

```

<!ELEMENT Buch (BuchEinfach|BuchIntern)>
<ENTITY %Bibliotheksbenutzer "INCLUDE">...
<ENTITY %Bibliothekspersonal "IGNORE">...

```

Wird nun von diesem XML-Dokument eine externe DTD referenziert, kann in dieser anstelle der direkten Angabe von IGNORE und INCLUDE entsprechend der Vorgabe aus dem XML-Dokument ein Entity-Bezeichner eingesetzt werden (siehe Abb. 14.9). Ein XML-Dokument

```

<![%Bibliotheksbenutzer
[<!ELEMENT BuchEinfach (#PCDATA)>
<ATTLIST BuchEinfach
Titel CDATA #REQUIRED
id IDREF #REQUIRED>
]]>
<![%Bibliothekspersonal
[<!ELEMENT BuchIntern (#PCDATA)>
<ATTLIST BuchIntern
isbn ID #REQUIRED
ausgeliehen CDATA #REQUIRED>
]]>

```

Abb. 14.9. XML-DTDs – Bedingte Deklarationen und Parameter Entities

für Bibliotheksbenutzer wäre demnach lediglich in der Lage, Elemente vom Typ Bibliotheksbenutzer zu verwenden.

Ein entscheidender Nachteil der DTDs liegt darin, daß sich die verwendete Syntax von der XML-Syntax unterscheidet. DTDs sind stets abgeschlossen, eine Erweiterung ist lediglich sehr eingeschränkt über Parameter-Entities möglich. Daher wurden vom W3C **XML-Schemata** (siehe Abschnitt 14.3.2) vorgeschlagen, die diese Nachteile umgehen. Trotz allem sind DTDs weit verbreitet und selbst zur Grundlage verschiedener Spezifikationen vielfältiger Markup-Sprachen geworden.

Weiterführende Literatur:

T. Bray, J. Paoli, C. M. Sperberg-McQueen: Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998, available at <http://www.w3.org/TR/1998/REC-xml-19980210>

C. F. Goldfarb, P. Prescod: The XML-Handbook, 2nd. ed., Prentice Hall, Upper Saddle River NJ, USA, 2000.

14.3.2 XML-Schemata

Um die aufgezeigten Nachteile der XML-DTDs wettzumachen, wurde vorgeschlagen, Dokumententyp-Deklarationen für XML-Dokumente mit Hilfe der XML-Syntax zu definieren. Die Gegenüberstellung einer einfachen DTD und des entsprechenden XML-Schemata macht deutlich, wie einfach ein XML-Schema verstanden werden kann (siehe Abb. 14.10).

Wie die Deklaration von XML-Elementen mit Hilfe von XML-Schemata vollzogen werden kann, soll im vorliegenden Abschnitt erläutert werden.

XML-DTD:

```

<!ELEMENT Buch (Titel, Subtitel?, Autor+, isbn, preis)>
<!ELEMENT Titel (#PCDATA)>
<!ELEMENT Subtitel (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT Preis (#PCDATA)>

```

XML-Schema:

```

<Schema ... >
  <element name="Buch">
    <type>
      <element name="Titel" type="string"/>
      <element name="Subtitel" type="string"
        minOccurs="0" maxOccurs="1"/>
      <element name="Autor" type="string" minOccurs="1"/>
      <element name="isbn" type="string"/>
      <element name="preis" type="string"/>
    </type>
  </element>
</Schema>

```

Abb. 14.10. Gegenüberstellung einer einfachen XML-DTD und eines XML-Schemas

Ein XML-Schema ist dabei nichts anderes, als ein spezielles XML-Dokument, das eigene Elemente zur Beschreibung neuer Dokumententypen enthält. Das XML-Dokument mit der Schema-Deklaration enthält das Wurzel-Element **schema**, das alle weiteren Schema-Elemente als Nachfolge-Elemente beinhaltet. Jedes XML-Schema muß mit einer Präambel starten, die optionale Definitionen und Deklarationen zum vorliegenden Dokument enthält. Die Präambel steht im **schema**-Wurzel-Element und verfügt über folgende Attribute:

- **targetNS:** spezifiziert den Namensraum und den URI des verwendeten Schemas
- **version:** Versionsnummer des Schemas
- **xmlns:** Namensraum der XML-Schemata-Spezifikation

Grundlegender Bestandteil eines XML-Schemas sind sogenannte **Typen**. Während einfache Datentypen über **elementare Typdefinitionen** spezifiziert werden, werden Elemente und Attribute eines XML-Dokuments innerhalb einer komplexen Typdefinition (**<type>**) wie in Abb. 14.10 gezeigt vorgenommen. Ein elementarer Datentyp, wie z.B. **KleineZahlen**, wird wie folgt deklariert:

```

<datatype name="KleineZahlen" source="integer"/>
  <!minExclusive value= "0"/>
  <!maxExclusive value= "12"/>
</datatype>

```

Dabei wird ein neuer Datentyp **KleineZahlen** definiert, der auf dem Grundtyp **integer**, also ganze Zahlen, beruht und dessen Wertebereich zwischen 0 und 12 liegt. Der Datentyp, auf dem der in der elementaren Typdefinition deklarierte Typ basiert, wird als **primitiver Datentyp** bezeichnet. In Tabelle 14.4 sind die verfügbaren primitiven Datentypen zusammengestellt.

Tabelle 14.4. Primitive Datentypen für XML-Schemata

| Datentyp | Bedeutung |
|-------------------------------|---|
| <code>string</code> | endliche Folge von Unicode-Zeichen |
| <code>boolean</code> | die booleschen Werte (<code>true</code> , <code>false</code>) |
| <code>float</code> | 32 Bit Fließkommazahl mit einfacher Genauigkeit |
| <code>double</code> | 64 Bit Fließkommazahl mit doppelter Genauigkeit |
| <code>decimal</code> | Fließkommazahl in Dezimaldarstellung |
| <code>timeInstant</code> | String, der eine Zeitangabe der Form <code>jjjj-mm-ttt hh:mm:ss.ssss</code> beinhaltet, gefolgt von einem <code>Z</code> , das eine Zeitangabe in Coordinated Universal Time beschreibt bzw. der Zeitdifferenz der aktuellen Zeitzone als <code>+/-mhh:mm</code> |
| <code>timeDuration</code> | String, der eine Zeitdauer beschreibt. Der String startet mit einem <code>P</code> , dem ein Vorzeichen vorangestellt wird, worauf sich jeweils numerische Werte und Bezeichner für die Zeitintervalle (<code>Y</code> =Jahr, <code>M</code> =Monat, <code>D</code> =Tag, <code>H</code> =Stunden, <code>M</code> =Minuten und <code>S</code> =Sekunden) anschließen, z.B. beschreibt <code>+P3Y2M13DT12H0M0S</code> eine positive Zeitdifferenz von 3 Jahren, 2 Monaten, 13 Tagen und 12 Stunden. |
| <code>recurringInstant</code> | definiert einen periodisch wiederkehrenden Zeitpunkt und folgt dem Format von <code>instantTime</code> . Ausgelassene Details müssen durch Bindestriche („-“) ergänzt werden. |
| <code>binary</code> | Binärdaten beliebiger Länge |
| <code>uri</code> | URI |

Aus den primitiven Datentypen lassen sich generierte Datentypen und benutzerdefinierte Datentypen zusammensetzen. Der primitive Datentyp, auf den dabei aufgebaut wird, heißt **Basistyp**. So lassen sich z.B. aus dem Basistyp `decimal` generierte Typen wie `integer`, `positive-integer` oder `non-positive-integer` durch Restriktion des Definitionsbereichs ableiten. Die Definition von Element-Attributen erfolgt über das `<attribute>`-Tag. Neben einer Typangabe für das betreffende Attribut kann ebenfalls die Kardinalität des Attributs über die Parameter `minOccurs` und `maxOccurs` festgelegt werden, die angeben, wie oft ein Attribut vorkommen darf. Vorgabewerte können über das Schlüsselwort `default` angegeben werden, die Bedeutung der Schlüsselworte `fixed` und `implied` ist analog zur XML-DTD. Beispiel:

```
<attribut name="persnr"type=" integer" default="0"/>
```

Um ein XML-Element deklarieren zu können, muß die Möglichkeit bestehen, das Inhaltsmodell des Elements zu beschreiben. XML-Schemata erlauben dabei eine differenziertere Beschreibung von XML-Elementen, als dies

mit DTDs möglich ist. Das Attribut `content` legt die Art des Inhalts des Dokuments fest:

- `content=unconstrained`: Inhalt beliebiger Art
- `content=empty`: leeres Element
- `content=mixed`: Subelemente und Character Data

Zur Deklaration eines Elements dient das `<element>`-Tag, dem über das Attribut `name` ein Name zugewiesen wird. Einfache Elemente bestehen lediglich aus einer Referenz auf den Datentyp des Elements und einer Reihe von Attribut-Deklarationen. Beispiel:

```
<element name="Rezept" type="string"/>
<element name="Gewicht" type="float"/>
```

Die Möglichkeit, eine feste Reihenfolge (Sequenz) oder eine Auswahl zwischen verschiedenen Sub-Elementen als Bestandteil einer Element-Deklaration zu schaffen, bietet ein spezielles Konstrukt, der sogenannte **Compositor**. Der Compositor kommt stets in einem `<group>`-Tag unter dem Attribut `order` zur Anwendung. Man unterscheidet dabei:

- `order=seq`: definiert eine Sequenz von Sub-Elementen und entspricht den Kommata in DTDs,
- `order=Choice`: nur genau eines der angegebenen Subelemente darf ausgewählt werden (entspricht dem „|“ in der DTD).

Beispiele für die beiden Compositoren sind in Abb. 14.11 zusammen mit ihrem DTD-Äquivalent dargestellt.

Das `<group>`-Tag kann auch zur Referenzierung und Wiederverwendung von Elementen oder Attributen in der Deklaration von Elementen verwendet werden. Dazu wird eine Gruppe von Elementen oder Attributen deklariert, die mit einem Namen (`name`) versehen wird, über den diese an späterer Stelle referenziert werden kann (siehe Abb. 14.12).

Das Schlüsselwort `any` kann in einer Deklaration als Repräsentant für beliebige Konstrukte stehen, solange diese den folgenden Regeln entsprechen:

- beliebige wohlgeformte XML-Konstrukte,
- beliebige wohlgeformte Element-Konstrukte, die einem anderen Namensraum angehören müssen als dem, in dem das `any`-Element gesetzt wurde,
- beliebige wohlgeformte Element-Konstrukte, die aus einem bestimmten, vorgegebenen Namesraum stammen,
- beliebige wohlgeformte Element-Konstrukte, die dem aktuell verwendeten Namesraum entstammen.

Soll ein beliebiges Attribut über `any` angegeben werden, so erfolgt dies über das Schlüsselwort `anyAttribute`. Abbildung 14.13 zeigt eine Element-Deklaration, die ein beliebiges Attribut aus dem angegebenen Namensraum enthalten darf.

Sequenz:

```

<element name="Obst"/>
  <type>
    <group order="seq">
      <element type="Apfel"/>
      <element type="Birne"/>
      <element type="Melone"/>
    </group>
  </type>
</element>

<!ELEMENT Obst (Apfel, Birne, Melone)>

```

Selektor:

```

<element name="Heißgetränk"/>
  <type>
    <group order="Choice">
      <element type="Tee"/>
      <element type="Kaffee"/>
    </group>
  </type>
</element>

<!ELEMENT Heißgetränk (Tee|Kaffee)>

```

Abb. 14.11. XML-Schemata - Elemente-Definition mit Compositoren**Sequenz:**

```

<group name="Belag" order="Choice">
  <element type="Erdbeeren"/>
  <element type="Himbeeren"/>
  <element type="Brombeeren"/>
</group>

<element name="Obstkuchen"/>
  <type>
    <group ref="Belag"/>
    <attribute name="durchmesser" type="integer"/>
  </type>
</element>

```

Abb. 14.12. XML-Schemata - Wiederverwendung von Modellgruppen

```

<element name="Obstkuchen"/>
  <type>
    <element name="Belag" type="Obst"/>
    <anyAttribute namespace="http://www.test.de/XMLSchema"/>
  </type>
</element>

```

Abb. 14.13. XML-Schemata - Verwendung beliebiger Repräsentanten von Elementen oder Attributen

Im Gegensatz zu DTDs bieten XML-Schemata die Möglichkeit, bestehende Deklarationen zu erweitern oder auch einzuschränken. Zu diesem Zweck wird das Attribut `derivedBy` verwendet. Von einer Erweiterung eines Typs spricht man, wenn einer bestehenden Deklaration neue Eigenschaften hinzugefügt werden. Das Beispiel in Abb. 14.14 zeigt, wie aus der Typ-Deklaration des Elements `obstkuchen` ein neuer Typ `obsttorte` abgeleitet wird, der ein zusätzliches Sub-Element `glassur` zusammen mit den aus `obstkuchen` stammenden Sub-Elementen enthält. Daß sich der neue Typ `obsttorte` auf den Typ `obstkuchen` bezieht, wird durch das Attribut `source` kenntlich gemacht, das als Wert den Namen des grundlegenden Typs enthalten muß. Die Erweiterung des bestehenden Typs wird schließlich durch das Attribut `derivedBy` angegeben, dem der Wert `extension` zugewiesen wird.

```
<type name="obstkuchen">
  <element name="Belag" type="Obst"/>
  <attribut durchmesser type="float"/>
</type>

<type name="obsttorte" source="obstkuchen" derivedBy="extension">
  <element name="glassur" type="string"/>
</type>
```

Abb. 14.14. XML-Schemata - Ableiten von Typdefinitionen durch Erweiterung

Soll ein Typ eingeschränkt werden, so erfolgt dies in identischer Weise, unter Angabe der Sub-Elemente, auf die der ursprüngliche Typ eingeschränkt werden soll und der Wertzuweisung `derivedBy="restriction"`.

Eine weitere Besonderheit der XML-Schemata, die diese gegenüber den DTDs auszeichnet, ist die Möglichkeit, Schemata miteinander zu kombinieren. Die **Komposition** von Schemata kann dabei über die beiden Elemente `<import>` und `<include>` erfolgen.

- `<import>`
Dient der Einbindung eines Schemas aus einem alternativen Namensraum. Über das Attribut `namespace` erfolgt die Angabe des betreffenden Namensraums und `schemaLocation` gibt den URI des verwendeten XML-Schemas an.
- `<include>`
Die Einbindung eines externen XML-Schemas über `<include>` erfordert die Angabe des Attributs `schemaLocation`, das den URI des betreffenden Schemas angibt. Beide Schemata, das eingebundene sowie das eigene müssen über einen identischen Attributwert `targetNamespace` verfügen.

Kommentare können in XML-Schemata über das Element `<annotation>` spezifiziert werden. Dabei kann das Element `<annotation>` selbst zwei verschiedene Subelemente beinhalten:

- `<info>`: für menschliche Leser,
- `<appinfo>`: für Anwendungen, die das XML-Schema interpretieren.

14.3.3 XML Information Set

Ein XML-Dokument kann man als ein aus vielen verschiedenen Bestandteilen – unterschieden nach ihrer Semantik – zusammengesetztes Konstrukt begreifen. Die Bestandteile, oder besser **Objekte**, sind miteinander verknüpft und erlauben die Anwendung verschiedener Mechanismen zur Navigation, Verknüpfung und Suche in einzelnen Dokumenten oder in Dokument-übergreifenden Suchräumen. Das W3C hat die einzelnen Bestandteile, die ein wohlgeformtes XML-Dokument ausmachen, im sogenannten **XML Information Set**, oder kurz **Infoset**, zusammengefaßt und beschrieben. Die einzelnen Objekttypen des Infoset werden als **Informationstypen** bezeichnet. Ein Parser, der ein XML-Dokument interpretiert, muß über die Spezifikation der Informationstypen verfügen, um alle angegebenen Informationen semantisch korrekt interpretieren zu können.

Insgesamt enthält das XML-Infoset 15 verschiedene Informationstypen:

- **Dokument-Informationstyp**

Jedes wohlgeformte XML-Dokument muß stets genau eine Instanz des Dokument-Informationstyps enthalten. Diese Instanz beinhaltet alle Informationen über das gesamte Dokument:

- Eine Liste (**children**) der Instanzen der Kind-Elemente in der Reihenfolge ihres Auftretens im Original-Dokument. Diese Liste muß mindestens aus einer Instanz des Element-Informationstyps bestehen. Zusätzlich muß diese Liste Informationen über alle Verarbeitungsanweisungen enthalten, die außerhalb des Wurzel-Elements eines Dokuments vereinbart wurden. Darüberhinaus kann die Liste auch noch Informationen über optionale Informationstypen, wie z.B. Kommentare oder Dokumenttyp-Deklarationen enthalten.
- Für alle im Dokument definierten Notationen muß eine Instanz des Notations-Informationstyps in einer ungeordneten Liste (**notations**) angegeben werden.
- Für alle nicht vom Parser zu interpretierenden Entities muß eine Instanz des Entity-Informationstyps in einer ungeordneten Liste angegeben werden (**unparsed entites**).
- Optionale Angabe eines URI (**base uri**) für das vorliegende Dokument.

- **Element-Informationstyp**

Für jedes Element eines XML-Dokuments muß eine Instanz des Element-Informationstyps vorhanden sein, der folgende Informationen umfassen kann:

- Für alle Elemente, Verarbeitungsanweisungen, Referenzen auf ausgelassene Entites und Zeichen müssen die entsprechenden Informationstypen in der Reihenfolge, in der diese im XML-Dokument auftreten, in Form

WWW

Kommunikation, Internetworking, Web-Technologien

Meinel, C.; Sack, H.

2004, XLII, 1178 S. In 2 Bänden, nicht einzeln erhältlich.,

Hardcover

ISBN: 978-3-540-44276-9