

2 XML-Grundlagen *

Dieses Kapitel führt Sie in die Grundlagen von XML ein. Hierzu sind keine Vorkenntnisse erforderlich:

- »Modell der strukturierten Dokumente«, S. 3
- »Aufbau eines XML-Dokumentes«, S. 10
- »XML-Elemente und -Attribute«, S. 13
- »Wohlgeformtes XML«, S. 20



Zum Schreiben eines XML-Dokumentes eignet sich jeder normale Texteditor. Allerdings werden Sie sehr schnell feststellen, dass dies alles andere als komfortabel ist. Daher sollten Sie möglichst früh mit einem speziellen XML-Editor arbeiten. Eine Auswahl wird im kostenlosen E-Learning-Kurs zu diesem Buch vorgestellt.

Was Sie brauchen

[RoRi03], [Vonh07], [Haro02], [Ray 04]

Literatur

2.1 Modell der strukturierten Dokumente *

Bestandteile eines Dokumentes sind Inhalt, Struktur und Layout. Mit XML ist es möglich eine Strukturdefinition für eine Klasse von Dokumenten anzugeben. Die dazu passenden Dokumentinstanzen werden entsprechend ausgezeichnet Das Layout wird in einem *Stylesheet* beschrieben. Instanz und Stylesheet werden einem Prozessor übergeben, der das Ergebnisdokument generiert. Eine XML-Anwendung ist eine Auszeichnungssprache, die mit XML definiert worden ist.

XML hat heute zwei Haupteinsatzgebiete: Zum einem ist mit XML eine layoutunabhängige Beschreibung von Dokumenten möglich und somit kann XML als Ausgangsformat für Single-Source-Publishing verwendet werden. Das zweite Einsatzgebiet ist der Einsatz von XML als universelles Datenaustauschformat.

In vielen Anwendungsbereichen ist es heute erforderlich, Dokumente für mehrere Medien bereitzustellen. Zum Beispiel sollen Dokumente der Marketingabteilung gedruckt und online bereitgestellt werden, Schulungsunterlagen in Form eines Skriptes und einer Präsentation veröffentlicht werden. Gewünscht wird häufig auch eine adressatenge-

Single-Source-Publishing

rechte Produktion, die Schulungsunterlage für den Dozenten soll auch Musterlösungen enthalten, die Bedienungsanleitung für ein Auto nur genau die Komponenten beschreiben, die dieses Fahrzeug auch enthält. Es wird schnell klar, dass es wenig effektiv und sehr kostenintensiv ist, jedes dieser Dokumente einzeln zu erstellen. Wünschenswert ist, alle Dokumente aus einer Quelle und möglichst automatisiert zu generieren. Dieser Prozess wird heute als Single-Source-Publishing bezeichnet.

Verwendet man zu Erstellung der Ausgangsdokumente ein herkömmliches Textverarbeitungsprogramm, stößt man sehr schnell an Grenzen, was die Qualität und auch die Flexibilität anbelangt. Die Entwicklung einer besseren Lösung erfordert zunächst einen genauen Blick auf die Bestandteile eines Dokumentes.

- Dokument Inhalt, Struktur, und Layout sind Bestandteile jedes Dokumentes.
- Inhalt Inhalt ist die zu vermittelnden Information, oft in Textform, ergänzt durch Bilder, immer häufiger auch durch multimediale Elemente, wie Töne oder Videos.
- Struktur Struktur ist die Aufteilung und Abfolge der Informationen. Beispielsweise haben die Kapitel dieses Buches – grob betrachtet – immer den gleichen Aufbau: Zuerst eine Überschrift, dann eine Kurzfassung – die Essenz des Inhalts –, dann folgt der eigentliche Inhalt (Abb. 2.1-1).



Abb. 2.1-1: Grobstruktur.

Diese grobe Struktur kann nun weiter verfeinert werden: Zum Beispiel besteht die Überschrift aus einem Text, diesem Text wird die Kapitelnummerierung vorangestellt, nach dem Text folgt eine Angabe über den Schwierigkeitsgrad des Kapitels (Abb. 2.1-2).



Abb. 2.1-2: Feinstruktur.

Das Layout, ausgedrückt durch eine entsprechende Formatierung, dient zur Visualisierung des Inhaltes und der Struktur des Dokumentes. Für Überschriften wird im Vergleich zum Inhaltstext eine größere Schriftart und der Fettdruck verwendet, die Kurzfassung wird fettgedruckt, in gleicher Schriftgröße wie der Inhaltstext. Im Inhaltstext selbst werden unterschiedlichste Stilmittel verwendet: Zwischenüberschriften dienen einer weiteren Strukturierung des Inhalts, diese werden fettgedruckt, Beispiele werden grau hinterlegt, Glossar-begriffe werden fett und englische Begriffe kursiv angezeigt.

Layout

Die Formatierung erleichtert das Erkennen der Struktur, die Struktur wiederum erleichtert das Erkennen des Informationsgehaltes eines Dokumentes.

Zusammen-
gefasst

Beim Publizieren für eine anderes Medium genügt es nicht, die Publikation lediglich in ein neues Format zu konvertieren. Es müssen auch die medienspezifischen Besonderheiten des Zielproduktes bezüglich Struktur und Layout berücksichtigt werden. Beispielsweise soll ein Buch nicht als eine lange HTML-Seite dargestellt werden, sondern aufgeteilt in viele Seiten, etwa eine Seite pro Kapitel, versehen auch mit passenden Hyperlinks.

Dies lässt sich kaum automatisieren, wenn die Strukturinformation nicht wirklich vorhanden ist, sondern mit Formatierungsinformationen vermischt ist. Im dargestellten Beispiel können die Kapitel nur dann in einzelne HTML-Seiten transformiert werden, wenn die Kapitelgrenzen klar erkennbar sind.

Die Vermischung von Struktur, Inhalt und Layout kommt daher, dass viele Dokumentformate es nur ermöglichen, ein Dokument zu formatieren. Strukturinformation lassen sie nicht wirklich zu. In einer Textverarbeitungssoftware, wie z. B. MS Word, kennzeichnen Sie beispielsweise einen Absatz als Überschrift, indem Sie für diesen Absatz eine größere Schriftart und den Fettdruck wählen. Sind Sie ein fortgeschrittener Word-Benutzer, arbeiten Sie mit Formatvorlagen und weisen dem Absatz die Formatvorlage »Überschrift« zu. Aber unabhängig von Ihrer Vorgehensweise, Sie kennzeichnen die Struktur, indem Sie formatieren.

Trennung:
Struktur -
Inhalt - Layout

Für einen automatisierten Single-Source-Publishing-Prozess ist es jedoch notwendig, Struktur und Inhalt vom Layout zu trennen. Realisiert werden kann dies, indem Inhalte nicht mit Formatierungs-, sondern mit Strukturinformationen gespeichert werden. Das heißt, ein Absatz, der eine Überschrift ist, wird als solcher gekennzeichnet, »ausgezeichnet«. Alle Formatierungsinformationen für diese Überschrift werden davon getrennt – im Idealfall in einem eigenen Dokument – gespeichert.

Markup

Die Auszeichnungen werden Markierungen (*markup*) genannt. Der Begriff des *Markup* stammt aus dem Verlagswesen, aus einer Zeit als Begriffe wie Desktop Publishing noch unbekannt waren. Hiermit wurden die typografischen Festlegungen bezeichnet, die in Form handschriftlicher Markierungen in das Manuskript eingefügt und anschließend im Satz berücksichtigt wurden. Im Kontext elektronischer Dokumente ist **Markup** eine Folge von Zeichen, die an bestimmten Stellen in einem Dokument eingefügt werden, um die Form der Darstellung, des Druckes oder der Struktur der Dokumente zu beschreiben. Die einzelnen voneinander getrennten Markup-Elemente nennt man Tags.

Struktur-
definition

Zusätzlich wird noch vorab, in einer so genannten **Strukturdefinition** (Dokumentgrammatik), festgelegt, wie Dokumente aufgebaut sind, also welche Struktur sie haben.

Dokument-
instanz

Dokumente, die einer bestimmten Strukturdefinition entsprechen, werden dann als **Dokumentinstanz**, kurz Instanz, bezeichnet (Abb. 2.1-3)

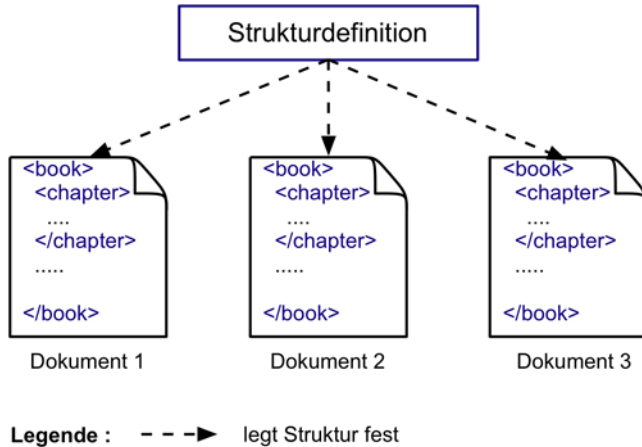


Abb. 2.1-3: Strukturdefinition – Dokumentinstanz.

- + Es kann per Programm überprüft werden, ob die Dokumente einer vorgegebenen Struktur entsprechen. Dies wird als **Validierung** bezeichnet. Vorteile
- + Die logische Strukturierung unterstützt weitere Anwendungen der Dokumentenverarbeitung, z.B. kann die Kapitelnummerierung berechnet, ein Inhaltsverzeichnis oder eine Übersicht der verwendeten englischen Begriffe automatisch generiert werden.

Das Layout wird in einem sogenannten **Stylesheet** beschrieben, d.h. ein *Stylesheet* enthält Anweisungen oder Regeln, wie die Inhalte in einem strukturierten Dokument formatiert werden sollen.

Das Dokument mit der Strukturmarkierung und das *Stylesheet* werden einem Programm (Prozessor) übergeben. Dieses generiert daraus das Ergebnisdokument (Abb. 2.1-4).

Die Trennung der Formatierungsinformation vom Inhalt und die Auslagerung in *Stylesheets* hat folgende Vorteile:

- + Verschiedene Dokumente können mit ein- und demselben *Stylesheet* einheitlich formatiert werden.
- + Ein- und dasselbe Dokument kann durch Kombination mit verschiedenen *Stylesheets* passend zum Anwendungszweck formatiert werden.

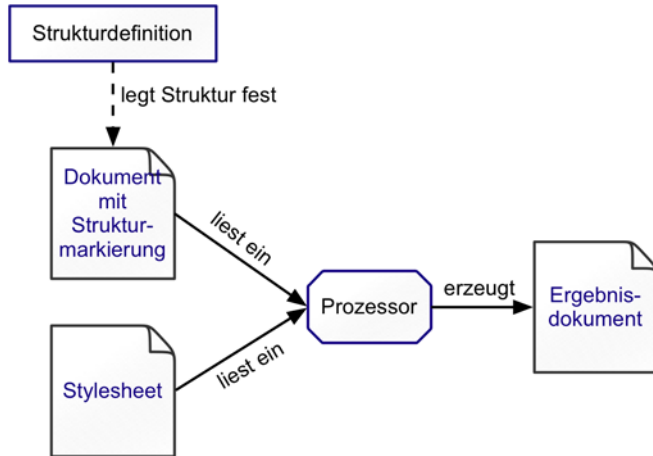


Abb. 2.1-4: Prozess.

- ✦ Es kann arbeitsteilig und parallel gearbeitet werden: ein Mitarbeiter ist für die Texte zuständig, ein anderer für die *Stylesheets*.

Benötigt wird jetzt noch eine Sprache zur Beschreibung der Strukturdefinition eines Dokumentes bzw. einer Klasse von Dokumenten und auch eine Festlegung, wie die strukturelle Auszeichnung in den Dokumenten realisiert wird.

SGML 1986 wurde die **SGML** (*Standard Generalized Markup Language*) als ISO-Standard veröffentlicht. SGML ist eine Metasprache zur Beschreibung von **Auszeichnungssprachen** (Markup-Sprachen). Mit ihr ist es also möglich eigene Markup-Sprachen zu definieren. Die Strukturdefinition wird in einer sogenannten DTD (*Document Type Definition*) beschrieben. Durch diese DTD sind nun die Auszeichnungselemente, **Tags**, festgelegt. Die Namen für die Auszeichnungselemente wählt der DTD-Entwickler so, dass sie eine semantische Bedeutung haben. Der Name eines Auszeichnungselementes gibt somit bereits Informationen über den Inhalt.

HTML Die bekannteste Sprache, die mit SGML definiert wurde, ist **HTML**. D. h. mit SGML wurde festgelegt, welche *Tags* in HTML erlaubt sind. HTML zu lernen und anzuwenden ist jedoch auch ohne Kenntnis von SGML problemlos möglich.

SGML selbst ist sehr komplex und hat sich daher nur in wenigen Branchen durchgesetzt. Mit der Verbreitung des Internets wuchs jedoch der Bedarf nach einer Metasprache, die Anwendungsmöglichkeiten wie SGML bietet, aber weniger komplex ist.

Hintergrund dieses Bedarfs ist, dass ein Format, das Informationen durch Auszeichnungselemente kennzeichnet, sich sehr gut als universelles Datenaustauschformat zwischen Anwendungen eignet. Es ist menschen- und maschinenlesbar und kann beliebig komplexe, hierarchische Strukturen abbilden.

Universelles
Datenaustauschformat

Das W3C beauftragte daher eine Arbeitsgruppe zur Entwicklung der Sprache ***extensible Markup Language***, kurz XML. Beim Entwurf von XML wurden die folgenden zehn Ziele verfolgt:

Entwurfsziele
von XML

- 1 XML soll sich im Internet auf einfache Weise nutzen lassen.
- 2 XML soll ein breites Spektrum von Anwendungen unterstützen.
- 3 XML soll zu SGML kompatibel sein.
- 4 Es muss einfach sein, Programme zu schreiben, die XML-Dokumente verarbeiten.
- 5 Die Anzahl optionaler Merkmale in XML soll minimal sein, idealerweise Null.
- 6 XML-Dokumente sollen für Menschen lesbar und angemessen verständlich sein.
- 7 Der XML-Entwurf soll zügig abgefasst werden.
- 8 Der Entwurf von XML soll formal und präzise sein.
- 9 XML-Dokumente sollen leicht zu erstellen sein.
- 10 Die Knappheit von XML ist von minimaler Bedeutung.

1998 wurde vom W3C die Version 1.0 der XML-Spezifikation verabschiedet, siehe W3C-XML-Website (<http://www.w3.org/TR/REC-xml/>). Innerhalb kurzer Zeit wurden auf XML-basierende Sprachen und auch Werkzeuge zur Verarbeitung – viele davon Open-Source – entwickelt.

Eine Auszeichnungssprache, die mit XML definiert worden ist, heißt **XML-Anwendung**. Eine XML-Anwendung konzentriert sich immer auf ein bestimmtes Problemfeld.

XML-
Anwendung

Beispiele für XML-Anwendungen sind:

- eXtensible Hypertext Markup Language (<http://www.w3.org/TR/xhtml11/>), kurz XHTML, Neuformulierung von HTML mit XML.
- Scalable Vector Graphics (<http://www.w3.org/Graphics/SVG/>), kurz SVG, für die Erstellung von zweidimensionalen Vektorgrafiken. Die Grafiken lassen sich verlustfrei skalieren, sie können Animationen enthalten. Interaktionen können mit JavaScript programmiert werden.
- Synchronized Multimedia Integration Language (<http://www.w3.org/AudioVideo/>), kurz SMIL, zur Beschreibung von zeitsynchronisierten, multimedialen Inhalten. Mathematical Markup Language (<http://www.w3.org/TR/MathML2/>), kurz MathML, zur Darstellung mathematischer Formeln.
- XForms (<http://www.w3.org/MarkUp/Forms/>) zur Darstellung elektronischer Formulare.
- DocBook (<http://www.docbook.org/>) zur Erstellung von Büchern, Artikeln und Dokumentationen im technischen Umfeld (Hardware oder Software).
- Text Encoding Initiative (<http://www.tei-c.org>), kurz TEI, zur Darstellung von Texten, insbesondere in den Geisteswissenschaften.
- RSS, Akronym für *Really Simple Syndication*, *Rich Site Summary* oder *RDF Site Summary*. XML-Anwendung für *Newsfeeds*. Es gibt verschiedene Versionen, siehe Webseite des RSS Advisory Boards (<http://www.rssboard.org/>).
- BMECat, ein standardisiertes Austauschformat für Katalogdaten im Katalogmanagement, das vom Bundesverband Materialwirtschaft, Einkauf und Logistik e. V. (<http://www.bmecat.org>) gepflegt wird.
- eXtensible Business Reporting Language (<http://www.xbrl.de>), kurz XBRL, zur Darstellung von elektronischen Dokumenten im Bereich der Finanzberichterstattung, z. B. Jahresabschlüssen.

2.2 Aufbau eines XML-Dokumentes *

Jedes XML-Dokument sollte mit einem Prolog beginnen. Er enthält die XML-Deklaration, die Informationen für den Parser angibt, insbesondere über die verwendete XML-Version und Kodierung. Optional sind Verarbeitungsanweisungen

gen und der Verweis auf eine Doctype-Definition oder ein XML-Schema. Dann folgen die XML-Daten, der mit Markierungen ausgezeichnete Text. XML-Parser lesen XML-Dokumente und prüfen sie auf Korrektheit.

Die Abb. 2.2-1 zeigt den allgemeinen Aufbau eines XML-Dokumentes.

Prolog	XML-Deklaration	<code><?xml version="1.0" encoding="UTF-8"?></code>
	Verarbeitungsanweisung (optional) hier: Stylesheet-Zuordnung	<code><?xml-stylesheet type="text/css" href="d.css" ?></code>
	Verweis auf DTD (optional)	<code><!DOCTYPE dozent SYSTEM "dozent.dtd"></code>
XML-Daten (getaggtter Text)		<code><dozent did="d1" anrede="Frau"> <name>Schmitt</name> <vorname>Sabine</vorname> </dozent></code>

Abb. 2.2-1: Aufbau eines XML-Dokumentes.

Jedes XML-Dokument *sollte* mit einem **Prolog** beginnen. Die erste Zeile des Prologs ist die so genannte **XML-Deklaration**. Vor der XML-Deklaration dürfen keine Leerzeichen stehen.

Prolog

```
<?xml version="1.0" encoding="UTF-8" ?>
```

XML-Deklaration

Mögliche Attribute sind `version`, `encoding` und `standalone`.

Nur das Attribut `version` ist Pflicht. Werden auch die anderen notiert, müssen sie in der angegebenen Reihenfolge auftreten.

Mit `version` wird die verwendete XML-Version angegeben. Diese Angabe ist zwingend erforderlich. Zur Zeit existieren die Versionen 1.0 und 1.1. Der Einsatz der Version 1.0 wird empfohlen, da die Version 1.1 nicht in allen Fällen rückwärtskompatibel ist.

version

- encoding** Das Attribut `encoding` gibt die im Dokument verwendete **Zeichenkodierung** an. Sie gibt an, mit welcher Codierung die Datei gespeichert wird. Fehlt die Angabe wird als Vorgabe UTF-8 (*8-Bit Unicode Transformation Format*) verwendet.
- standalone** Ein weiteres – aber selten verwendetes – Attribut, ist `standalone`. Erlaubte Attributwerte sind `yes` und `no`. `standalone="no"` bedeutet, dass die DTD innerhalb der DOCTYPE-Deklaration angegeben, also gemeinsam mit der XML-Instanz gespeichert wird. Man spricht in diesem Fall von einer internen DTD (*internal subset*). `standalone="yes"` bedeutet, dass die DTD in einer eigenen Datei abgespeichert ist. (siehe dazu auch das Kapitel »Verknüpfung DTD – XML-Instanz«, S. 35)
- Verarbeitungsanweisung** **Verarbeitungsanweisungen** (*processing instructions*) sind Anweisungen für weiterverarbeitende Programme. Mit ihrer Hilfe können Informationen vom XML-Parser an eine Anwendung weitergereicht werden. Sie beginnen mit `<?` und enden mit `?>`.
- Die Syntax einer Verarbeitungsanweisung lautet:
- ```
<?PI-Name PI-Anweisung?>
```
- Als `PI-Name` ist die Zeichenkette »xml« nicht erlaubt.
- Der gebräuchlichste Einsatz von Verarbeitungsanweisungen ist die Zuweisung von *Stylesheets* (CSS oder XSLT) zu XML-Dokumenten, die sogenannte **Stylesheet-Referenz**. Moderne Webbrowser und XML-Editoren können diese Verarbeitungsanweisung interpretieren und zeigen das XML-Dokument entsprechend formatiert an.
- Beispiel** Mit folgender Verarbeitungsanweisung wird ein XML-Dokument mit einem CSS-Stylesheet verbunden. Ein Webbrowser zeigt das XML-Dokument dann entsprechend den im CSS-Stylesheet angegebenen Formatierungsregeln an.
- ```
<?xml-stylesheet type="text/css" href="format.css" ?>
```
- Verweis auf DTD** Soll das XML-Dokument bzgl. einer DTD validiert werden, wird mit einer so genannten DOCTYPE-Deklaration auf die DTD verwiesen oder sie wird als interne DTD eingebunden. Genauere Informationen darüber erhalten Sie im Kapitel »Verknüpfung DTD – XML-Instanz«, S. 35.

Den größten Teil des XML-Dokumentes umfasst der mit den Strukturmarkierungen versehene, so genannte getaggte Text.

Getaggter Text

XML-Parser sind Programme, die ein XML-Dokument lesen und die einzelnen Markierungen herausfiltern können. XML-Parser prüfen ein Dokument beim Einlesen auf Korrektheit. Dabei unterscheidet man:

XML-Parser

■ **Nicht-validierende Parser**

Sie prüfen, ob ein Dokument die syntaktischen Regeln der Spezifikation (korrekte Schachtelung und Bezeichnung der Strukturelemente) erfüllt. Wird ein Fehler gefunden, meldet der Parser diesen und bricht den Parsing-Vorgang ab.

■ **Validierende Parser**

Sie prüfen, ob die Struktur des XML-Dokumentes den Vorgaben einer Dokumenttypdefinition oder eines Schemas entspricht.

Weiterhin kann eine Applikation über den Parser auf das XML-Dokument zugreifen. Dazu bietet der Parser der aufrufenden Applikation eine vergleichsweise komfortable Schnittstelle (*Application Programming Interface*, kurz: API) für den Zugriff auf die einzelnen Dokumentbestandteile (siehe »SAX und DOM«, S. 293).

API

2.3 XML-Elemente und -Attribute *

Es gibt einfache, strukturierte und leere Elemente sowie Elemente mit gemischtem Inhalt. Zu jedem Element können beliebig viele Attribute in Form von Name-Wert-Kombinationen angegeben werden. Abschnitte, die nicht geparkt werden sollen, können als CDATA-Abschnitte notiert werden. Da die Zeichen <, >, ", ' und & in XML eine besondere Bedeutung haben, müssen diese durch Zeichen-Referenzen ersetzt werden.

Elemente

Die »Grundbausteine« eines XML-Dokumentes sind **Elemente**. Sie beschreiben die Struktur des XML-Dokuments und enthalten andere Elemente oder Text oder beides.

Elemente haben einen Namen und bestehen aus (Abb. 2.3-1)

- einem Start-Tag (*start tag*),
- dem dazugehörigen Ende-Tag (*end tag*) und
- einem Inhalt (*content*).

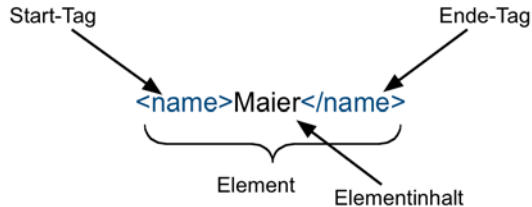


Abb. 2.3-1: Ein einfaches Element.

Elementname Für Elementnamen gelten die folgenden Regeln:

- Der Name muss mit einem Buchstaben oder Unterstrich beginnen.
- Danach dürfen als zusätzliche Zeichen Ziffern, Bindestrich und Punkt verwendet werden.
- Den Doppelpunkt sollten Sie möglichst nicht verwenden, da er das Namensraum-Präfix vom Elementnamen trennt (vgl. Kapitel »XML-Namensräume«, S. 79).
- Die Zeichenfolge »xml« ist reserviert und darf nicht am Anfang eines Namens stehen.
- Die Länge des Namens ist nicht begrenzt.
- Groß- und Kleinschreibung werden unterschieden.

Hinweis Wählen Sie Namen mit semantischer Bedeutung, so dass der menschliche Leser anhand des Namens bereits Informationen über die Bedeutung des Inhalts erhält.

Unterschieden werden

- einfache Elemente,
- strukturierte Elemente,
- Elemente mit gemischtem Inhalt und
- leere Elemente.

Einfache Elemente Bei einfachen Elementen besteht der Inhalt aus einer Zeichenkette.

Beispiel 1 Ein einfaches Element: `<name>Maier</name>`

Strukturierte Elemente enthalten selbst wieder Elemente, die sowohl einfache als auch strukturierte Elemente sein können. Diese Elemente müssen korrekt ineinander verschachtelt sein, ansonsten wird die so genannte Wohlgeformtheit (siehe dazu Kapitel »Wohlgeformtes XML«, S. 20) verletzt.

Strukturierte
Elemente

Das Element `dozent` ist ein strukturiertes Element, das die beiden einfachen Elemente `name` und `vorname` enthält:

Beispiel 2

```
<dozent>
  <name>Maier</name>
  <vorname>Fritz</vorname>
</dozent>
```

Ein Element, das andere Elemente enthält, wird **Elternelement** dieser Elemente genannt. Die enthaltenen Elemente werden **Kindelemente** oder auch **Unterelemente** des Elternelementes genannt.

Im Beispiel 2 ist `dozent` das Elternelement der Elemente `name` und `vorname`. Umgekehrt sind `name` und `vorname` Kindelemente von `dozent`.

Beispiel 3

Jedes XML-Dokument besitzt ein **Wurzelement** (*root element*). Es ist das erste Element im Dokument und dasjenige, das alle anderen Elemente enthält. Jedes XML-Dokument bildet daher eine Baumstruktur von Elementen.

Wurzelement

Es wird empfohlen, die Elemente entsprechend der Schachtelung einzurücken. XML-Editoren bieten hierzu oft einen Menüpunkt oder ein Icon zur automatischen Formatierung an.

Hinweis

Elemente mit **gemischtem Inhalt** (*mixed content*) sind Elemente, die andere Elemente und auch Text gleichzeitig enthalten.

Gemischter
Inhalt

Das Element `<beschreibung>` ist ein Element mit gemischtem Inhalt. Es enthält Text und das Kindelement `em`:

Beispiel 4

```
<vorlesung>
  <beschreibung>Bei dieser Veranstaltung
  gilt<em>Anwesenheitspflicht.</em>
</beschreibung>
</vorlesung>
```

-
- Hinweis** Elemente mit gemischtem Inhalt werden eher bei narrativen Texten, also Texten im »Erzählstil«, selten bei XML-Dokumenten für den elektronischen Datenaustausch (*Electronic Data Interchange*) verwendet.
-
- Leere Elemente** Neben Elementen mit Inhalt, gibt es auch **leere Elemente**, d. h. Elemente, die keinen eigenen Inhalt in Form von Elementen oder Daten haben. Sehr üblich ist statt der Schreibweise `<Elementname></Elementname>`, auch die Kurzform `<Elementname/>`.
-
- Hinweis** Beachten Sie, dass zwischen dem Start- und Ende-Tag eines leeren Elementes kein Zeichen, auch kein Leerzeichen oder Zeilenumbruch, stehen darf.
-

Attribute

Jedes Element kann beliebig viele **Attribute** enthalten. Sie werden im Start-Tag durch Name-Wert-Paare angegeben (Abb. 2.3-2). Die Attributwerte müssen stets in einfachen oder doppelten Hochkommata angegeben werden. Werden mehrere Attribute aufgeführt, müssen diese durch ein Leerzeichen getrennt werden.

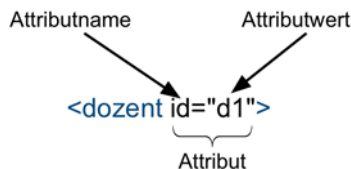


Abb. 2.3-2: Ein Element mit Attribut.

- Attributname** Für die Bildung von Attributnamen gelten die gleichen Regeln wie für Elementnamen. Innerhalb eines Elementes muss jeder Attributname eindeutig sein. Es ist jedoch erlaubt, den gleichen Attributnamen bei verschiedenen Elementen zu verwenden.

Elemente oder Attribute?

Häufig stellt sich die Frage, ob Informationen als Elemente oder in Form von Attributen modelliert werden sollen. Hier-

auf gibt es keine allgemeingültige Antwort, sondern lediglich einige Entscheidungshilfen.

- Elemente sollten eher den eigentlichen darzustellenden Inhalt enthalten, während in Attributen Zusatzinformationen oder Metadaten über den Inhalt gespeichert werden sollten.

In Abb. 2.3-2 wird dem `dozent`-Element das Attribut `id` zugefügt, um ihm eine eindeutige Kennung zu geben.

Hier ist die Angabe der Sprache eine Metainformation zum Text:

```
<titel sprache="de">XML-Kurs</titel>
<titel sprache="en">XML course</titel>
```

Beispiel

- Soll eine Information weiter strukturiert werden oder tritt eine Information mehrfach auf, müssen Elemente verwendet werden.

```
<name>
  <vorname>Hans</vorname>
  <vorname>Peter</vorname>
  <nachame>Maier</nachame>
</name>
```

Ein Name besteht aus 1 oder mehreren Vornamen und einem Nachnamen.

Beispiel

- In Weiterverarbeitungsprozessen, z. B. mit XSLT, kann auf Elementinhalte und Attributwerte zugegriffen werden. Werden jedoch alternative Bedingungen durch verschiedene Attributwerte repräsentiert und nicht durch An- oder Abwesenheit von Elementen, lassen sich entsprechende Fallunterscheidungen einfacher und besser verständlich programmieren [EcEc04].
- Wird eine Strukturdefinition über eine DTD angegeben, sind für Elemente und Attribute unterschiedliche Datentypen vorgesehen. Für Attributwerte etwa kann eine Liste von erlaubten Werten notiert werden, für Elemente ist dies nicht möglich. Bei der Verwendung von XML-Schema entfällt dieses Argument, da alle Datentypen sowohl für Elemente als auch für Attribute verwendet werden können.

Kommentare

Wie in Programmiersprachen üblich, können Kommentare verwendet werden, um Notizen in das XML-Dokument einzufügen oder bestimmte Abschnitte zeitweise vom Parsing-Vorgang auszuschließen, d. h. »auszukommentieren«.

Kommentare beginnen mit `<!--` und enden mit `-->`. Alles was zwischen `<!--` und `-->` steht wird vom XML-Parser ignoriert.

Beispiel 5

```
<?xml version="1.0" ?>
<!-- nun folgt ein dozent-Element -->
<dozent>
  <name>Maier</name>
  <vorname>Fritz</vorname>
</dozent>
```

Beachten Sie:

- Kommentare dürfen *nicht* vor der XML-Deklaration stehen.
- Kommentare dürfen nicht in Tags eingefügt werden.
- Die Zeichenfolge `--` (zwei Bindestriche) darf nicht innerhalb eines Kommentars vorkommen. Daher sind auch Kommentare innerhalb von Kommentaren nicht erlaubt.

Zeichen-Entities

Die Zeichen `<`, `>`, `"`, `'` und `&` haben in XML eine besondere Bedeutung. Sie dienen als Begrenzungszeichen für Tags bzw. Attributwerte und dürfen daher nicht innerhalb des Inhaltes eines Elementes oder innerhalb eines Attributwertes verwendet werden. Werden sie verwendet, zeigt der XML-Parser eine entsprechende Fehlermeldung an.

Für diese Zeichen wird also eine Ersatzdarstellung benötigt. In der XML-Spezifikation wurden daher die in Tab. 2.3-1 aufgeführten fünf Zeichenreferenzen, oft auch Entity-Referenzen genannt, vordefiniert.

Weitere Informationen über Zeichen-Entities erhalten Sie im Kapitel »Zeichen-Entities«, S. 59.

Zeichen	Zeichenreferenz	Erläuterung
<	<	Kleiner als (<i>less than</i>)
>	>	Größer als (<i>greater than</i>)
"	"	Doppeltes Anführungszeichen (<i>quotation mark</i>)
'	'	Einfaches Anführungszeichen (<i>apostrophe</i>)
&	&	Kaufmännisches Und (<i>ampersand</i>)

Tab. 2.3-1: In XML vordefinierte Zeichenreferenzen.

CDATA-Abschnitte

Innerhalb des getaggten Textes können so genannte **CDATA-Abschnitte** (*character data section*) eingefügt werden. CDATA-Abschnitte sind Abschnitte mit Zeichendaten, die nicht geparkt werden sollen. Das bedeutet, dass die Zeichen <, >, &, ", ' nicht als Begrenzungszeichen des Markups erkannt werden und in dieser Form erhalten bleiben.

CDATA-Abschnitte beginnen mit »<![CDATA[« und enden mit »]]>«. Die Zeichenfolge »]]>« darf in einem CDATA Abschnitt nicht vorkommen, da sie das Ende des CDATA-Abschnittes markieren würde.

Sinnvoll ist die Verwendung von CDATA-Abschnitten für Textabschnitte, in denen die Zeichen <, >, ", ' oder & häufig vorkommen. Dies ist zum Beispiel der Fall bei der Einbettung von JavaScript-Programmen. Ein anderer Anwendungsbereich ist die Einbettung von XML-Code in ein XML-Dokument, wenn der XML-Code wie »einfacher Text« behandelt und nicht geparkt werden soll.

```
<erklaerung>
<![CDATA[
<dozent> ist ein Start-Tag und
</dozent> ein Ende-Tag.
&lt; ist das Zeichen-Entity fur das
"<"-Zeichen.
]]>
</erklaerung>
```

Beispiel 6

Ohne CDATA-Abschnitt sähe dieser Abschnitt so aus:

```
<erklaerung>
  &lt;dozent&gt; ist ein Start-Tag und
  &lt;/dozent&gt; das Ende-Tag.
  &amp;lt; ist das Zeichenentity für das
  &quot;&lt;&quot;;-Zeichen.
</erklaerung>
```

Ausblick

Innerhalb eines CDATA-Abschnittes werden allgemeine Entities (siehe Kapitel »Allgemeine Entities«, S. 62) nicht ersetzt / aufgelöst.

Beispiel 7

Dieses Beispiel zeigt eine Dozentenliste in XML. In ihm werden die meisten der in diesem Wissensbaustein vorgestellten Konzepte verwendet.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Alle Dozenten -->
<dozentenliste>
  <dozent did="d1" anrede="Herr">
    <name>Maier</name>
    <vorname>Fritz</vorname>
    <bild ref="maier.jpg"/>
    <aktuelles>Nächste Woche findet die Sprechstunde
      am <em>Montag</em> statt.
    </aktuelles>
  </dozent>
  <dozent did="d2" anrede="Frau">
    <name>Müller</name>
    <vorname>Sabine</vorname>
    <bild ref="mueller.jpg"/>
    <aktuelles>Heute: Vortrag zum
      Thema &quot;Auslandsstudium&quot;
    </aktuelles>
  </dozent>
</dozentenliste>
```

2.4 Wohlgeformtes XML *

Wohlgeformtheit ist die Mindestvoraussetzung, damit XML-Dokumente weiterverarbeitet werden können. In der XML-Spezifikation wurden eine Reihe von Syntaxregeln festgelegt, die erfüllt sein müssen, damit ein XML-Dokument wohlgeformt ist. Dazu gehören u. a. »Attribute in Hochkom-

**mata setzen«, »Jeder Start-Tag braucht einen Ende-Tag«,
»Es darf nur nur ein Wurzelement geben«.**

Ein Dokument heißt **wohlgeformt** (*well-formed*), wenn es den Syntaxregeln und Wohlgeformtheitsbeschränkungen der XML-Spezifikation genügt.

Definition

HTML-Browser zeigen HTML-Dokumente an, auch wenn diese fehlerhaft sind. Eine entsprechende Fehlermeldung wird niemals angezeigt.

Ganz anders ist die Situation bei XML: Die XML-Spezifikation verbietet es XML-Parsern strikt, nicht wohlgeformte Dokumente zu korrigieren oder für eine Weiterverarbeitung bereit zu stellen. Wird ein Fehler gefunden, meldet der XML-Parser diesen und bricht den Parsingvorgang ab (Abb. 2.4-1).

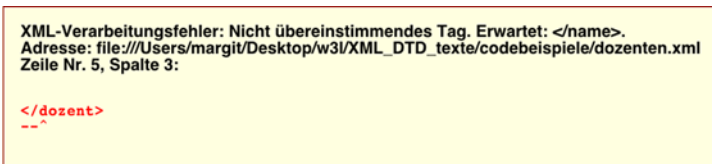


Abb. 2.4-1: Fehleranzeige im Browser.

Damit ein XML-Dokument wohlgeformt ist, müssen mehr als 100 Regeln befolgt werden. Im Folgenden werden die wichtigsten dieser Regeln für Wohlgeformtheit aufgeführt. Es wird jeweils ein Beispiel für eine Regelverletzung sowie die entsprechende Fehlermeldung des XML-Parsers Xerces angegeben.

- Jeder Start-Tag muss einen entsprechenden Ende-Tag haben.

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did="d1">
  <name>Maier
  <vorname>Fritz</vorname>
</dozent>
```

Fehlermeldung:

The element type "name" must be terminated by the matching end-tag "</name>".

Beispiel

Der Fehler ist klar: der Ende-Tag `</name>` fehlt.

- XML unterscheidet zwischen Groß- und Kleinschreibung.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did="d1">
  <name>Maier</name>
  <vorname>Fritz</vorname>
</Dozent>
```

Fehlermeldung:

The element type "dozent" must be terminated by the matching end-tag "</dozent>".

Da XML zwischen Groß- und Kleinschreibung unterscheidet, wird `</Dozent>` nicht als Ende-Tag zu `<dozent>` akzeptiert.

- Auch leere Elemente müssen explizit geschlossen werden. Statt `<element></element>` kann auch die Kurzform `<element/>` verwendet werden. Die Kurzform ist vorzuziehen.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did=d1>
  <name>Maier</name>
  <vorname>Fritz</vorname>
  
</dozent>
<br/>
```

Fehlermeldung:

The element type "img" must be terminated by the matching end-tag "".

Das Ende-Tag `` fehlt.

Korrekt ist:

```
</img>
oder in Kurzform: 
```

- Elemente dürfen verschachtelt sein, sich aber nicht überlappen.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did="d1">
```

```
<name>Maier<vorname>Fritz</name></vorname>
</dozent>
```

Fehlermeldung:

The element type "vorname" must be terminated by the matching end-tag "</vorname>".

Die Elemente name und vorname überlappen sich. Der Start-Tag <name> steht vor dem Start-Tag <vorname>, jedoch steht der Ende-Tag </name> vor dem Ende-Tag </vorname>-Tag statt dahinter.

- Ein XML-Dokument muss genau ein Wurzelement besitzen, das alle anderen Elemente vollständig enthält.

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did="d1">
  <name>Maier</name>
  <vorname>Fritz</vorname>
</dozent>
<dozent did="d2">
  <name>Müller</name>
  <vorname>Sabine</vorname>
</dozent>
```

Beispiel

Fehlermeldung:

The markup in the document following the root element must be well-formed.

Sollen zwei oder mehr <dozent>-Elemente im Dokument erfasst werden, muss ein weiteres Element z. B. <dozentenliste> als Wurzelement in das XML-Dokument eingefügt werden.

Korrekt ist also:

```
<?xml version="1.0" encoding="UTF-8"?>
<dozentenliste>
  <dozent did="d1">
    <name>Maier</name>
    <vorname>Fritz</vorname>
  </dozent>
  <dozent did="d2">
    <name>Müller</name>
    <vorname>Sabine</vorname>
  </dozent>
</dozentenliste>
```

- Attributwerte müssen immer in Anführungszeichen (doppelte oder einfache) stehen.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<dozent did=d1>
  <name>Maier</name>
  <vorname>Fritz</vorname>
</dozent>
```

Fehlermeldung:

Open quote is expected for attribute "did" associated with an element type "dozent".

Der Fehler ist klar:

Die Hochkommata beim Attributwert des Attributes did fehlen.

- Ein Element darf nicht zwei Attribute mit dem gleichen Namen besitzen.

Beispiel

```
<dozent did="d1" anrede="Herr" anrede="Dr.">
  <name>Maier</name>
  <vorname>Fritz</vorname>
  
</dozent>
```

Fehlermeldung:

Attribute "anrede" was already specified for element "dozent".

- Kommentare und Verarbeitungsanweisungen dürfen nicht in Tags auftreten.

Beispiel

```
<dozent did="d1" <!--Beginn Dozent-->>
  <name>Maier</name>
  <vorname>Fritz</vorname>
  
</dozent>
```

Fehlermeldung:

Element type "dozent" must be followed by either attribute specifications, ">" or "/>".

Weiterhin gilt, dass die aus zwei Bindestrichen bestehende Zeichenkette (--) nicht innerhalb eines Kommentars vorkommen darf und dass Kommentare nicht verschachtelt werden dürfen.