" \/>');
        document.write('</head>');
        document.write('<body>');
        document.write('<table border="1" cellpadding="3">');

⑤         Titel = XMLObjekt.getElementsByTagName('TITEL');
⑥         i = -1;
⑦         if (Titel.length > 0)
⑧             do {
⑨                 i++;
⑩                 if (getNodeText(Titel[i]) == AlbumName) {
⑪                     Album = Titel[i].parentNode;
⑫                     for (j = 0; j < Album.childNodes.length; j++) {
⑬                         if (Album.childNodes[j].nodeType == 1) {
⑭                             document.write('<tr>');
                                document.write('<td>' + Album.childNodes[j].nodeName + ': </td>');
                                if (getNodeText(Album.childNodes[j]) != '') {
                                    document.write('<td>' + getNodeText(Album.childNodes[j]) + '</td>');
                                } else {
                                    document.write('<td>- Keine Angabe -</td>');
                                }
                                document.write('</tr>');
                            }
                        }
                    } while (getNodeText(Titel[i]) != AlbumName);

        document.write('</table>');
        document.write('</body></html>');
⑮     }
⑯     return

```

```

    }
  }
</script>

```

- ① Die JavaScript-Funktion `OeffneInfoFenster` muss mit einem Wert aufgerufen werden, der innerhalb der Funktion über die Variable `AlbumName` angesprochen werden kann. Dieser Wert soll den Titel des Albums enthalten.
- ② Es wird ein neues Fenster mit einer Höhe von 400 Pixeln und einer Breite von 300 Pixeln geöffnet. Dieses Fenster kann über das Objekt `Fenster` angesprochen werden.
- ③ Die Anweisung `with` sorgt dafür, dass sich die in geschweiften Klammern stehenden Anweisungen auf das geöffnete Fenster beziehen.
- ④ Mit `document.write` werden das Grundgerüst und der Beginn der HTML-Tabelle in das Dokument des neuen Fensters eingefügt.
- ⑤ Der Variablen `Titel` wird das Objekt mit den gefundenen Elementen `TITEL` zugewiesen.
- ⑥ Die Variable `i` wird mit dem Wert `-1` initialisiert und im weiteren Verlauf als Zählvariable zum Ansprechen der verschiedenen Elemente `TITEL` verwendet.
- ⑦ Mit der Methode `Titel.length` kann die Anzahl der gefundenen Elemente `TITEL` abgefragt werden. Die `if`-Anweisung überprüft, ob überhaupt ein Element mit der Bezeichnung `TITEL` im XML-Dokument gefunden wurde und somit die Anzahl größer null ist.
- ⑧ Diese Anweisung leitet die fußgesteuerte Wiederholung `do while` ein. Dies bedeutet, dass die Schleife durchlaufen wird, solange die Bedingung ⑮ erfüllt ist.
- ⑨ Die Zählvariable `i` wird erhöht.
- ⑩ Mithilfe der Funktion `getNodeText()` wird getestet, ob das angesprochene Element `TITEL` den an die Funktion übergebenen Namen des Albums enthält und somit angezeigt werden soll.
- ⑪ Der Variablen `Album` wird der dem selektierten Element `TITEL` übergeordnete Eltern-Knoten zugeordnet.
- ⑫ Innerhalb dieser `for`-Schleife sollen alle Daten des selektierten Albums ausgegeben werden. Somit wird die Schleife so oft durchlaufen, bis die Anzahl `Album.childNodes.length` erreicht ist.
- ⑬ Nach der Abfrage des Knotentyps `1` und der Ausgabe des Knotennamens wird ausgewertet, ob der Kind-Knoten des Albums einen Inhalt besitzt (`!= ''`). Ist dies der Fall, wird der Text des Kind-Knotens ausgegeben.
- ⑭ Besitzt das Element keinen Inhalt, wird als Alternative der Text `- Keine Angabe -` in das Dokument eingefügt.
- ⑮ Diese Zeile stellt den Fuß der Wiederholung `do while` dar. Wenn der Inhalt des selektierten Knotens `TITEL` nicht mit dem übergebenen Wert `AlbumName` übereinstimmt, wird die Schleife und somit die Datenausgabe beendet.
- ⑯ Die JavaScript-Funktion wird verlassen.

Beispiel: *kap15\musiklink.html*

In diesem Beispiel wird bei den Namen der Alben ein Hyperlink hinterlegt, über den der Inhalt des jeweiligen Albums angezeigt werden kann. Ersetzen Sie dazu den Quelltext aus dem Beispiel *musik2.html*

```

output = output + '<td>';
Titel = Album[i].getElementsByTagName('TITEL');
output = output + getNodeText(Titel[0]);
output = output + '</td>';

```

durch die nachfolgenden Zeilen.

```

① output = output + '<td>';
② Titel = Album[i].getElementsByTagName('TITEL');
③ TitelText = getNodeText(Titel[0]);
output = output + '<a href="javascript:OeffneInfoFenster(\'\' + TitelText
    + \'\'")">' + TitelText + '</a>';
output = output + '</td>';


```

- ① Der Variablen `Titel` werden die Elemente `TITEL` zugeordnet.
- ② Die Variable `TitelText` erhält als Wert den Inhalt des ersten Elements von `Titel`, also den Titel des selektierten Albums.
- ③ Die Definition eines Hyperlinks wird eingeleitet und in der Variablen `output` übergeben. Wird dieser Verweis vom Anwender ausgewählt, soll die JavaScript-Funktion `OeffneInfoFenster()` mit dem Wert von `TitelText` ausgeführt werden. Als sichtbarer Teil des Hyperlinks wird der Titel des selektierten Albums ausgegeben.

Musiksammlung

Datenumfang: 5 Einträge

Interpret	Album	Bemerkung
Moby	Play	Lieder: 18
a-ha	Minor earth major sky	Lieder: 13
Mesh	Fragmente	Lieder: 12
Ludwig van Beethoven	Berühmte Klavier-Sonaten	Lieder: 3
Philip Glass	Koyaanisqatsi	Lieder: 6



AUTOR:	Heiko Schroder
INTERPRET:	Philip Glass
TITEL:	Koyaanisqatsi
GESAMTZEIT:	46:26
LABEL:	Island Records Limited
JAHR:	1983
BILD:	- Keine Angabe -
LIED:	Koyaanisqatsi
LIED:	Vessel
LIED:	Cloudscape
LIED:	Pruit Igoe
LIED:	The Grid
LIED:	Prophecies

Ausgabe von weiteren Daten nach dem Auslösen des Hyperlinks

15.4 SAX

Die zweite Art, auf ein XML-Dokument zuzugreifen, ist die Verwendung von SAX (Kurzform für Simple API for XML). Dies ist ein ereignisgesteuerter Mechanismus zum Lesen der einzelnen XML-Elemente. Dabei wird nicht wie beim DOM ein Objektmodell des XML-Dokuments aufgebaut, sondern es werden Ereignisse ausgelöst, auf die der Programmierer reagieren kann.

Das XML-Dokument wird geparkt, und die entsprechenden Ereignisse werden ausgelöst, wenn im Dokument Folgendes gefunden wird:

- ✓ der Anfang eines Elements (Anfangs-Tag)
- ✓ der Inhalt eines Elements
- ✓ das Ende eines Elements (Ende-Tag)
- ✓ Kommentare
- ✓ Entity-Definitionen
- ✓ Fehler oder Warnungen

Ein SAX-Parser kann je nach Anforderung diese Ereignisse erzeugen. Der Nachteil dieses Parsers ist jedoch, dass ein Ereignis nur einmal erzeugt wird und somit nach dem Parsen nicht mehr auf die Elemente zugegriffen werden kann. Dies bedeutet beispielsweise, dass für weitere Zugriffe die Struktur des XML-Dokuments intern gespeichert werden muss oder der SAX-Parser erneut das XML-Dokument durchläuft.

Beispiel

```
<ZITAT>
  Sein oder nicht sein, das ist hier die Frage.
</ZITAT>
```

Der Parser löst die folgenden Ereignisse aus:

- ✓ Anfang des Elements `<ZITAT>`
- ✓ Inhalt des Elements `"Sein oder..."`
- ✓ Ende des Elements `</ZITAT>`

Wenn ein Ereignis eintritt, wird die entsprechende Methode, die das Ereignis verarbeitet im Programm (welches den SAX-Parser nutzt) aufgerufen. Diese Methode wird auch als EventHandler bezeichnet. Beispielsweise könnte beim Auffinden des Elements `ZITAT` der nachfolgende Text in roter und kursiver Schriftformatierung auf dem Bildschirm angezeigt werden.

Der Zugriff auf die XML-Daten wird überwiegend über das Document Object Model (DOM) realisiert. Da die Struktur des XML-Dokuments als Dokumentenbaum im Speicher gehalten wird, können Sie jederzeit auf die einzelnen Elemente zugreifen. SAX wird vorwiegend zum einmaligen Einlesen eines XML-Dokuments genutzt, ist dafür schnell und benötigt keinen Speicher für die Repräsentation des XML-Dokuments.



15.5 XML-Datenblöcke in HTML5

Befindet sich XML-Quelltext innerhalb einer HTML-Seite, wird dies als Dateninsel (engl. Data Island) bezeichnet. Eine Dateninsel ermöglichte Ihnen bisher, XML direkt in HTML zu integrieren, ohne die entsprechenden Daten durch ein Skript oder eine XSL-Anweisung laden zu müssen. Der Nachteil war bisher, dass Dateninseln nur vom Internet Explorer unterstützt wurden.

Mit Einführung von HTML5 ist es möglich, sogenannte Datenblöcke in HTML-Code unterzubringen, wie z. B. XML.



Interner Datenblock

Ein XML-Datenblock wird über das Tag `script` und die Angabe, dass es sich um XML-Daten handelt, eingeleitet.

```
<script id="" type="application/xml">
  XML Daten
</script>
```

Über das Attribut `id` weisen Sie dem Datenblock eine eindeutige Kennung zu, mit der Sie später auf die Daten zugreifen können. Das Attribut `type` bestimmt, dass es sich nachfolgend um XML-Daten handelt. Die XML-Daten geben Sie direkt zwischen den `script`-Tags an.

Innerhalb eines HTML-Dokuments können Sie auch mehrere XML-Datenblöcke einbinden. Dabei ist zu beachten, dass jeder Datenblock eine eindeutige Kennung `id` besitzt, mit der die jeweiligen Daten angesprochen werden können.



Beispiel: *kap15\datenblock1.html*

Im dem HTML-Dokument werden die bisherigen XML-Daten als Datenblock eingebunden.

```
① <!DOCTYPE html>
   <html>
   <head>
   <title>XML Datenblock</title>
   <script type="text/javascript" src="loadxmldoc.js"></script>
   <link type="text/css" rel="stylesheet" href="musik_link.css">
   ② <script id="musiksammlung" type="application/xml">
   ③ <!DOCTYPE MUSIKSAMMLUNG SYSTEM "musikstil.dtd" [
     <!ENTITY MRL "Mute Records Limited">
     <!ENTITY IRL "Island Records Limited">
     <!ENTITY JR "Jarrett Records">
     <!ENTITY WR "WEA Records">
     <!ENTITY PCD "Pilz Compact Disc">
     <!ENTITY hs "Heiko Schröder">
     <!ENTITY mm "Max Mustermann">
   ]>
   <MUSIKSAMMLUNG>
```

```

④ <ALBUM>
    <!-- alle weiteren XML-Elemente -->
  </ALBUM>
</MUSIKSAMMLUNG>
</script>
</head>

```

- ① Der Inhalt der HTML-Datei wird über die Dokumenttyp-Definition als HTML5 definiert.
- ② Der Datenblock wird über `script` und die Angabe von `application/xml` als XML definiert. Zur Identifikation erhält der Datenblock die eindeutige Kennung `musiksammlung`.
- ③ Hier geben Sie den Inhalt der bisher eingebundenen Datei *musiksammlung.xml* inklusive der Entitäten-Definition an.
- ④ Mit `</script>` schließen Sie den XML-Datenblock.

Die XML-Daten werden zwar in den Browser geladen, jedoch nicht angezeigt.

XML-Daten auslesen

Mithilfe von JavaScript können Sie die Informationen des Datenblocks auslesen und auswerten. Dabei wird der Datenblock als Objekt über die festgelegte Kennung `id` angesprochen.

Um auf die Daten zugreifen zu können, müssen Sie den Inhalt des XML-Datenblocks über dessen ID ansprechen und den Inhalt einer Variable übergeben.

```

xmlSource = document.getElementById("id").textContent;
parser = new DOMParser();
Objekt = parser.parseFromString(xmlSource, "application/xml");

```

Die Browser stellen über `DOMParser()` eine API zur Verfügung, mit der eine Zeichenkette anhand einer festgelegten Spezifikation geparkt und das Ergebnis an ein Objekt übergeben werden kann.

Beispiel: *kap15datenblock2.html*

Fügen Sie in die HTML-Datei *datenblock1.html* die nachfolgende, bereits bekannte JavaScript-Anweisung zum Auflisten der Interpreten, Alben und Bemerkungen ein.

```

<!-- ... -->
<h2>Musiksammlung</h2>
<p>Datenumfang: <span id="counter"></span> Einträge</p>
<div id="tabelle"></div>

<script type="text/javascript">
①  xmlSource = document.getElementById("musiksammlung").textContent;
②  parser = new DOMParser();
③  XMLObjekt = parser.parseFromString(xmlSource, "application/xml");
④  Album = XMLObjekt.getElementsByTagName('ALBUM');
    document.getElementById('counter').innerHTML = Album.length;
    output = '<table><tr class="firstline">';
    output = output + '<td><b>Interpret</b></td>'
                                     + '<td><b>Album</b></td><td><b>Bemerkung</b></td></tr>'

⑤  for (i=0; i < Album.length; i++) {
⑥    output = output + '<tr>'; output = output + '<td>';
      Interpret = Album[i].getElementsByTagName('INTERPRET');
      if (Interpret.length == 0) {
        Interpret = Album[i].getElementsByTagName('GRUPPE');
      }
      output = output + getNodeText(Interpret[0]);
      output = output + '</td>';

```

```

        output = output + '<td>';
        Titel      = Album[i].getElementsByTagName('TITEL');
        output = output + getNodeText(Titel[0]);
        output = output + '</td>';

        output = output + '<td>';
        Lieder = Album[i].getElementsByTagName('LIED');
        output = output + 'Lieder: ' + Lieder.length;
        output = output + '</td>';

        output = output + '</tr>';
    };
    output = output + '</table>';
    document.getElementById('tabelle').innerHTML = output;
</script>
<!-- ... -->

```

- ① Der Inhalt des Elements mit der ID `musiksammlung` wird der Variablen `xmlSource` übergeben.
- ② Das Objekt `parser` vom Typ `DOMParser` wird erstellt.
- ③ Über dieses Objekt wird mithilfe der Methode `parseFromString` der Inhalt der Variablen `xmlSource` geparkt. Der zu erwartende Inhalt ist vom Typ XML. Das Ergebnis und somit das Document Object Model wird in der Variablen `XMLObject` abgelegt.
- ④ Die Anzahl der gefundenen Elemente `ALBUM` aus der Variablen `XMLObject` wird am HTML-Tag mit der ID `counter` ausgegeben.
- ⑤ In Abhängigkeit von der Albenanzahl werden die entsprechenden XML-Daten in der Variablen `output` gespeichert.
- ⑥ Am Ende wird der Inhalt dieser Variablen als HTML-Text dem Tag mit der ID `tabelle` übergeben.

Dieses Beispiel lässt sich beliebig erweitern. So könnte auch in dieser HTML-Seite eine Informationsabfrage des jeweiligen Albums nach dem Betätigen des entsprechenden Hyperlinks realisiert werden.

Externen Datenblock einbinden

Das Einbinden externer Datenblöcke wird derzeit nur vom Internet Explorer und dem Mozilla Firefox unterstützt. Die anderen Browser werten den Inhalt des `object`-Tags falsch aus.



Sind die Daten der XML-Datei sehr umfangreich, können Sie die Daten wie bisher auch in einer separaten XML-Datei definieren und dann im HTML-Quelltext einbinden. Hierfür können Sie das HTML-Tag `object` verwenden.

```
<object id="ID" data="XML-Datei" type="text/xml" style="display:none;"></object>
```

Über den Parameter `id` definieren Sie die eindeutige Kennzeichnung und über `data` legen Sie die einzulesende XML-Datei fest. Mit `type="text/xml"` stellen Sie sicher, dass der Inhalt auch als XML angesehen wird. Die Stylesheet-Angabe `style="display:none"` bewirkt, dass das Objekt nicht im Browser angezeigt wird.

Den Inhalt der XML-Datei laden Sie über die Angabe der ID und der Methode `contentDocument`.

```
XMLObject = document.getElementById("id").contentDocument;
```

Beispiel: kap15\datenblock3.html

```

<!DOCTYPE html>
<html>
<head>
<title>XML Datenblock</title>
<script type="text/javascript" src="loadxmldoc.js"></script>
<link type="text/css" rel="stylesheet" href="musik_link.css"></link>

```

```

<script type="text/javascript">
function loadData() {
    var XMLObjekt = document.getElementById("musiksammlung").contentDocument;

    var Album = XMLObjekt.getElementsByTagName('ALBUM');
    /* ... */
}
</script>
</head>
<body onLoad="loadData();" >
    <h2>Musiksammlung</h2>
    <p>Datenumfang: <span id="counter"></span> Einträge</p>
    <div id="tabelle"></div>
    <object id="musiksammlung" data="musiksammlung.xml" type="text/xml"
        style="display: none;"></object>

</body>
</html>

```

15.6 Übungen

Übung 1: Interner Datenblock

Übungsdatei: *kap15\ms.xml,*
kap15\loadxmldoc.js

Ergebnisdateien: *kap15\uebung1.html,*
kap15\uebung2.html

- ① Erweitern Sie das Beispiel des eingebundenen internen Datenblocks. Erstellen Sie einen Hyperlink zum Namen des Interpreten. Beim Auslösen des Verweises sollen alle Alben des Interpreten in einem neuen Fenster aufgelistet werden. Dazu muss der Interpret mit mehreren Alben im XML-Dokument vorhanden sein.
- ② Realisieren Sie das Beispiel mithilfe eines externen Datenblocks für die Browser Firefox und Internet Explorer.

Übung 2: Daten separieren

Übungsdatei: *kap15\kfz.xml*

Ergebnisdatei: *kap15\uebung3.html*

Erstellen Sie eine Übersicht einer Fahrzeugverwaltung mit der Ausgabe der Hersteller und des Fahrzeugmodells. Über einen entsprechenden Link sollen die technischen Daten in einem neuen Fenster eingeblendet werden.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

16 XHTML 1.1

In diesem Kapitel erfahren Sie

- ✓ wie sich XHTML vom bisherigen Standard HTML unterscheidet
- ✓ wie Sie Ihre bisherigen HTML-Dokumente unter Beachtung des XHTML-Standard anpassen

Voraussetzungen

- ✓ HTML-Kenntnisse

16.1 Überblick zu XHTML

Ansatz

Ausgehend von der Meta-Auszeichnungssprache **SGML** entstand **HTML** als SGML-Applikation. Mit der fortschreitenden Verbreitung von HTML erwies es sich als problematisch, dass von den Herstellern eine Reihe unterschiedlicher browserinterner HTML-Tags unterstützt wurde. Ein Beispiel ist das für den Internet Explorer typische Tag `marquee`. Dieser ermöglicht einen Lauftext, ähnlich einem Nachrichtenticker. Dieses Element entspricht nicht dem HTML-Standard des W3-Konsortiums, wird aber aus Gründen der Kompatibilität auch von anderen Browsern interpretiert. Die Browser Firefox und Opera interpretieren beispielsweise wiederum das Element `blink`, andere Browser nicht.

Ein weiteres Problem von HTML ist der notwendige großzügige Umgang der Browser mit fehlerhaften Seiten. Um diese verarbeiten und darstellen zu können, müssen Browser entsprechende Routinen enthalten. Dies führt zu sinkenden Ausführungszeiten und zu einer unnötigen Vergrößerung der Programme.

XHTML ist eine auf XML basierende Empfehlung zur Weiterentwicklung von HTML zur Behebung dieser Probleme. Die wichtigste Änderung gegenüber HTML ist, dass die XHTML-Quelldateien entsprechend den XML-Regeln wohlgeformt sein müssen, nach den XHTML-Regeln geschriebene HTML-Dokumente also keine Fehler in der Struktur der HTML-Tags enthalten dürfen. Dies ermöglicht es den Entwicklern, die Ausführungsgeschwindigkeit von Browsern zu erhöhen, da in den Programmen keine Routinen mehr zur Verarbeitung von fehlerhaftem Code enthalten sein müssen. Des Weiteren kann XHTML-Code über eine standardisierte Dokumenttyp-Definition geprüft werden. Damit verringert sich die Größe der Programme.

Nicht regelkonforme Webseiten zeigen deshalb fehlerhafte Dokumente nicht mehr an. Stattdessen erscheint eine Fehlermeldung, ähnlich den Fehlermeldungen beim Ausführen von fehlerhaftem JavaScript.

Entwicklung

Das W3-Konsortium hatte mit der HTML-Version 4.01 vom 24. Dezember 1999 die weitere Standardisierung der Sprache HTML bis ins Jahr 2008 eingestellt. Die Sprache XHTML wurde als neuer Standard für Webseiten entwickelt. Ziel dieser Entwicklung war es, den Inhalt, die Struktur und die Formatierung eines Dokuments voneinander zu trennen und die Vorteile von HTML 4.0 und XML 1.0 zu vereinen.

In der Empfehlung XHTML 1.0 (**Extensible Hypertext Markup Language**) aus dem Jahr 2000 (siehe <http://www.w3.org/TR/xhtml1/>) wurde auf eine Einhaltung der Rückwärtskompatibilität zu den verbreiteten HTML-Browsern geachtet, wobei neuere Browser die Dokumente entsprechend der strengeren Regeln von XHTML verarbeiten konnten. XHTML 1.0 ist im Prinzip eine Neuformulierung von HTML 4.01 unter Berücksichtigung der Einhaltung der XML-Regeln. Die vorhandenen Dokumenttypen Strict, Transitional und Frameset werden unterstützt.

Der zweite Entwicklungsschritt XHTML 1.1 aus dem Jahr 2010 (siehe <http://www.w3.org/TR/xhtml11/>) schränkte die ursprünglich erlaubten Möglichkeiten von HTML stärker ein. Der Umfang der Sprache wurde im Wesentlichen auf die Unterstützung des Dokumenttyps Strict reduziert. Elemente und Attribute der Dokumenttypen Transitional und Frameset, welche die Darstellung der Seite direkt steuern, wurden entfernt. Die fehlende Kompatibilität zu bestehenden Browserprogrammen wurde dabei bewusst akzeptiert. Veraltete Elemente und Attribute, die bisher als „deprecated“ geduldet wurden, dürfen nicht mehr verwendet werden. XHTML wurde modularisiert, indem die HTML-Tags in Modulen zusammengefasst werden (z. B. Text-, Formular-, Tabellen-, Grafik-, Objektmodul, siehe <http://www.w3.org/TR/xhtml-modularization>). Weitere wichtige Änderungen sind:

- ✓ Die Attribut-Angabe `lang` entfällt an einzelnen Elementen und wird zentral als Attribut `xml:lang` festgelegt.
- ✓ Die Attributangabe von `name` bei den Elementen `a` und `map` entfällt und ist durch das Attribut `id` zu ersetzen. Ankerpunkte sind z. B. als `` zu setzen.
- ✓ Hinzugekommen ist das Modul `ruby` mit verschiedenen neuen Elementen (`rb`, `rtc`, `rb`, `rt`, `rp`). Damit können kurze Anmerkungen oder die Aussprache von Texten hinzugefügt werden. Diese Art der Darstellung ist vor allen Dingen aus dem ostasiatischen Raum bekannt. Somit kann z. B. neben einer Abkürzung der vollständige Begriff angegeben werden (siehe nebenstehende Abbildung).



Scalable Vector Graphics
SVG

HTML5 und XHTML5

Lange Zeit wurde seitens des W3-Konsortiums an einer Spezifikation für XHTML 2.0 gearbeitet. Im Dezember 2010 wurde die entsprechende Arbeitsgruppe geschlossen und die Entwicklung von XHTML 2.0 zugunsten des neuen HTML5-Standards eingestellt. Eine Ursache für diese Entwicklung war die zum Teil fehlende Akzeptanz der neuen strengerer Regeln der Webseitengestalter und -programmierer. Der für XHTML 2.0 geplante Bruch mit dem Erbe von HTML und die damit einhergehenden grundlegenden Änderungen der Webseitengestaltung waren nicht mehrheitsfähig. Stattdessen wurde eine schrittweise Verbesserung von HTML unter Beibehaltung der bisherigen Grundprinzipien und Regeln als neues Ziel bestimmt, welches mit der Entwicklung von HTML5 konsequent verfolgt wird.

HTML5 ist jedoch keine komplette Abkehr von den mit XHTML angestrebten Zielen. Die bei der Entwicklung von HTML5 federführende WHATWG (**Web Hypertext Application Technology Working Group**), ein Zusammenschluss führender Browserhersteller (siehe <https://whatwg.org/>), sieht in ihrer Einordnung HTML5 als abstrakte Sprache, welche unterschiedliche Syntaxvarianten enthalten kann – sowohl HTML als auch XHTML. Je nach verwendetem Typ wird es vom Browser als HTML-Dokument verarbeitet oder als XML Dokument geparkt. Wird ein HTML5-Dokument mittels XML-Syntax und nach den Regeln von XHTML dargestellt, wird dafür die Bezeichnung XHTML5 verwendet, ohne dass es sich dabei um eine definierte Empfehlung wie bei den Versionen XHTML 1.0 oder 1.1 handelt.

XHTML hatte bisher und wird damit auch weiterhin eine gewisse Bedeutung bei der Gestaltung von Webseiten besitzen. Es obliegt letztendlich dem Entwickler, welche Ausprägung der Syntax er nutzt. Die folgenden Abschnitte fassen die Unterschiede von HTML und XHTML zusammen und zeigen XHTML als eine mögliche Anwendung von XML.

16.2 XHTML-Dokument deklarieren

Dokumenttyp und Zeichensätze

Die Anzahl möglicher Sprachversionen, die ein Browser beim Lesen einer HTML-Datei beherrschen muss, wird immer größer. Deshalb ist es sinnvoll, wenn Sie die HTML-Version, die Sie in Ihren Dateien verwenden, an den Anfang einer jeden HTML-Datei schreiben. Die Versionsangabe ist ein Hinweis auf eine bestimmte HTML-Sprachversion.

Die Angabe von

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

besagt zum Beispiel, dass sich die nachfolgenden Zeilen des HTML-Dokuments genau an die HTML-4.01-Definition des W3-Konsortiums halten. Diese Angabe hatte keine Konsequenzen für die Darstellung der HTML-Datei und wurde deshalb von den Autoren einer Webseite vernachlässigt. Das ändert sich jedoch mit der Verwendung von XHTML.

Um festzulegen, dass es sich im Nachfolgenden um eine Datei nach dem Standard XHTML 1.1 handelt, ist die folgende Dokumenttyp-Definition anzugeben:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Mit der Angabe dieses Dokumenttyps weisen Sie darauf hin, dass Ihr Dokument den strengen XHTML-Regeln der Version 1.1 genügt. Die in der Version 1.0 noch geduldeten veralteten Angaben werden entweder durch entsprechende Stylesheet-Angaben ersetzt oder alternativ über die Standardeinstellungen des Clients formatiert. Aus HTML bekannte, aber nicht mehr empfohlene Elemente werden nicht verwendet.

Weiterhin müssen Sie in einer HTML-Datei den im Dokument verwendeten Zeichensatz spezifizieren. Diese Spezifikation muss ebenfalls in XHTML erfolgen.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

In diesem Fall wird darauf hingewiesen, dass innerhalb des Dokuments Zeichen des westeuropäischen Zeichensatzes ISO-8859-1 verwendet werden.

Grundstruktur eines XHTML-Dokuments

Folgende Angaben sind innerhalb eines Dokuments vorzunehmen, um es eindeutig als XHTML-Dokument zu deklarieren.

- ✓ Der Dokumenttyp ist anzugeben.
- ✓ Das Hauptelement ist wie bisher das Element `html`.
- ✓ Das Hauptelement muss eine `xmlns`-Deklaration für XHTML-Namensräume enthalten. Der Namensraum für XHTML ist definiert als `http://www.w3.org/1999/xhtml`.

Das Grundgerüst eines XHTML-Dokuments sieht z. B. folgendermaßen aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title> </title>
  </head>
  <body>
    </body>
</html>
```


16.3 Elemente in XHTML

Groß- und Kleinschreibung

Entsprechend der Regelungen von XML ist die einheitliche Groß- bzw. Kleinschreibung der Tags zu beachten. In XHTML werden alle HTML-Tags kleingeschrieben.

Das folgende Beispiel ist in HTML korrekt, in XHTML jedoch falsch:

```
<H1 Align="center">nachfolgender Text</h1>
```

In XHTML müsste dieses Beispiel so geschrieben werden:

```
<h1 align="center">nachfolgender Text</h1>
```

Öffnende und schließende Tags

Alle geöffneten Tags müssen in XML auch ein schließendes Tag besitzen. Diese Regel wurde ebenfalls in XHTML übernommen. In HTML können Sie an vielen Stellen das schließende Tag weglassen, ohne dass der Browser eine Fehlermeldung einblendet.

```
<ol>
  <li>Dies ist ein Listenelement
  <li>Dies ist ein weiteres Listenelement
</ol>
```

In XHTML müssen Sie das schließende Tag `` hinzufügen.

```
<ol>
  <li>Dies ist ein Listenelement</li>
  <li>Dies ist ein weiteres Listenelement</li>
</ol>
```

Jedes Start-Tag müssen Sie auch wieder mit dem entsprechenden Ende-Tag schließen. Ein Spezialfall sind die allein stehenden Elemente: `area`, `base`, `br`, `col`, `frame`, `hr`, `img`, `input`, `isindex`, `link`, `meta`, `option` und `param`. Um mit älteren Browsern kompatibel zu bleiben, müssen Sie vor dem Zeichen `>` ein Leerzeichen und einen Schrägstrich `/` schreiben.

```
<br />
<hr />

```

Sie können diese Elemente jedoch auch als öffnendes und schließendes Tag schreiben. Es ist daher nicht falsch, wenn Sie statt der möglichen Kurzform `
` die ausführliche Schreibweise `
</br>` benutzen.

Verschachtelung von HTML-Tags

Die Reihenfolge der Elemente muss logisch gegliedert sein. Einzelne Elemente dürfen sich in XML nicht überlappen. Das folgende Beispiel ist daher in XHTML falsch:

```
<b><i>fetter kursiver Text</b></i>
<p>Ein Text mit <em>einer logischen Textauszeichnung.</p></em>
```

Die korrekte Schreibweise in XHTML lautet:

```
<b><i>fetter kursiver Text</i></b>
<p>Ein Text mit <em>einer logischen Textauszeichnung.</em></p>
```

Blockelemente, wie `h1` ... `h6`, `p` oder auch `hr`, erzeugen im Dokument stets einen Zeilenvorschub. Inline-Elemente tun dies nicht, sie enthalten die Textdaten und weitere Inline-Elemente. Sie erzwingen keinen Zeilenvorschub und werden verwendet, um Textelemente besonders zu kennzeichnen.

Da die Inline-Elemente innerhalb von Blockelementen verwendet werden, sind sie den Blockelementen untergeordnet. Es ist daher nicht erlaubt, dass ein Inline-Element ein Blockelement umschließt.

```
<span><p>lange Textpassagen oder einzelne Worte</p></span>
```

Das Element `p` ist dem Element `span` übergeordnet, da `p` ein Blockelement und `span` ein Inline-Element ist. Somit darf `span` nicht das Element `p` umschließen, sondern `p` muss `span` umschließen.

```
<p><span>lange Textpassagen oder einzelne Worte</span></p>
```

Kombinationen

Einige Kombinationen von HTML-Tags sind verboten. So darf beispielsweise innerhalb der physischen Auszeichnung `pre` in XHTML kein Bild `img` angegeben werden.

Folgende Elemente dürfen die anderen angegebenen Elemente nicht enthalten:

Das Element	darf das Element ... nicht enthalten
<code>a</code>	<code>a</code>
<code>pre</code>	<code>img</code> , <code>object</code> , <code>big</code> , <code>small</code> , <code>sub</code> , <code>sup</code>
<code>button</code>	<code>input</code> , <code>select</code> , <code>textarea</code> , <code>label</code> , <code>button</code> , <code>form</code> , <code>fieldset</code> , <code>iframe</code> , <code>isindex</code>
<code>label</code>	<code>label</code>
<code>form</code>	<code>form</code>

16.4 Attribute und Werte in XHTML

Anführungszeichen

In XHTML müssen alle Attribute auch einen Wert besitzen. Die Werte sind immer in Anführungszeichen zu setzen, auch wenn sie nur aus einer Zahl oder einer Prozentangabe bestehen. Beispielsweise ist

```
<td valign=top>
  <img alt="alternativer Text" width=300% height=25 />
</td>
```

keine gültige XHTML-Angabe. Die Anführungszeichen müssen in diesem Fall angegeben werden.

```
<td valign="top">
  <img alt="alternativer Text" width="300%" height="25" />
</td>
```

Kurzschreibweise

Die Kurzschreibweise von Attributen ist in XHTML erlaubt. Folgende Attribute sind von dieser Regelung betroffen: `compact`, `nowrap`, `ismap`, `declare`, `noshade`, `checked`, `disabled`, `readonly`, `multiple`, `selected`, `noresize`, `defer`.

```
<input name="text" rows="20" cols="50" readonly />
<select name="auswahl" size="5" multiple>
  <option>Auswahl 1</option>
  <option>Auswahl 2</option>
  <option>Auswahl 3</option>
</select>
```

Stattdessen ist der Name des Attributs noch einmal als Wert in Anführungszeichen anzugeben.

```
<input name="text" rows="20" cols="50" readonly="readonly" />
<select name="auswahl" size="5" multiple="multiple">
  <option>Auswahl 1</option>
  <option>Auswahl 2</option>
  <option>Auswahl 3</option>
</select>
```

Eine weitere Änderung betrifft die Tags `a`, `applet`, `form`, `frame`, `iframe`, `img` und `map`. In HTML werden sie über die Attribute `name` oder `id` eindeutig gekennzeichnet, in XHTML werden sie nur noch über das Attribut `id` referenziert. Es ist aus Gründen der Kompatibilität zu älteren Browsern noch möglich, zusätzlich das Attribut `name` zu belassen. Aus der Schreibweise

```
<a name="marke">Stelle, die angesprungen werden soll.</a>
```

wird

```
<a name="marke" id="marke">Stelle, die angesprungen werden soll.</a>
```

16.5 JavaScript und Stylesheets in XHTML

In HTML können JavaScript- und Stylesheet-Anweisungen im Dokument angegeben werden. Diese sind vom Typ `PCDATA` und können somit eine beliebige Zeichenfolge enthalten. Damit müssen die Daten vor der Verwendung überprüft werden. So wird sichergestellt, dass die Angaben beim Seitenaufbau korrekt verwendet werden können. In XHTML müssen die internen Stylesheet-Angaben abgewandelt werden, da sie nicht vom Programm überprüft werden sollen.

Aus der internen Stylesheet-Angabe

```
<style>
  .Klasse { <!-- Angabe der Stylesheets --> }
</style>
```

wird in der XHTML-1.1-Spezifikation:

```
<style type="text/css">
<![CDATA[
  .Klasse { <!--Angabe der Stylesheets --> }
]]>
</style>
```

Die einzelnen Stylesheet-Klassen werden durch einen `CDATA`-Block `<![CDATA[...]]>` umschlossen. Damit werden die Angaben nicht vor der Verwendung überprüft. Dies ist wichtig, da die Stylesheet-Angabe keine XML-Elemente enthält, sondern die Formatierungsdefinitionen für das Dokument.

Ähnlich verhält es sich mit der internen Angabe von JavaScript-Anweisungen. Auch hier muss der bisherige JavaScript-Programmabschnitt

```
<script type="text/javascript">
    // JavaScript-Anweisungen
</script>
```

abgewandelt werden, damit diese nicht als XML-Anweisungen ausgewertet werden.

```
<script type="text/javascript">
<![CDATA[
    // JavaScript-Anweisungen
]]>
</script>
```

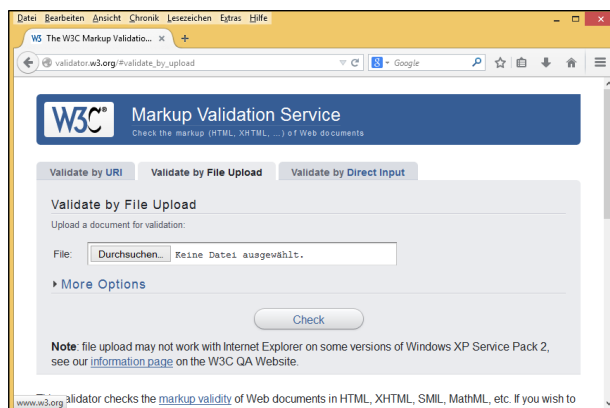


Eine Alternative ist es, externe JavaScript- und Stylesheet-Dokumente zu verwenden. An der Einbindung dieser Daten ändert sich gegenüber HTML 4.0 nichts.

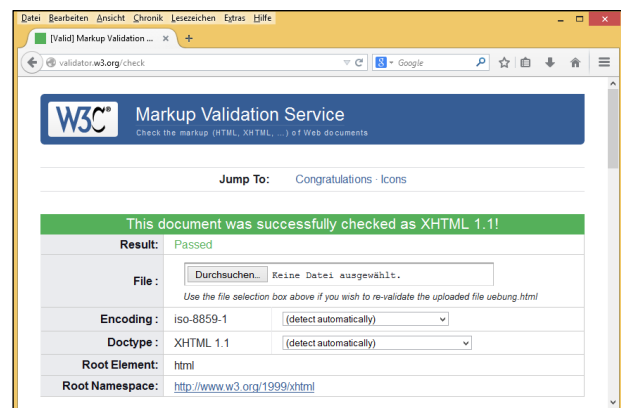
16.6 W3C MarkUp Validation Service

Ob eine HTML-Datei den Anforderungen des XHTML-Formats entspricht, können Sie mit dem Markup Validation Service des W3-Konsortiums testen.

- Öffnen Sie in Ihrem Browser die Internetadresse <http://validator.w3.org>.



Webseite zum Testen eines XHTML-Dokuments



Erfolgreich getestetes XHTML-Dokument

- Wechseln Sie in das Register *Validate By File Upload*, um die XHTML-Datei auf den Server zu übertragen.
- Klicken Sie neben dem Eingabefeld *File* auf die Schaltfläche *Durchsuchen*.
- Wählen Sie Ihre XHTML-Datei aus.
- Betätigen Sie die Schaltfläche *Check*.

Ihr Dokument wird auf den Server des W3-Konsortiums übertragen. Dort wird es mithilfe eines Skripts auf die korrekte Verwendung von XHTML-Elementen und Attributen getestet. Verstößt Ihr Dokument gegen keine XHTML-Regel, erscheint die Ausgabe: *This document was successfully checked as XHTML1.1!* In diesem Fall können Sie das offizielle Zeichen des W3-Konsortiums auf Ihrer Webseite integrieren.

Damit zeigen Sie Ihren Besuchern, dass Ihre Webseite XHTML-1.1-kompatibel ist.



16.7 Übung

HTML-Datei in XHTML-Datei umwandeln

Übungsdatei: *kap16\vorgabe.html*

Ergebnisdatei: *kap16\uebung.html*

- ① Konvertieren Sie die nachfolgende HTML-Datei in das XHTML-Format. Welche HTML-Tags sind nach den XHTML-Richtlinien falsch?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>HTML nach XHTML</title>
    <style type="text/css">
      h2 { font-family: Verdana, Arial, sans-serif; font-size: 14pt;
        color: #CC0133; font-weight: bold }
      p, li { font-family: Georgia, 'Times New Roman', Times, serif;
        font-size: 10pt; color: #000080 }
    </style>
  </head>
  <body>
    <H2>HTML nach XHTML</h2>
    <p>Dieses HTML-Dokument soll in das XHTML-Format konvertiert werden.
    <ul>
      <Li>Listeneintrag 1
      <Li>Listeneintrag 2
      <Li>Listeneintrag 3
    </ul>
    <form method="POST" action="mailto:">
      <p><input type="text" name="Eingabe" size=20 readonly>
      <p><textarea rows=2 name="Memo" cols=20 readonly></textarea>
      <p><input type="submit" value="Abschicken">
        <input type="reset" value="Zurücksetzen">
      </form>
    </body>
  </html>
```

- ② Kontrollieren Sie mithilfe des HTML Validation Service des W3-Konsortiums die Korrektheit Ihres XHTML-Dokuments. Nehmen Sie Änderungen an dem XHTML-Code vor, bis Sie das offizielle XHTML-Zeichen des W3-Konsortiums erhalten.

17 SVG

In diesem Kapitel erfahren Sie

- ✓ was SVG ist
- ✓ wie Sie eine SVG-Grafik erzeugen können
- ✓ wie Sie eine SVG-Grafik in eine HTML5-Webseite integrieren können

Voraussetzungen

- ✓ Aufbau von XML-Dokumenten
- ✓ HTML-Kenntnisse

17.1 SVG-Grundlagen

SVG (**S**calable **V**ector **G**raphics) ist eine XML-Anwendung zur Darstellung von Vektorgrafiken. Sie ist eine der wichtigsten Formen zur Darstellung von grafischen Inhalten im Web und auf Mobilgeräten. Der Vorteil der Verwendung von einer SVG-Grafik im Vergleich zu einer Datei im Bildformat besteht darin, dass SVG lediglich aus textlichen Informationen – den XML-Daten – besteht, was sich positiv auf die Datenmenge und damit auf die Übertragungsgeschwindigkeit auswirkt.

Viele der gängigen Webbrowser sind in der Lage, ohne die zusätzliche Installation einer Erweiterung (eines Plug-In) SVG-Grafiken in nahezu vollem Sprachumfang darzustellen. Bei älteren Browsern, beispielsweise dem Internet Explorer 8 und früher, ist die Installation eines Plug-in zur Darstellung von SVG-Grafiken erforderlich.

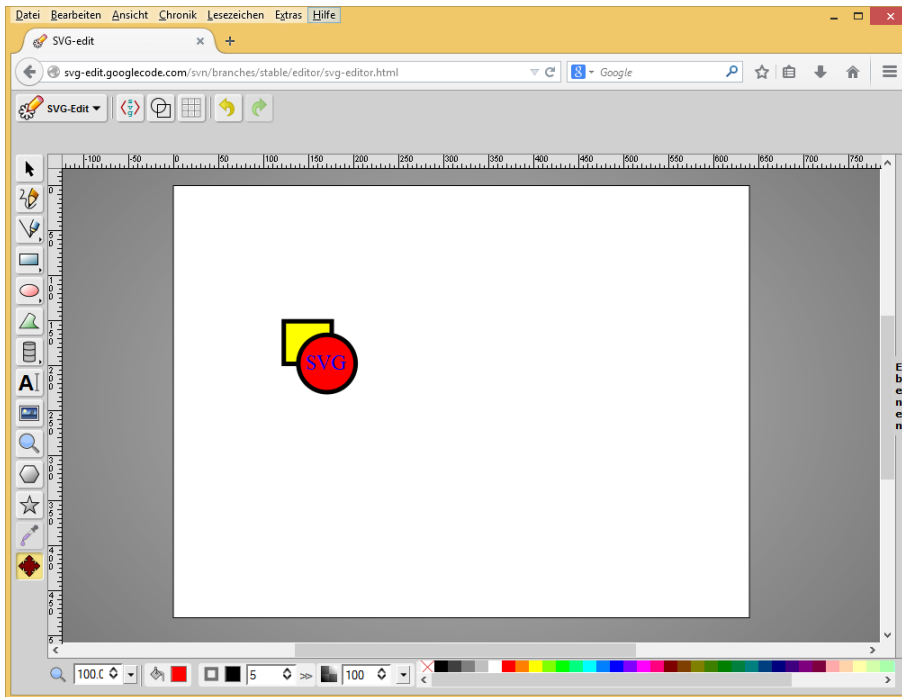
SVG basiert auf einer DTD (<http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd>). Das zeigt, dass diese Technik trotz der Einführung von XML-Schema sehr leistungsfähig ist. Die Elemente der DTD ermöglichen alle typischen Elemente einer Grafik – von einzelnen zweidimensionalen geometrischen Objekten wie Kreisen und Rechtecken, über Texte bis hin zu Spezialeffekten wie Farbverläufen und Animationen. SVG entspricht dem Objektmodell und ermöglicht so die dynamische Manipulation der vorhandenen Objekte über Skripte.

Die erste SVG-Spezifikation des W3C stammt bereits aus dem Jahr 2001. Die aktuelle Empfehlung 1.2 wurde 2011 veröffentlicht. Mit der Aktualisierung wurde eine Modularisierung eingeführt. Damit existieren nun spezielle Teile für mobile (SVG mobile) und für plattformübergreifende (SVG tiny) Anwendungen. Die Informationen des W3C finden Sie unter dem Link <http://www.w3.org/Graphics/SVG/> über die Einstiegsseite zum Thema SVG.

17.2 Erzeugung von SVG-Grafiken

SVG-Editoren

Zur Erstellung von SVG-Grafiken existieren viele Programme. Ein oft genutztes Werkzeug ist beispielsweise das Open Source Programm Inkspace (<https://inkscape.org/de/>). Webeditoren, wie SVG-Edit (<http://svg-edit.googlecode.com/svn/branches/stable/editor/svg-editor.html>), bieten ebenfalls umfangreiche Möglichkeiten zur Grafikerstellung. Des Weiteren unterstützen viele Programme, zum Beispiel Microsoft Visio, den Export von Zeichnungen im SVG-Format.



Der Editor SVG-Edit

SVG-Tags

SVG-Dateien bestehen analog zu HTML aus einer Reihe definierter Tags. Einige wichtige sind:

Element	Beschreibung
<a>	erstellt einen Link um ein SVG-Element
<animate>	definiert die Änderung eines Attributs innerhalb einer bestimmten Zeit
<animateColor>	definiert eine Farbänderung
<animateMotion>	bewirkt die Bewegung eines referenzierten Elements entlang eines Pfades
<animateTransform>	transformiert ein Attribut eines Elementes (z.B. hinsichtlich Größe und Rotation)
<circle>	definiert einen Kreis
<ellipse>	definiert eine Ellipse
font	definiert einen Zeichensatz
font-face	beschreibt einen Zeichensatz
<g>	gruppiert Elemente
<image>	definiert ein Bild
<line>	definiert eine Linie
metadata	beschreibt die Metadaten der Abbildung
<rect>	definiert ein Rechteck
script	dient als Container für Skripte (z.B. ECMAScript)
set	setzt den Wert eines Attributs für eine bestimmte Zeitdauer
style	ermöglicht die direkte Einbettung von Stylesheets in SVG
<svg>	umschließt das SVG-Dokument
<text>	definiert einen Text
<title>	beschreibt eine SVG-Grafik oder ein -Element, wird nicht angezeigt, kann jedoch als Tooltip (Mouse-Over-Effekt) angezeigt werden

Die SVG-Referenz finden Sie auf der Webseite http://www.w3schools.com/svg/svg_reference.asp.



Beispiel *kap17/svg_demo.xml*

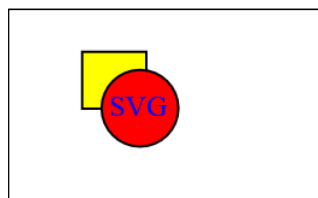
Das Beispiel stellt in einer einfachen Grafik drei Elemente dar – ein Rechteck, einen Kreis und einen Text. Der Quellcode wurde automatisch durch das Zeichnen der Abbildung im Webeditor SVG-Edit und der anschließenden Speicherung erzeugt. Einige Attribute der Elemente mit dem Wert `null` wurden hier aus Gründen der Übersichtlichkeit entfernt.

```

① <?xml version="1.0"?>
② <svg width="640" height="480"
  xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/svg">
  <!-- Created with SVG-edit - http://svg-edit.googlecode.com/ -->
③ <g>
④ <title>Layer 1</title>
⑤ <rect id="svg_1" height="47.25" width="53.250005" y="150.750008" x="122.750038"
  stroke-width="2" stroke="#000000" fill="#ffff00"/>
⑥ <circle id="svg_2" r="31.878906" cy="197.625011" cx="170.750022"
  stroke-width="2" stroke="#000000" fill="#ff0000"/>
⑦ <text xml:space="preserve" text-anchor="middle" font-family="serif"
  font-size="24" id="svg_3" y="204.000015" x="169.250053" stroke-width="0"
⑧ stroke="#000000" fill="#0000ff">SVG</text>
  </g>
</svg>

```

- ① Die Größe der SVG-Grafik wird über die Attribute `width` und `height` im die Grafik umschließenden `<svg>` Tag festgelegt.
- ② Der SVG-Namensraum wird angegeben.
- ③ Die Elemente werden in einer Gruppe zusammengefasst.
- ④ Der Titel `Layer 1` wird der Gruppe zugewiesen.
- ⑤ Das Rechteck wird definiert. Um es referenzieren zu können, erhält es eine eindeutige ID. Als Attribute werden die Höhe und Breite, die Position, die Linienstärke und -farbe sowie die Füllfarbe angegeben.
- ⑥ Die Definition des Kreises mit den Eigenschaften Radius, Position, Linienstärke bzw. -farbe und Füllfarbe erfolgt.
- ⑦ Das Textelement wird erstellt. Die Attribute für die Darstellung der Zeichen werden definiert und der auszugebende Text als Inhalt des Tags angegeben.



Die Beispielsgrafik in der Browserdarstellung

Beispiel *kap17/svg_demo2.xml*

Durch das Hinzufügen des Sub-Tags `<animateTransform>` lässt sich der Text in der Grafik animieren.

```

...
① <text xml:space="preserve" text-anchor="middle" font-family="serif"
  font-size="24" id="svg_3" y="204.000015" x="169.250053" stroke-width="0"
  stroke="#000000" fill="#0000ff" >SVG
② <animateTransform
  attributeName="transform" type="scale"
  begin="0" dur="3"
  from="0" to="1" />
  </text>
...

```


- ① Die Definition des Textes.
- ② Der SVG-Namensraum wird angegeben. Der Typ der Transformation ist `scale`. Über die Attribute `begin` und `dur` werden der Startzeitpunkt und die Dauer der Animation angegeben. Die Spanne der Attribute `from ... to` bestimmt die Vergrößerung des Textes.

17.3 Einbindung von SVG in HTML5

Ursprünglich war SVG als Ergänzung zu XHTML angedacht. Da sich dieses nicht in dem Umfang wie erwartet unter den Webentwicklern durchgesetzt hat, war die Zukunft von SVG unsicher. Dies änderte sich mit der Einführung von HTML5.

Eine SVG-Grafik kann direkt in eine HTML5-Datei eingebunden werden – sowohl in integrierter Form als auch als externe Ressource (als Embedded Content). Die Tags der Grafik werden vom Browser wie normale HTML-Tags verarbeitet. Eine Verwendung eines speziellen XML-Parsers ist somit nicht notwendig. Da der Code der SVG-Insel im HTML5-Code vom Browser als reiner HTML-Code betrachtet wird, muss er nicht zwingend wohlgeformt sei.

In XHTML5 kann der SVG-Code als wohlgeformtes XML-Fragment unter Verwendung des zugehörigen Namensraums ebenfalls integriert werden.



Zur Einbindung einer externen SVG-Datei stehen in HTML5 mehrere Varianten zur Verfügung:

Einbindung über	Prinzip
den <code><object></code> -Tag	<code><object type="image/svg+xml" data="datei.svg"></code> Alternativtext <code></object></code>
ein <code><iframe></code> -Element	<code><iframe src="bild.svg"></code> Alternativtext <code></iframe></code>
den <code></code> -Tag	<code></code>
CSS als Hintergrund	<code>#back {background-image: url(image.svg);}</code>

Bei der Verwendung des Tags `` und beim Einbinden der SVG-Grafik als Hintergrundbild mittels CSS sind Einschränkungen bei den enthaltenen Elementen in der Grafik zu beachten. Beispielsweise dürfen diese keine Animationen enthalten.



Beispiel *kap17/svg_demo3.html*

Die Datei *svg_demo3.html* wird über das `<object>`-Tag in eine HTML5-Webseite eingebunden.

①	<pre> <!DOCTYPE html> <head> <meta charset="utf-8" /> <title>Eingebettete SVG Grafik</title> </head> <body> <object type="image/svg+xml" data="svg_demo.xml"> Ihr Browser unterstützt keine SVG-Grafik </object> </body> </html> </pre>
---	---

- ① Aufruf der SVG-Grafikdatei unter Angabe des Typs `image/svg+xml`. Für Browser, welche die Daten nicht darstellen können, wird ein Alternativtext festgelegt.

#

#FIXED.....	42
#IMPLIED.....	42
#PCDATA.....	31
#REQUIRED	42

&

>.....	132
<.....	132

<

<!ATTLIST	40
<!DOCTYPE	29, 34
<!ELEMENT	30
<!ENTITY.....	47, 50
<!NOTATION	51
<?xml?>.....	16
<?xml-stYLESHEET.....	99, 116

A

Achse.....	104
achse: :knotenprüfung [prädikat]	104
Ajax.....	156
all.....	86
Altova XMLSpy.....	5
Anfangs-Tag.....	15
any.....	90
ANY.....	33
anyAttribute.....	91
anyURI.....	80
Anzahl, Elemente.....	87
Apple Safari	4
arc.....	142
ASCII-Zeichensatz.....	17
Attribut.....	21
Attribut, Schema.....	71
Attributaufzählung	45
Attribute node.....	159
Attribute, Kurzschreibweise.....	178
Attribute, XHTML	178
Attribute, XLink	137
attributeGroup.....	89
Attributes	159
Attributliste.....	40
Attributname.....	22
Attributtyp.....	45
Attributvorgabe.....	41
Aufzählung, Attribut-.....	45
Auswahl, Elemente.....	86
Auszeichnung	15
Auszeichnungssprachen	8
Axis Specifier.....	104

B

Baumstruktur.....	18
Baumstrukturen.....	102
Baumtransformation	115
Bedingte Datentypen	94

Befehlskombination.....	178
Bogen	142
boolean.....	80
Browserweiche	157

C

Cascading Style Sheets	98
Case-sensitive.....	15
CDATA	16, 45
Child node	158
childNodes	159
choice	86
Chrome	4, 5
CML.....	10
complexType.....	82
Conditional Type Alternatives	94
CSS	9, 98
CTA	94

D

date.....	78
Datenaustausch.....	11
Datenblock, extern.....	171
Datenblock, intern	169
Datenformatierung.....	96
Dateninsel (Data Island).....	169
Datenselektion	116
Datentyp.....	76
Datentyp, Schema	70
Datentypen, XPath 2.0	110
Datum.....	78
decimal.....	77
Dezimalzahl.....	77
display:	99
Document Object Model, DOM	156
Dokument strukturieren	23
Dokument, gültiges.....	37
Dokumenttyp	175
Dokumenttyp-Definition.....	176
Dokumenttyp-Definition (DTD)	28
DOMParser	170
DSSSL	96
DTD.....	10, 28
DTD (Document Type Definition)	9
DTD, Aufbau.....	32
DTD, externe	34
DTD, interne	29
duration.....	79
Durchlaufregel	140

E

Editor	25
Einschränkung, Schema.....	72
Element	14, 18
Element, Attribut	21
Element, Inhalt.....	32
Element, komplex	82
Element, leer	83
Element, mit einem Inhalt	18
Element, ohne Inhalt.....	20

Elementbezeichnung.....	15
Elementgruppe	30
Elementhierarchie.....	20
Elementinhalt.....	32, 120
Elementtyp.....	30
Elementverknüpfung.....	30
Eltern-Element	105
Eltern-Knoten.....	159
EMPTY	33
encoding.....	17
End-Tag	15
Entität	15
Entity.....	47
ENTITY.....	47
Entity, externe.....	50
Ereignisse	168
erweitern, Schema.....	90
EventHandler	169
Extensible HTML	174
extension.....	85

F

Fallunterscheidung	130, 132
Fassetten	72, 91
Feinstruktur.....	24
Filter für XML-Daten.....	97
Filter, XPath	119
Firefox	4
firstChild	159
FLWOR-Ausdrücke	150
for, XPath 2.0	110
Formal Public Identifier	35
Formatierung.....	96
FPI	35
Free XML Editor Editix	5
Funktionen, XPath.....	112

G

Ganzzahl	77
Geschwister.....	105
getAttribute	159
getElementById.....	160
getElementsByTagName	160
Google Chrome.....	4, 5
Grobstruktur	23
Groß- und Kleinschreibung	15
group.....	88
Gruppe, Attribute	89
Gruppe, Elemente.....	88
Gültiges Dokument.....	37

H

hasChildNodes	159
Hauptelement.....	19, 105
Hierarchie, Element-	20
HTML	8
HTML in XSL	128
HTML Validation Service.....	180
HTML4.....	174
HTML5.....	175
Hyperlink.....	136

I

ID 45, 46
 IDREF.....45
 Indikator.....86
 Inhaltsmodell.....33
 integer.....77
 Internet Explorer.....4
 Intervall, Datentyp.....79
 ISO.....17
 ISO 639.....22

J

JavaScript.....179
 JSON.....156

K

Kalender.....78
 Kind-Element.....104
 Kind-Knoten.....159
 Knoten.....105
 Knotenanzahl.....162
 Knotenprüfung.....107
 Kombination, Befehls-.....178
 Kommentare.....22
 Kommentare, XPath.....107
 Kurzschreibweise.....21, 178

L

last.....108
 lastChild.....159
 Latin-Zeichensatz.....17
 length.....74
 Location Paths.....105
 locator.....141
 Lokalisierung.....104

M

MathML.....10
 maxOccurs.....87
 minOccurs.....87
 mixed.....85
 Mozilla Firefox.....4
 Mustervergleich.....119

N

Nachfahre.....104
 Nachkomme.....105
 Namensraum.....57
 Namensraum, XLink.....137
 nextSibling.....159
 NMTOKEN.....46
 node.....107
 Node Test.....104, 107
 nodeName.....159
 nodeType.....159
 nodeValue.....159
 normalizedString.....76
 Notation.....51
 NOTATION.....47

O

Obergrenze.....87
 ODER-Funktion.....31
 Offene Modelle.....93
 Opera.....4, 5
 Operatoren, XPath.....111

P

Parent node.....158
 parent::node.....108
 parentNode.....159
 Parser.....37
 Parser, XML-.....9
 pattern.....73
 PCDATA.....31
 position.....108
 Prädikat.....108
 previousSibling.....159
 Prolog.....14, 16
 PUBLIC.....35

R

RegExpressions.....73
 Reihenfolge, Elemente.....86
 Reservierte Zeichen.....15
 responseText.....157
 responseXML.....157
 restriction.....72

S

Safari.....4
 SAX.....168
 SAX-Parser.....168
 Schema.....10, 64
 Schema, Attribut.....71
 Schema, Datentyp.....70
 Schema, einfaches Element.....70
 Schema, Einschränkungen.....72
 Schema, extern.....67
 Schemaweite Attribute.....93
 Schleifen.....126
 Selektierungspfad.....105
 Selektor.....115
 self::node.....108
 Semantik.....97
 sequence.....86
 SGML.....8
 Simple API for XML, SAX.....168
 simpleContent.....84
 simpleType.....72
 SMIL.....10
 Sortieren.....129
 Sprungmarke.....136
 SQL.....146
 standalone.....34
 string.....76
 Strukturieren eines Dokuments.....23
 Stylesheet.....179
 SVG.....96

T

tagName.....159
 Template.....116, 118
 Template-Aufrufe.....122
 textNode.....159
 time.....79
 title.....143
 token.....76
 Token.....45
 Transformation.....97
 Transformationsphase.....115

U

Übungsdateien.....6
 Uhrzeit.....79
 Unterelement.....19
 URI.....34, 57
 URL.....34
 UTF-8.....16
 UTF-8/16.....17

V

Valid Document.....37
 Validierung.....28, 38
 Verknüpfung, Element-.....30
 Verschachtelung von HTML-Tags... 177
 Versionsangabe.....16
 Verweis, erweiterter.....141, 142
 Verweis, lokaler.....138
 Verweis, multidirektionaler.....140
 Vorfahre.....104, 105
 Vorlage.....116, 118

W

W3C.....8
 Wahrheitswert, Datentyp.....80
 Webcode.....6
 Well Formed Document.....23
 Wert prüfen,
 Fallunterscheidung.....132
 Wertebereich, Schema.....72
 Wertvorgabe, Attribut-.....41
 WHATWG.....175
 whitespace.....74
 Wildcard.....90
 Wohlgeformtheit.....23, 37

X

XAMPP.....157
 XBase.....144
 XHTML.....10, 174
 XLink.....10, 136
 XLink, einfacher Link.....138
 XLink, erweiterte Links.....140
 XLink, multidirektionale Links.....140
 xlink:extended.....140
 xlink:simple.....137
 XML (Extensible Markup
 Language).....9, 14
 XML Namespaces.....57

XML Schema.....	64	XPath-Ausdrücke.....	149	xsl:otherwise.....	132
XML Schema 1.1.....	91	XPointer.....	10, 136	xsl:sort.....	129
XML verwenden.....	11	XQJ.....	148	xsl:value-of.....	120
XML, Entstehung	8	XQuery.....	10	xsl:when.....	132
XML-Element	18	XQuery 1.0.....	146	XSL-FO.....	97, 114, 115
XML-Fragment.....	152	XQuery, Abfrageausdrücke.....	147	XSL-FO (XSL Formatting Objects) ...	98
XMLHttpRequest.....	157	XQuery, Datenmodell	146	XSLT.....	10, 114
XML-Knoten.....	102	XQuery, Implementierungen	148	XSLT (XSL-Transformation).....	97
xmlns.....	57, 59	XQuery, Syntax.....	146		
xmlns:Namensraum.....	58	XQuery-Abfrage	152	Z	
XML-Prüfung.....	38	XSL.....	10, 114	Zeichen, reservierte.....	15
XMLSpy	152	XSL (Extensible Stylesheet		Zeichenkette, Datentyp	76
XML-Syntax	14	Language)	96	Zeichenlänge, Schema	74
XML-Tools	5	XSL, Namensraum.....	115	Zeichensatz	17, 176
XPath.....	10, 97, 102, 114, 119, 148	xsl:apply-templates.....	116	Zeitraum.....	79
XPath 2.0.....	109	xsl:call-templates.....	118	Zusicherungen.....	92
XPath-Datentypen	103	xsl:choose.....	132		
XPath-Dokumentreihenfolge	103	xsl:for-each.....	126		
XPath-Funktionen.....	108	xsl:if	130		

Impressum

Matchcode: XML11

Autor: Elmar Fuchs, Heiko Schröder

Redaktion: Andrea Weikert

Produziert im HERDT-Digitaldruck

5. Ausgabe, Januar 2015

HERDT-Verlag für Bildungsmedien GmbH
Am Kümmerling 21-25
55294 Bodenheim
Internet: www.herdtd.com
E-Mail: info@herdtd.com

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.