

XML

Web Services kann auf die Formel „**Web Services = XML über HTTP + Java mit Standardprotokollen**“ reduziert werden. Java ist ein Bestandteil, um plattformunabhängige Programme schreiben zu können. XML ist dazu das Pendant auf Seite der Daten.

In diesem Abschnitt wird XML anwendungsbezogen erläutert. Eine Spezifikation ist zu finden unter <http://www.w3.org/TR/1998/REC-xml-19980210> und als deutsche Übersetzung <http://www.edition-w3c.de/TR/1998/REC-xml-19980210.html>.

Die Ausarbeitung basiert auf „[XML in der Praxis](#)“, Hennig Behme, Stefan Mintert.

Inhalt

1. Einleitung	4
2. Erste Schritte	8
3. XML Beschreibung	11
3.1. Dokumente	11
3.2. Elemente	13
3.3. Zeichen, Namen und Zeichendaten	16
3.4. Kommentare.....	20

3.5. Processing Instructions	22
3.6. Einbeziehen von Binärdaten	23
3.7. Dokumenttyp-Definition (DTD)	24
3.7.1. Elementtyp-Deklaration	25
3.7.2. Attributlisten-Deklaration	30
3.7.3. Möglichkeiten, die DTD zu gestalten und zu gliedern	32
3.7.4. Notation-Deklaration	35
4. XML: Linking	37
4.1. XLink	37
4.2. XPointer	44
4.2.1. Absolute Verweisausdrücke	45
4.2.2. Relative Verweisausdrücke	46
4.2.3. Verweisausdrücke für Bereiche	48
4.2.4. Attributverweisausdrücke	50
4.2.5. Stringverweisausdrücke	50
5. XSL	52

1. Einleitung

XML, die erweiterbare Auszeichnungssprache (**eXtensible Markup Language**) wurde entwickelt, um eine Auszeichnungssprachen für Dokumente, die im World Wide Web zur Veröffentlichung kommen sollen, verfügbar zu haben.

Im Vergleich zu HTML bietet XML deshalb Möglichkeiten, weil es eine **Metasprache** ist. **HTML** ist eine **Strukturbeschreibung für Web-Dokumente**. **XML** hingegen ist als Teilmenge von SGML eine Sprache, in der wiederum sich beliebig viele Sprachen (wie z.B. HTML) definieren lassen.

Ähnlich HTML lassen sich XML-Daten »on the fly« aus Datenbankbeständen erzeugen. Angesichts der im Gegensatz zu HTML »richtig« geschachtelten Elemente eignet sich diese Auszeichnungssprache besser zur Speicherung in **Datenbanken**, vor allem in objektorientierten, die eine komplexe Elementstruktur leichter abbilden können als relationale. Die **Elemente in XML**-Dokumenten müssen **strikt** ineinander **geschachtelt** sein. Daher kann man sie als Einheiten (Objekte) betrachten, die sich samt ihrer Struktur in einem objektorientierten Datenbanksystem abbilden lassen.

Mit **XML** ein einheitliches (universelles) **Datenformat** verfügbar, das sowohl **maschinenlesbar** als auch für **menschliche Augen verständlich** ist. Somit dient XML als Basis für Schnittstellen, bei denen unterschiedliche Anwendungen auf unterschiedlichen Plattformen Daten austauschen.

Die folgenden Punkte umschreiben die Designziele der Entwickler 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 soll einfach sein, **Programme** zu schreiben, die **XML-Dokumente verarbeiten**.
5. Die Anzahl optionaler Merkmale in XML soll minimal sein, idealerweise Null.
6. XML-Dokumente sollten für **Menschen lesbar** und angemessen verständlich sein.
7. Der Entwurf von XML soll **formal** und präzise sein.
8. XML-Dokumente sollen leicht zu erstellen sein.
9. Knappheit von XML-Markup ist von minimaler Bedeutung.

Ein Beispiel verdeutlicht diese Ziele:

```
<mitarbeiter id="firma.abteilung.4725721.r238-4523/003"
             eintritt="01011976"
             lastmod="04021997"
             lastmodby="firma.abteilung.31296.261263c-3461/001">
  <name>Meier</name>
  <vorname>Heinz</vorname>
  <gebdatum>
    <jahr>1938</jahr>
    <monat>10</monat>
    <tag>29</tag>
  </gebdatum>
  <!-- viele weitere Daten -->
</mitarbeiter>
```

Für jeden Mitarbeiter existiert ein mit seinen Detaildaten gefülltes Element `MITARBEITER`. Die Attribute enthalten Metadaten wie eine Kennnummer, das Eintrittsdatum sowie, wann und wie zum letzten Mal eine Veränderung vorgenommen worden ist.

Für Menschen lesbar sind hier auf jeden Fall die Element- und Attributnamen; selbst mit dem Inhalt kann man etwas anfangen. Die Struktur der Kennnummer dagegen ist schon in diesem Beispiel eher etwas für Maschinen.

Erkennbar ist, dass alle Elemente geschlossen sind, d.h. dass jede Elementinstanz außer dem **Start-Tag** auch ein **End-Tag** haben muss.

XML beschreibt Dokumente. Dokumente bestehen aus

- Inhalt,
- Formatierung,
- Struktur und
- logischen Informationen.

Bei einem Vorlesungsskript besteht der Inhalt aus Text, Bildern, Programmcode. Die Formatierung besteht aus der gewählten Schriftart usw. und bestimmt das visuelle Erscheinungsbild. Die Struktur des Texts, also die Aufteilung in Kapitel, Abschnitte usw. Die logische Information beschreibt, ob es sich um ein Zitat oder um eine wichtige Textstelle handelt.

Während **HTML Inhalt** und **Layout** beschreibt, ist **XML** geeignet, neben dem **Inhalt**, die **Struktur** und **logische Informationen** von Dokumenten zu beschreiben.

2. Erste Schritte

XML **trennt Inhalt** und **Struktur** von Dokumenten. Die **Struktur** wird in einer Document Type Definition (**DTD**) beschrieben. Eine DTD beschreibt den strukturellen Aufbau und die logischen Elemente einer Klasse von Dokumenten, den so genannt **Dokumenttyp**.

Als ein Beispiel für einen sehr einfachen Dokumenttyp betrachten wir E-Mails. Im Wesentlichen besteht eine E-Mail aus

- einem Empfänger,
- einem Absender,
- einem Thema (Betreff oder »Subject«) sowie
- der eigentlichen Nachricht.

Eine DTD für E-Mails in XML hat folgendes Aussehen:

```
<!-- E-Mail-DTD Version 1 -->
<!ELEMENT email      (empfaenger, absender,
                      thema, nachricht) >
<!ELEMENT empfaenger (#PCDATA)>
<!ELEMENT absender   (#PCDATA)>
<!ELEMENT thema      (#PCDATA)>
<!ELEMENT nachricht  (#PCDATA)>
```

Eine `email` besteht aus einem `empfaenger`, einem `absender`, einem `thema` sowie einer `nachricht`, und zwar in dieser Reihenfolge. Alle diese Elemente enthalten ihrerseits ausschließlich reinen Text, hier **Parsed Character Data (PCDATA)** genannt.

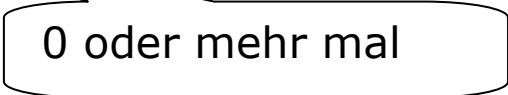
Eine E-Mail, die zum o.a. Dokumententyp gehört hat dann folgendes Aussehen in XML:

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "email1.dtd">
<email>
  <empfaenger>Veronika Feldbuch</empfaenger>
  <absender>Alois Schuette</absender>
  <thema>Vorlesung Web Services</thema>
  <nachricht>
    Hallo Veronika,
    in der Veranstaltung Web Services werden Telefondienste besprochen.
    Wenn Du Lust hast, kannst Du einen Vortrag halten.
    Gruss
  </nachricht>
</email>
```

Mittels DTD kann ein XML Parser prüfen, ob ein XML Dokument richtig aufgebaut ist.

Will man **mehrere Elemente** (ev. auch keine) erlauben, z.B. „KopieAn“, so ist dies in der DTD mit einem „*“ zu kennzeichnen.

```
<!-- E-Mail-DTD Version 2 -->
<!ELEMENT email      (empfaenger, kopieAn*, absender, thema, nachricht)>
<!ELEMENT empfaenger (#PCDATA)>
<!ELEMENT kopieAn    (#PCDATA)>
<!ELEMENT absender   (#PCDATA)>
<!ELEMENT thema      (#PCDATA)>
<!ELEMENT nachricht  (#PCDATA)>
```



Damit ist die vorige E-Mail korrekt, aber auch folgende:

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "email2.dtd">
<email>
  <empfaenger>Veronika Feldbuch</empfaenger>
  <kopieAn>JL</kopieAn>
  <kopieAn>ps</kopieAn>
  <absender>Alois Schuette</absender>
  <thema>Vorlesung Web Services</thema>
  <nachricht>
    Hallo Veronika,
    in der Veranstaltung Web Services werden Telefondienste besprochen.
    Wenn Du Lust hast, kannst Du einen Vortrag halten.
    Gruss
  </nachricht>
</email>
```

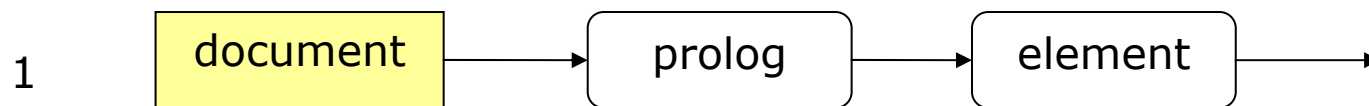
3. XML Beschreibung

Die Beschreibung von XML erfolgt anwendungsorientiert. Die Syntax wird mit Syntaxdiagrammen verdeutlicht, wobei die Nummer der Regel sich auf die Original-[Spezifikation](#) bezieht.

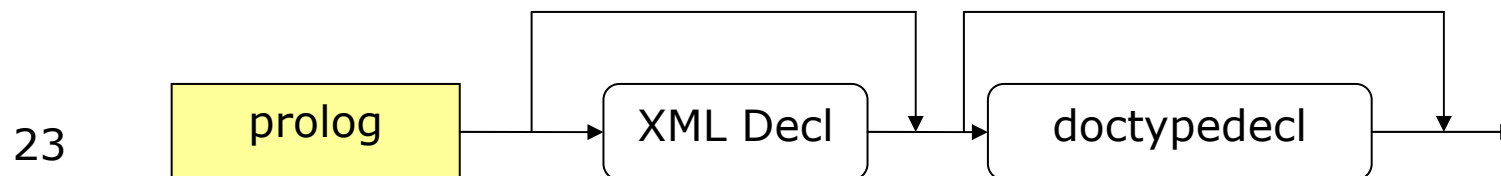
3.1. Dokumente

Das zentrale Objekt ist das Dokument. XML-Daten sind reine **Textdateien**, die eine Struktur besitzen, die durch eine Sprache definiert ist.

Eine Regel für ein **Dokument** sieht wie folgt aus:



Der **Prolog** zeigt an, dass es sich um ein XML-Dokument handelt.



Der `prolog` [23] kann optional leer sein, besitzt aber üblicherweise mindestens folgende Form:

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

Diese Zeile ist die **XML-Deklaration**. Falls notwendig, so enthält sie auch noch Informationen über die Kodierung des Dokuments:

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

Die Versionsnummer bezieht sich auf die Version der XML-Spezifikation. Zur Zeit darf dort nur die Nummer »1.0« stehen. In Zukunft kann es auch weitere Möglichkeiten geben, je nachdem, wie sich die Weiterentwicklung der Sprache gestaltet.

Wenn sich ein **Dokument** an die Regeln der Spezifikation hält, so nennt man es **wohlgeformt**, es ist also von einem **XML Parser** als **syntaktisch korrekt** erkennbar.

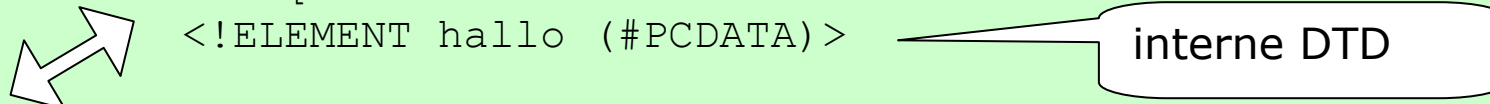
Neben der Wohlgeformtheit gibt es eine zweite Eigenschaft, die ein Dokument auszeichnen kann, die **Gültigkeit**. Bei einem gültigen Dokument muss im Prolog eine DTD angegeben werden, und zwar in der **Dokumenttyp-Deklaration**. Sie kann entweder extern oder intern vorhanden sein..

```
<?xml version="1.0"?>  
<!DOCTYPE hallo SYSTEM "hallo.dtd">  
<hallo>Hallo Welt!</hallo>
```



externe DTD

```
<?xml version="1.0"?>
<!DOCTYPE hallo [
  <!ELEMENT hallo (#PCDATA)>
]>
<hallo>Hallo Welt!</hallo>
```



Wenn das Dokument eine Struktur besitzt, die durch die DTD beschrieben wird, dann handelt es sich um eine **gültige Instanz**. Eine Voraussetzung dafür ist, dass es sich bei dem in Regel [1] gezeigten `element` um das **Wurzelement** aus der DTD handelt. Im obigen Beispiel ist dies das Element `hallo`.

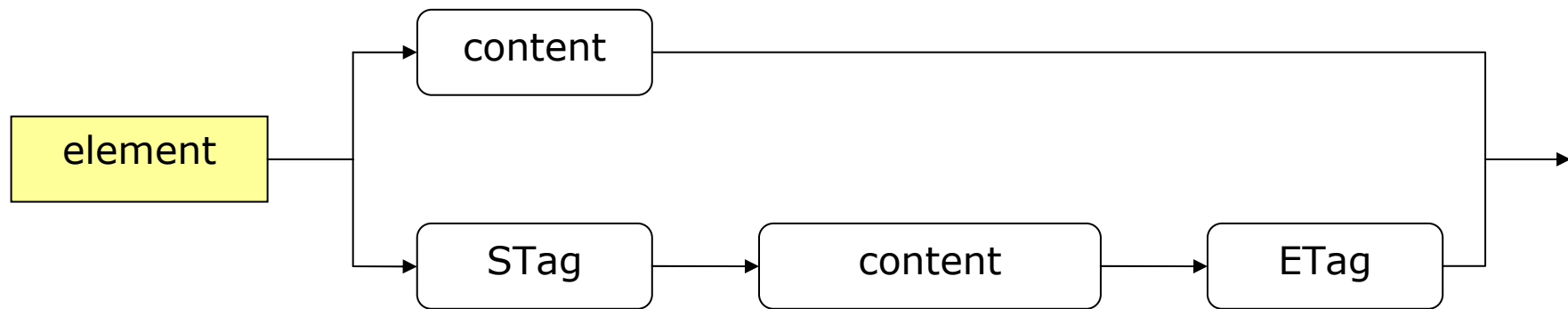
3.2. Elemente

Elementen können mit oder ohne Inhalt sein. Beispiele für beide sind auch in HTML bekannt:

```
<P>Dies ist ein Absatz in HTML. Er enthält Textzeichen
als Inhalt, kann aber auch <EM>weitere Elemente</EM>
aufnehmen.</P>
<P>Im Gegensatz dazu ist ein IMG-Element leer. Es
verwendet Attribute, um eine Grafikdatei zu
laden. <IMG SRC="bild.gif"> Doch Vorsicht:
In XML besitzen leere Elemente eine etwas andere
Schreibweise.
</P>
```

Folgende Abbildung zeigt das Syntax-Diagramm für ein `element`.

39



Elemente können **Attribute** besitzen. Im obigen HTML-Text ist SRC ein Beispiel für ein solches Attribut.

Elemente werden durch einen **Start-Tag** (STag [40]) eingeleitet. Dieser besteht aus dem Namen des Elements, eingeschlossen von spitzen Klammern.

Dem Start-Tag folgt der **Inhalt** (content) des Elements. Woraus dieser bestehen darf, d.h. welche anderen Elemente oder ob Text darin vorkommen darf, wird in der DTD festgelegt. Elemente in Instanzen ohne DTD - d.h. wohlgeformte, aber nicht gültige XML-Dokumente - dürfen Zeichen-
daten, weitere Elemente, Kommentare usw. enthalten. Die genaue formale Beschreibung enthält die Regel [43] content im nächsten Teil.

Den Abschluß des Elements markiert der **End-Tag** (ETag [42]). Es hat die Form `</elementname>`.

Folgende Beispiele sind **wohlgeformte** Elemente.

```
<p class="anrede">Sehr geehrte Damen und Herren!</p>
<witz>
  Treffen sich zwei XML-Experten,
  sagt der eine:
  <woertliche-rede>
    Guten Tag!
  </woertliche-rede>
</witz>
```

The diagram illustrates the structure of the XML snippet. It shows three callout boxes: 'STag' pointing to the opening tag `<woertliche-rede>`, 'content' pointing to the text 'Guten Tag!', and 'ETag' pointing to the closing tag `</woertliche-rede>`.

Damit die **Wohlgformtheit** erhalten bleibt, sind einige Punkte zu beachten:

- Kein Attributname (z.B. `class`) darf mehr als einmal in einem Element verwendet werden.
- Ein Attributwert (z.B. `anrede`) darf keinen Verweis auf ein externes Entity enthalten. Für den Augenblick genügt es zu wissen, dass Teile eines XML-Dokumentes auf mehrere Dateien verteilt werden können. Ein solcher externer Text darf nicht im Attributwert verwendet werden.
- Ein Attributwert darf keine öffnende spitze Klammer `<` enthalten.

Für **gültige** Dokumente gilt darüber hinaus, dass Attribute in der DTD zu deklarieren sind und dass die Wertetypen im Dokument mit dem in der DTD deklarierten Typ übereinstimmen.

Zur Veranschaulichung dienen die folgenden Beispiele, die **fehlerhafte** Elemente zeigen.

```
<kapitel name="einleitung"  
      name="grundlagen">
```

Fehler: Gleicher Attributname (name) mehrfach verwendet
</kapitel>

```
<if vergleich="x < y">
```

Fehler: Spitze Klammer im Attributwert
</if>

Leere Elemente

In HTML besitzt ein leeres Element die Form eines Start-Tags. Das ist in XML **anders**. Der Unterschied besteht darin, dass hinter dem Namen nicht die einfache spitze Klammer steht, sondern die Zeichenfolge `</>`. Ein Element wie `img` in HTML besitzt in XML folgende Schreibweisen, die beide zulässig sind.

```
  
</img>
```

der \ könnte in HTML entfallen, **nicht** in XML

3.3. Zeichen, Namen und Zeichendaten

Innerhalb von (nicht leeren) Elementen dürfen Zeichendaten (character data) stehen, insofern die DTD nichts anderes definiert.

Der von XML verwendete **Zeichensatz** ist **ISO/IEC 10646**. Dadurch sind insbesondere die westlichen Sprachen abgedeckt, was für deutsche Texte heißt, man aller Sonderzeichen direkt eingegeben werden. Nicht **darstellbare Zeichen** können in Form einer so genannten **Zeichenreferenz** eingegeben werden. Die Zeichenreferenz ist die Nummer (in dezimaler

referenz eingegeben werden. Die Zeichenreferenz ist die Nummer(in dezimaler Notation) des Zeichens im Zeichensatz. Diese Nummer wird zwischen `&#` und `;` eingeschlossen (oder die hexadezimale Notation durch `&#x` und `;`).

```
Sie sagte: &#xbb;Web Services sind zukunftsstrchtig!&#xab;
```

```
Sie sagte: &#187;Web Services sind zukunftsstrchtig!&#171;
```

```
Sie sagte: >Web Services sind zukunftsstrchtig!<<
```

Die **Metazeichen** von XML mssen in ihrer **Ersatzdarstellung** eingegeben werden.

Dies betrifft

- die spitzen Klammern,
- das et-Zeichen (&),
- das Apostroph sowie die
- Anfuhrungszeichen.

Neben der genannten Zeichenreferenz gibt es auch noch die **Entity-Referenzen**. Diese sind eine Art *Abkrzungen* fr beliebige andere Texte³⁹. Entity-Referenzen besitzen einen Namen, der zwischen `&` und `;` **eingeschlossen** wird.

Die folgenden Zeilen zeigen die Entities, die immer verfgbar sind.

Entity	Zeichen	Beschreibung
&	&	et-Zeichen
<	<	less than
>	>	greater than
'	'	apostrophe
"	"	quotation mark

Definitionen für **eigene Entity-Referenzen** kann man in der Dokumenttyp-Deklaration unterbringen und genauso verwenden wie die vordefinierten.

```
<?xml version="1.0"?>
<!DOCTYPE buch system "buch.dtd" [
  <!ENTITY titel "STUNNING - STUdie über die
    Neigung von ausgebildeten
    Informatikern, einfache sachverhalte
    in umständliche formulieruNGen mit
    merkwürdiger klein-/großschreibweise
    zu fassen, um möglichst sinnlose
    akronyme zu erhalten."
]>
```

Definition einer
Entity Referenz

```
<buch>
<buchtitel>&titel;</buchtitel>
```

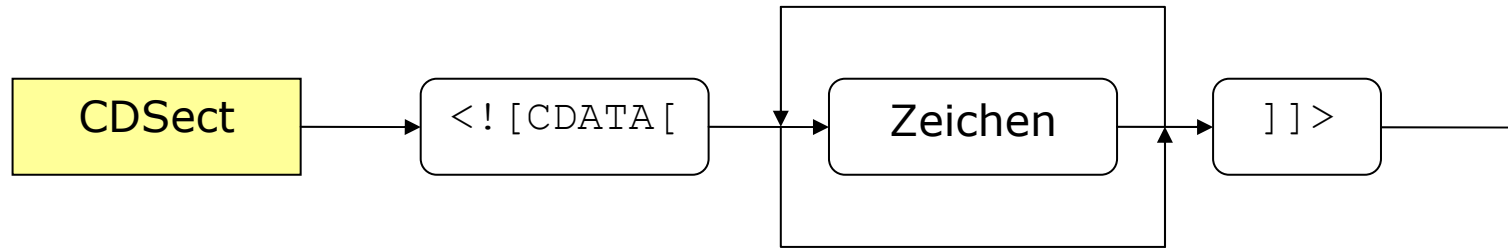
Verwendung einer
Entity Referenz

```
<absatz>Der obige Buchtitel, &titel;,
ist hervorragend dazu geeignet, den Sinn von Entities zu zeigen.
Denn zweimal möchte man den Satz &quot;&titel;&quot; sicherlich
nicht schreiben.
</absatz>
```

Falls Text **nicht interpretiert** übernommen werden sollen, also ohne Ersetzung von Entity-Referenzen o.ä., so steht dafür der **CDATA-Abschnitt** (character data) zur Verfügung.

Die Syntax zeigt folgende Abbildung:

19



Als Zeichendaten sind hier alle Zeichen, also auch spitze Klammern usw. erlaubt, abgesehen natürlich von der abschließenden Kombination „]]>“.

Das folgende Beispiel zeigt zwei verschiedene Schreibweisen für denselben Text: Zunächst unter Verwendung von Entity-Referenzen, anschließend als CDATA.

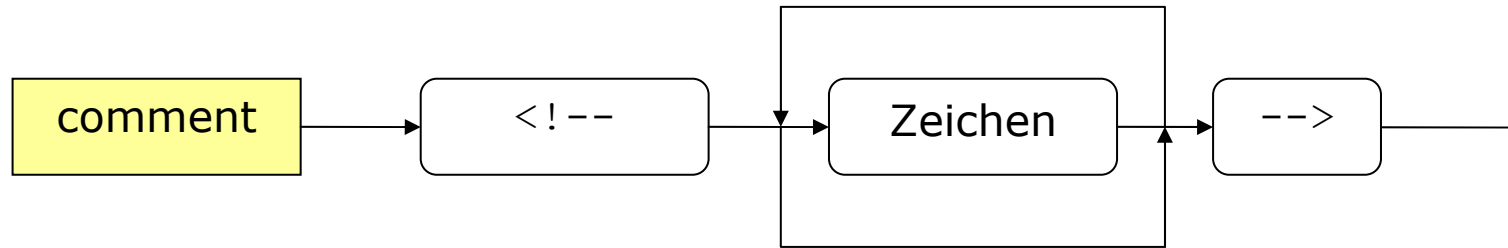
```
<!&absatz&gt;Beispiele in einem XML-Buch k&ouml;nnte man gut in einen
CDATA-Abschnitt einbetten&lt;/absatz&gt;
```

```
<![CDATA[<absatz>Beispiele in einem XML-Buch k&ouml;nnte man gut in einen
CDATA-Abschnitt einbetten</absatz>]]>
```

3.4. Kommentare

Kommentare in XML werden HTML-like verwendet. Das folgende Bild zeigt den Aufbau:

16



Kommentare dürfen überall dort im Dokument stehen, wo Text zulässig ist, also insbesondere **nicht** innerhalb der Tags anderer Elemente.

Der Text innerhalb eines Kommentars gehört nicht zum Dokument. Es ist einem XML-Prozessor freigestellt, ob er Kommentartexte für ein Anwendungsprogramm zugänglich macht oder nicht.

```

<!-- Der folgenden Abschnitt MUSS überarbeitet werden!
      Einfließen sollte ...
-->
  
```

...

```

diese Anweisung dient der
<em>interoperability<!-- dieses Wort
muss noch übersetzt werden! --></em>
  
```

Kommentare können sowohl zwischen den Start- und End-Tags von Elementen auftreten, als auch sich über mehrere Zeilen erstrecken.

3.5. Processing Instructions

Processing Instructions erlauben es Anweisungen für die Verarbeitung¹ in einem XML-Dokument unterzubringen.

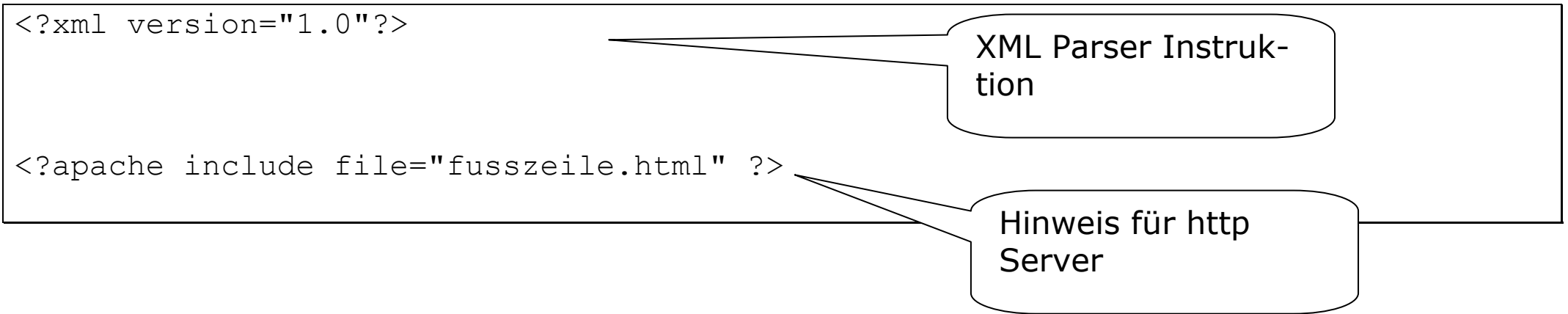
Syntax:



Processing Instructions stehen zwischen `<?` und `?>`.

Beispiel:

¹ Eigentlich widerspricht dies der XML Philosophie (Trennung Inhalt und Verarbeitung).



Es empfiehlt sich, als PITarget eine Zeichenfolge zu wählen, die jede Verwechslung ausschließt.

Aus Gründen der Eindeutigkeit ist auch die Bezeichnung `xml` verboten.

Die Anweisungen, die hinter dem PITarget stehen, sind keiner Einschränkung unterworfen, solange nicht die Kombination `?>` darin auftaucht.

3.6. Einbeziehen von Binärdaten

In ein XML-Dokumente sind auch andere Medien einbindbar; der XML Parser selbst ignoriert solche Daten.

Dazu wird ein externes Entity deklariert und in den Text eingefügt.

Beispiel: ein JPEG-Bild wie folgt eingebunden:

1. Deklaration in der DTD:

```
<!ENTITY am_strand SYSTEM
          "urlaub/strandfoto.JPEG"
          NDATA JPEG >
```

2. Benutzung in einer XML Instanz:

```
&am_strand;
```

NDATA zeigt dem Anwendungsprogramm, dass es sich hier nicht um XML-Daten handelt (non XML-Data), sondern um Daten in einer speziellen **Notation**, nämlich JPEG. Auch der Name JPEG muss ordentlich deklariert werden (-> DTDs).

3.7. Dokumenttyp-Definition (DTD)

Im vorigen Teil wurde gezeigt, wie XML Dokumente aussehen. Hier werden XML-Konstrukte beschrieben, um die Dokumentstrukturen in DTDs zu spezifizieren.

Eine DTD wird verwendet, etwa um die Struktur von Datenschnittstellen zu beschreiben, damit Anwendungen Daten austauschen können; jeder Teil der Schnittstelle verwendet die selbe DTD – damit ist das Datenformat verbindlich festgelegt.

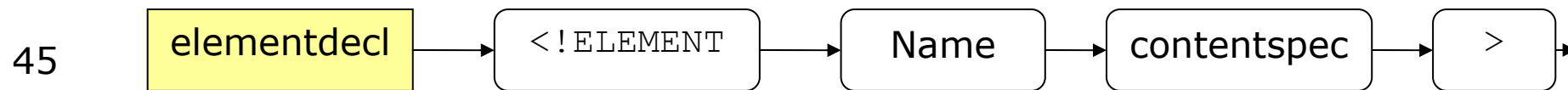
Eine **Dokumenttyp-Deklaration** informiert den XML-Prozessor darüber, *welche* Dokumenttyp-Definition (DTD) für die folgende Instanz gültig ist und entweder *wo* sich die Datei mit der DTD befindet (**externe DTD**) oder *wie* die DTD aussieht (**interne DTD**). Die DTD ist damit ein *Bestandteil der Dokumenttyp-Deklaration*.

Eine **Dokumenttyp-Definition** (DTD) enthält die Informationen darüber,

- welche Elementtypen es gibt,
- welchen Inhalt die Elementtypen haben dürfen,
- welche Attribute erlaubt sind und
- welche Werte die Attribute annehmen dürfen.

3.7.1. Elementtyp-Deklaration

Eine **Elementtyp-Deklaration** besitzt folgende Gestalt:



Der **Name** ist die Bezeichnung eines Elements. Der **Inhalt** (`contentspec`) beschreibt die Struktur.

Beispiele:

```
<!ELEMENT adresse1 (vorname, nachname, strasse, ort)>
<!ELEMENT adresse2 (anrede?, vorname, nachname, strasse, ort)>
```

```
<!ELEMENT adresse3 ((firma | person) , strasse, ort)>
<!ELEMENT firma      (firmenname)>
<!ELEMENT person     (anrede?, vorname, nachname)>
<!ELEMENT adressbuch (adresse)+>
<!ELEMENT adressbuch (adresse)*>
```

Die Namen von Elementen beginnen mit einem Buchstaben, dem Unterstrich oder einem Doppelpunkt. Als weitere Zeichen sind dann auch Ziffern, der Punkt und der Bindestrich gestattet (die genaue Syntax enthalten die Klauseln [4] bis [8] der Spezifikation).

Zwei Dinge sind wichtig:

- In der Praxis sollte man den **Doppelpunkt nicht** für selbst definierte Elemente **verwenden**. Zur Zeit wird er als Trennzeichen für »Namensräume« benutzt. In einer zukünftigen XML-Version könnte der Doppelpunkt eine besondere Bedeutung erhalten.
- Bei den Namen wird zwischen Klein- und Großschreibung unterschieden!

Das Repertoire zur Definition des Inhaltes zeigen im Wesentlichen die obigen Beispiele.

Das **Komma** zwischen den Elementen ist der so genannte **Sequenz-Operator**.

```
<!ELEMENT adresse1 (vorname, nachname, strasse, ort)>
```

Eine `adresse1` besteht einfach aus der Folge `vorname, nachname, strasse, ort`.

Optionale Elemente werden mit dem **?** gekennzeichnet.

```
<!ELEMENT adresse2 (anrede?, vorname, nachname, strasse, ort)>
```

Das Fragezeichen bedeutet, dass die `anrede` fehlen darf.

Alternativen werden mit dem **Oder-Operator**, der als senkrechter Strich (`|`) geschrieben wird, ausgedrückt.

```
<!ELEMENT adresse3 ((firma | person) , strasse, ort)>
```

Dieses Adreßtyp-Beispiel, `adresse3`, kann mit Firmen- und Privatadressen umgehen.

Wiederholungen können mit dem Plus- und dem Stern-Symbol ausgedrückt werden.

```
<!ELEMENT adressbuch (adresse)+>
```

Hier muss ein Adreßbuch *mindestens eine* Adresse enthalten.

```
<!ELEMENT adressbuch (adresse)*>
```

Der Stern in der zweiten Adreßbuch-Version bedeutet, dass *beliebig viele* Adressen aufeinander folgen dürfen. Im Unterschied zum Pluszeichen darf das Buch hier auch leer sein.

Für komplexere Elementinhalte können die Operatoren **kombiniert** werden. So kombiniert etwa das Beispiel `adresse3` den Sequenz- und den Oder-Operator. Um Verwechslungen zu vermeiden, müssen Klammern gesetzt werden.

Inhalte, die nicht nur aus Elementen bestehen

Sollen Inhalte neben Elementen auch **Texte** enthalten, so muss der **Typ des Inhalts** in der Deklaration bekannt gemacht werden. Die folgende Deklaration lässt dies zu, indem sie als Inhalt von `strasse` `PCDATA` (Parsed Character Data) einsetzt

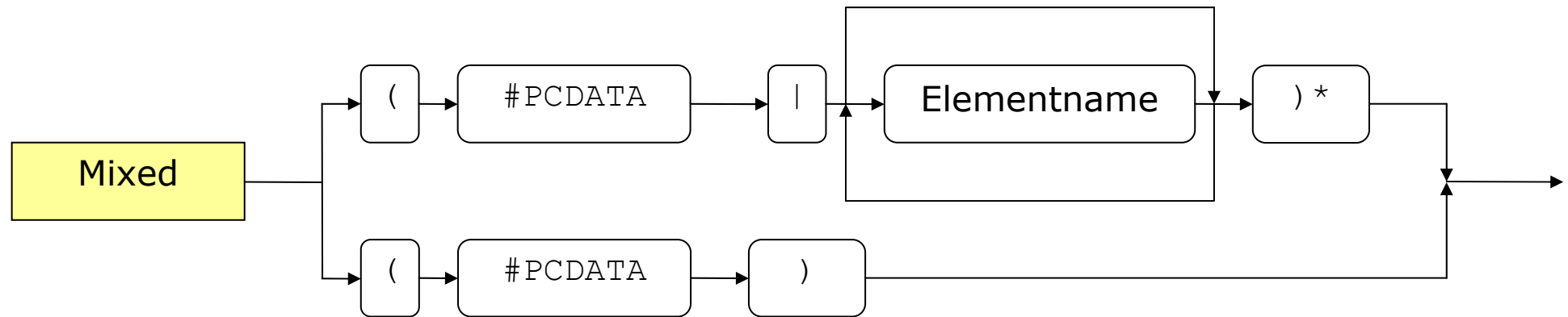
```
<!ELEMENT strasse (#PCDATA)>
```

Der Unterschied zwischen `PCDATA` und `CDATA` besteht darin, dass `PCDATA` vom Parser analysiert wird, während `CDATA` nicht mehr betrachtet wird. Die Konsequenz davon ist, dass innerhalb von `CDATA` keine Markup-Zeichen (z.B. spitze Klammern) erkannt werden. In `PCDATA` werden aber auch Entity References erkannt und aufgelöst.

Weiterhin ist so genannter **gemischter Inhalt** (mixed content) möglich. Darunter versteht man Inhalt, der *sowohl* Text (`PCDATA`) *als auch* andere Elemente enthält. In der Deklaration muss dann `PCDATA` vor den weiteren

Elementen stehen. Folgendes Syntaxdiagramm zeigt den Aufbau:

51



Eine Anwendung ist zum Beispiel der Inhalt eines »Absatzes«. Er enthält Text, aber auch weitere Elemente. Wir zeigen dies mit dem aus HTML bekannten Element `em` (emphasize) zur Hervorhebung. Zuerst die Deklaration, dann die Anwendung:

```

<!ELEMENT absatz (#PCDATA | em)* >
<!ELEMENT em      (#PCDATA)      >
  
```

Deklaration in DTD

```

<absatz>Ein Beispiel für eine <em>Hervorhebung</em>
in einem Absatz</absatz>
  
```

Anwendung im XML Do-
kument

Zwei reservierte Namen für den Inhalt eines Elements sind **ANY** und **EMPTY**.

ANY erlaubt **gemischte** Inhalte mit *jedem beliebigen* Element, das in der DTD deklariert ist. EMPTY deklariert ein **leeres** Element, wie etwa das `img` aus HTML.

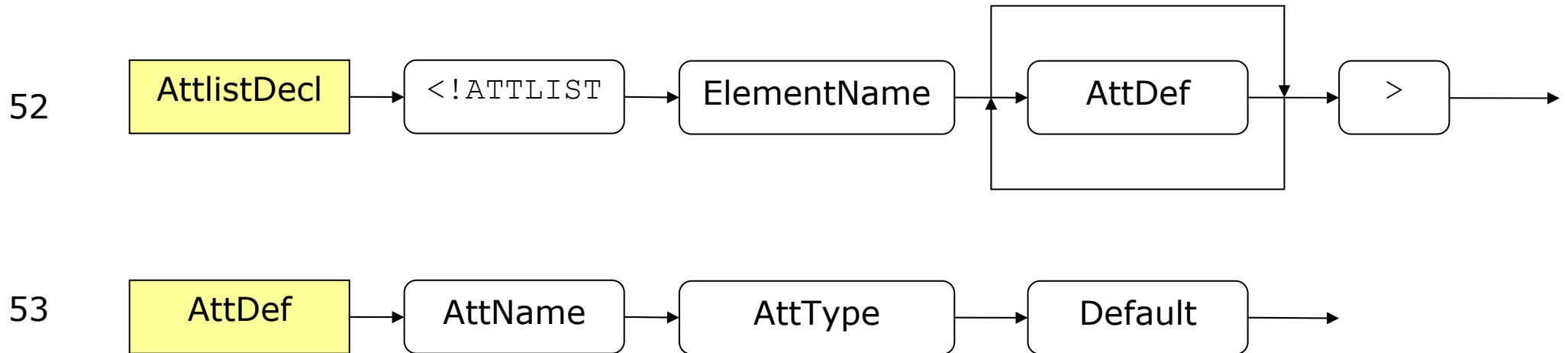
```
<!ELEMENT alles-in-einem ANY >  
<!ELEMENT hr EMPTY >
```

3.7.2. Attributlisten-Deklaration

Eine Attributlisten-Deklaration legt fest,

- welche Attribute für einen Elementtyp existieren,
- von welchem Typ die Attributwerte sind,
- und ggf. enthält sie einen Vorgabe-Wert (default).

Eine Attributlisten-Deklaration ist durch folgendes Syntaxdiagramm beschreiben:



Die folgenden Beispiele zeigen Attribute, die in HTML vorkommen.

```
<!ATTLIST img
  src      CDATA      #REQUIRED
  alt      CDATA      #REQUIRED
  height   CDATA      #IMPLIED
  width    CDATA      #IMPLIED
>

<!ATTLIST h1
  id       ID          #IMPLIED
  align    (left|center|right|justify) left
>
```

Die beiden *notwendigen* (REQUIRED) Attribute `src` und `alt` erwarten beide eine *Zeichenkette* als Wert (CDATA). Gleiche Wertetypen sind für die Höhen- und Breitenangabe erlaubt, jedoch müssen diese Werte nicht explizit angegeben werden. Das Anwendungsprogramm (bei HTML in der Regel der Web-Browser) muss ggf. die impliziten Werte (IMPLIED) verwenden. Bei Grafiken heißt das, der Browser muss die Ausmaße selbst ermitteln.

Für den HTML-Elementtyp `h1` sind im Beispiel zwei weitere Attributtypen zu sehen: `id` dient dazu, eine eindeutige *Identifikation* an einem Element anzubringen. Das zweite Attribut von `h1`, `align`, ist vom Typ **Aufzählungstyp**. Der Wertebereich dieses Typs ist *vollständig* in den Klammern enthalten. Andere Werte als die dort gezeigten sind nicht zugelassen. Die Vorgabe ist in diesem Fall der Wert `left`.

3.7.3. Möglichkeiten, die DTD zu gestalten und zu gliedern

Parameter Entity Reference

Entity-Referenzen zur Abkürzung und Mehrfachverwendung von wiederkehrenden Phrasen haben wir bereits kennen gelernt. Der Einsatz ist auf die Instanz beschränkt, in der DTD jedoch **nicht** zulässig.

An ihre Stelle tritt die so genannte **Parameter Entity Reference (PEReference)**. Die Schreibweise unterscheidet sich von den Entity References dadurch, dass statt des et-Zeichens (&) nun ein Prozentzeichen (%) als Präfix dient.

Beispiel:

```
<!ENTITY % block "absatz | listing | abbildung | kasten">

<!ELEMENT buch (kapitel+)>
<!ELEMENT kapitel (ueberschrift, (%block;)*, abschnitt+)>
<!ELEMENT abschnitt (ueberschrift, (%block;)*, unterabschnitt*)>
<!ELEMENT unterabschnitt (ueberschrift, (%block;)+)>
```

In obigem Beispiel gibt es mehrere Textblöcke, die im Inhaltsmodell dreier Elementtypen auftreten. Auf den ersten Blick ist damit nur eine kürzere Schreibweise erzielt worden. Sollten nun Änderungen anstehen, wie etwa das Hinzufügen eines Blockelements »Tabelle«, so muss lediglich die Definition der PEReference geändert werden. Neben der leichteren Wartbarkeit einer DTD erhöht sich durch den sinnvollen Einsatz von PEReferences auch die Lesbarkeit.

Externe Entities

Sollen Elementtypen in *mehreren* DTDs enthalten sein und nicht in jeder DTD-Datei einzeln hineingeschrieben werden, kann man externe Entities verwenden:

```
<!ENTITY % tab-modell SYSTEM "/usr/local/sgml/tabelle.dtd">
%tab-modell;
<!ENTITY % block "absatz | listing | abbildung | kasten | tabelle">
```

Hier sind die Blöcke um eine Tabelle ergänzt worden. Die Tabelle selbst ist in einer externen Datei namens `tabelle.dtd` gespeichert, die sich im Verzeichnis `/usr/local/sgml/` befindet.

Die erste Zeile definiert das Entity `tab-modell`. In der zweiten Zeile wird dieses Entity expandiert, d.h. der Inhalt der Datei wird an dieser Stelle »eingefügt«. Die Wirkung ist also die glei-

che, als ob die Deklaration für den Elementtyp »Tabelle« (und seine ggf. benötigten weiteren Elementtypen) in der DTD selbst stünde.

Bedingte Abschnitte

Bedingte Abschnitte einer DTD, sind Bereiche, die nicht immer »sichtbar« sind, sondern nur unter gewissen Bedingungen.

Realisiert wird ein bedingter Abschnitt durch einen markierten Abschnitt der *externen* (!) DTD, der entweder **berücksichtigt** (`include`) oder **ignoriert** wird (`ignore`).

Als Beispiel gehen wir davon aus, dass ein Kommentar in ein Dokument geschrieben werden soll, der nur zur Entwurfsphase sichtbar sein soll.

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

In der **Entwurfsphase** soll der **Kommentar sichtbar** sein:

```
<![INCLUDE [  
  <!ELEMENT kommentar (#PCDATA)>  
]]>
```

Am **Ende des Entwurfs** wird die **DTD geändert** in:

```
<![IGNORE [  
  <!ELEMENT kommentar (#PCDATA)>  
]]>
```

Nun ignoriert der XML-Parser diese Abschnitte (=Kommentare).

3.7.4. Notation-Deklaration

Das Einbinden von Bildern ist über die Notationen JPEG erfolgt:

```
<!ENTITY am_strand SYSTEM
        "urlaub/strandfoto.JPEG"
        NDATA JPEG >
```

Ein solches Entity kann man je nach Dokument in der internen Teilmenge der DTD einfügen und im Text in der Form `&am_strand;` verwenden.

Die **Notation-Deklaration** für Formate, wie in diesem Fall JPEG, sollte man besser in der externen Teilmenge unterbringen. Dann steht sie in jeder Instanz zur Verfügung. Die Syntax sieht so aus:

```
<!NOTATION JPEG PUBLIC "ISO/IEC 10918:1993//NOTATION
        Digital Compression and Coding of
        Continuous-tone Still Images
        (JPEG)//EN"
>
```

Der Name des Public Identifier ist in diesem Fall der internationale Standard, der das JPEG-Format definiert. An dieser Stelle kann auch ein System Identifier stehen, der beispielsweise den Dateinamen eines Programms zum Anzeigen von Grafiken enthält. Die Verwendung dieser Information ist durch XML nicht geregelt und bleibt voll und ganz dem Anwendungsprogramm überlassen.

Folgende Beispiele zeigen *mögliche* Notation-Deklarationen für non-XML-Data:

```
<!NOTATION MPEG SYSTEM
    "/usr/local/bin/mpeg_player" >

<!NOTATION WAV SYSTEM
    "C:\windows\player.exe" >

<!NOTATION fd          PUBLIC "ISO/IEC 10744:1992//NOTATION
                          File Descriptor Storage Object Specification//EN">
<!NOTATION unix       PUBLIC "ISO/IEC 10744:1992//NOTATION
                          Unix Storage Object Specification//EN" >
<!NOTATION dos        PUBLIC "ISO/IEC 10744:1992//NOTATION
                          DOS Storage Object Specification//EN" >

<!NOTATION SGML       PUBLIC "+//ISO 8879:1986//NOTATION
                          Standard Generalized Markup Language//EN">
```

4. XML: Linking

Von HTML ist die Möglichkeit, Textstellen und Dateien miteinander über einen Link (Verbindung) zu verknüpfen, bekannt. In XML ist dies auch vorgesehen. Dazu gibt es Vorschläge, **XLink** und **XPointer**, die zwischen einfachen Links und erweiterten sowie solchen auf einzelne Bestandteile einer XML-Instanz unterscheiden.

Diese Sprachen werden hier nicht ausführlich behandelt; es wird nur ein kurzes Beispiel angegeben.

4.1.XLink

XLink nimmt die Möglichkeiten von HTML auf:

- Links zwischen Dateien und Dateiteilen,
- Adreßangaben von Bildern und ImageMaps.

Im Vergleich zu HTML sind Links in XML jedoch immer etwas genauer zu bestimmen, weil es grundsätzlich zunächst zwei Arten von Links gibt:

- einfache und
- erweiterte.

Der einfache Link entspricht dem aus HTML bekannten und sieht nur in einem Detail anders aus:

HTML:

```
<A HREF="http://www.nimmermehr.de/wichtig.html#12">wow</A>
```

XML:

```
<a xml:link="simple" href="http://www.nimmermehr.de/wichtig.html#12">wow</a>
```



Durch „extended“ werden erweiterte Linkelemente gekennzeichnet.

Sowohl das einfache als auch das erweiterte Linkelement können über die Adressangabe (oben das `href`) hinaus weitere Attribute enthalten, die die Bedeutung der Adressangabe sowie der referenzierten Ressourcen und ihre Rolle betreffen. Es handelt sich dabei u.a. um:

- `href`
- `role`
- `title`
- `show`
- `actuate`
- `behavior`

`href` ist das Attribut, das die **Adressangabe** als Wert enthält. Ein Inline-Link ist wie im `A`-Element in HTML einer, der Bestandteil des Linkelements ist. Im Gegensatz dazu ist das beim

Out-of-Line-Link nicht der Fall. Vielmehr enthält hier ein im Linkelement eingeschlossenes Element die eigentliche Adressangabe.

Die Werte von `role` und `title` sind eine Beschreibung der **Funktion** des Links und, wie der zweite Name schon sagt, ein Titel; beide sind als CDATA deklariert. Allerdings gilt das hinsichtlich `role` nur für die erweiterten Links, denn für die einfachen existiert schon ein vorgegebenes Attribut dieses Namens, bezogen auf den gesamten Link.

Zwei weitere Attribute können jeweils vorgegebene Werte besitzen: `show` kann `embed`, `replace` oder `new` sein. Der Wert entscheidet darüber, ob die Ressource an dieser Stelle **eingebildet** wird (`embed`), den gegenwärtigen Inhalt **ersetzt** (`replace`) oder etwa in einem **neuen Fenster** erscheint (`new`).

`actuate`, das als Wert entweder `auto` oder `user` enthält, legt fest, **wann** zu einer Ressource **verzweigt** (traversiert) wird: denkbar wäre ein automatisches Einblenden einer fremden Textpassage in ein eigenes Dokument (`show="embed"`).

Schließlich kann ein Link noch das Attribut `behavior` enthalten, das inhaltlich nicht festgelegt ist (CDATA) und Hinweise auf die Art des Traversierens beinhalten kann. Hier kommt es offensichtlich darauf an, wie eine Anwendung mit dem Attribut umgehen soll.

Beispiel:

Einfacher Link (vergleichbar mit HTML Links)

DTD

```
<!ELEMENT ganzeinfach ANY>

<ATTLIST ganzeinfach
    xml:link      CDATA      #FIXED      "simple"
    href          CDATA      #REQUIRED  >
```

Dokument

```
<ganzeinfach href="http://www.nimmermehr.de/foo/bar.xml">
dieses spezielle Tutorial</ganzeinfach>
```

Einfacher Link mit mehreren Attributen

DTD

```
<!ELEMENT einfach ANY>

<ATTLIST einfach
  xml:link      CDATA      #FIXED      "simple"
  href          CDATA      #REQUIRED
  role          CDATA      #IMPLIED
  title         CDATA      #IMPLIED
  show          (embed|replace|new) #IMPLIED>
```

Dokument

```
<einfach href="http://www.nimmermehr.de/foo/bar.xml"
  content-role="Anleitung"
  content-title="Mehr X M L ist besser"
  show="new">dieses spezielle Tutorial</einfach>
```

Erweiterter Link

Zu den Besonderheiten der erweiterten Links gehört

- dass es möglich ist, Verweise von Read-only-Medien oder Dateien aus zu anderen Stellen einzurichten und
- dass man Links zu und von Daten aus erzeugen kann, die selbst kein Linking unterstützen.

Für eine solche Liste von Links sind Kindelemente vorgesehen, die die einzelnen Adressangaben liefern.

DTD:

```
<!ELEMENT erweitert ANY>

<ATTLIST erweitert
    xml:link      CDATA      #FIXED      "extended">

<!ELEMENT erwverweis ANY>

<ATTLIST erwverweis
    xml:link      CDATA      #FIXED      "locator"
    href          CDATA      #REQUIRED
    role          CDATA      #IMPLIED>
```

Dokument:

```
<erweitert xml:link="extended" inline="false">
  <erwverweis href="http://www.nimmermehr.de/foo/bar1.xml"
    role="Einstieg" />
  <erwverweis href="http://www.nimmermehr.de/foo/bar2.xml"
    role="Uebung" />
  <erwverweis href="http://www.nimmermehr.de/foo/bar3.xml"
    role="Vertiefung" />
  <erwverweis href="http://www.nimmermehr.de/foo/bar4.xml"
    role="Schluss" />
</erweitert>
```

Denkbar wäre, ein PopUp-Menü zu erzeugen, das über die alternativ zu findenden Ressourcen aufklärt und eine Auswahl bietet, wenn jemand die Maus über ein sensibles Gebiet auf der Web-Seite führt. In solchem Umfeld kämen Attribute wie `role` und `title` zum Tragen, denn ihre Werte könnten zur Beschreibung der Adressangaben, etwa im genannten Auswahlmenü, herangezogen werden.

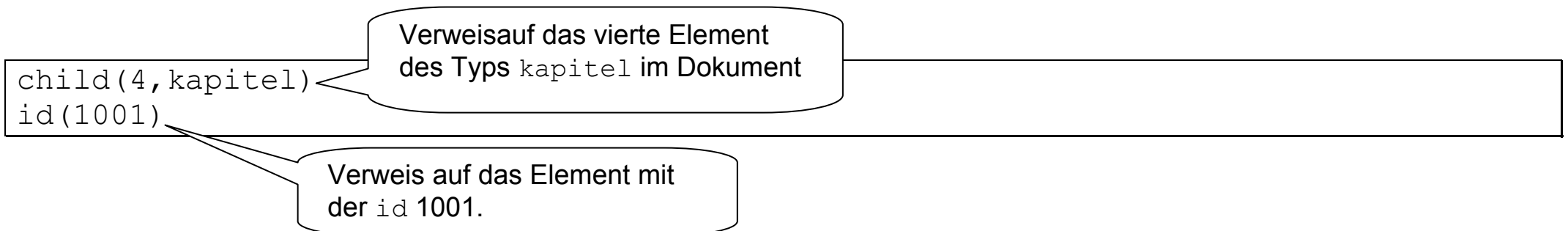
4.2.XPointer

Während mit **XLink auf** andere **Dokumente** verwiesen werden kann, ist es möglich mit **XPointer** auf Bestandteile anderer Dokumente zu verweisen – also Verweise **in** andere **Dokumente**.

XPointer beschreiben einen **Ort oder Bereich** innerhalb einer **XML-Instanz**. Um dies zu tun, bedienen sie sich der Struktur der einzelnen Dokumente.

Dazu bedienen sich die XPointer der Möglichkeit, einem URI (Universal Resource Identifier) nach dem Fragmentbezeichner ('#') einen XPointer anzugeben.

Beispiele für XPointer:



Es gibt verschiedene Arten von solchen **Verweisausdrücken**:

- absolute,
- relative,
- solche, die nach einem Bereich suchen,
- andere, die nach den Attributwerten gehen und
- solche, die nach Zeichendaten suchen.

Sie werden jeweils kurz beschrieben.

4.2.1. Absolute Verweisausdrücke

Absolute Verweisausdrücke werden durch

- `root()`
Suche startet beim Wurzelement des Dokumentes
- `origin()`
die Stelle, von der aus das Traversieren begonnen hat
- `id`
`id(Name)` verweist auf das Element, das ein Attribut vom Typ `ID` mit einem Wert von `Name` hat

- `html`
`html (AttrWert)` sucht nach dem ersten Element, das den Ausdruck `name="AttrWert"` enthält.

angegeben.

4.2.2. Relative Verweisausdrücke

Mit relativen Verweisausdrücken kann man **innerhalb** des **Dokumentbaums navigieren**.

Dazu stehen folgende Schlüsselworte zur Verfügung:

Schlüsselwort	bezieht sich auf
child	Direkte Kinder der Verweisquelle
descendant	Nachfahren innerhalb der Verweisquelle
ancestor	Vorfahren, die die Verweisquelle enthalten
preceding	Elemente vor der Verweisquelle
following	Elemente, die der Verweisquelle folgen
psibling	Ältere Geschwister-Elemente
fsibling	Jüngere Geschwister-Elemente

The diagram illustrates a tree structure with a central node 'X' marked with an 'X'. Above 'X' is a node labeled 'ancestor'. Below 'X' are two nodes labeled 'psibling' and 'fsibling'. Below these are two nodes labeled 'child'. Below the children are four nodes labeled 'descendant'.

Beispiele:

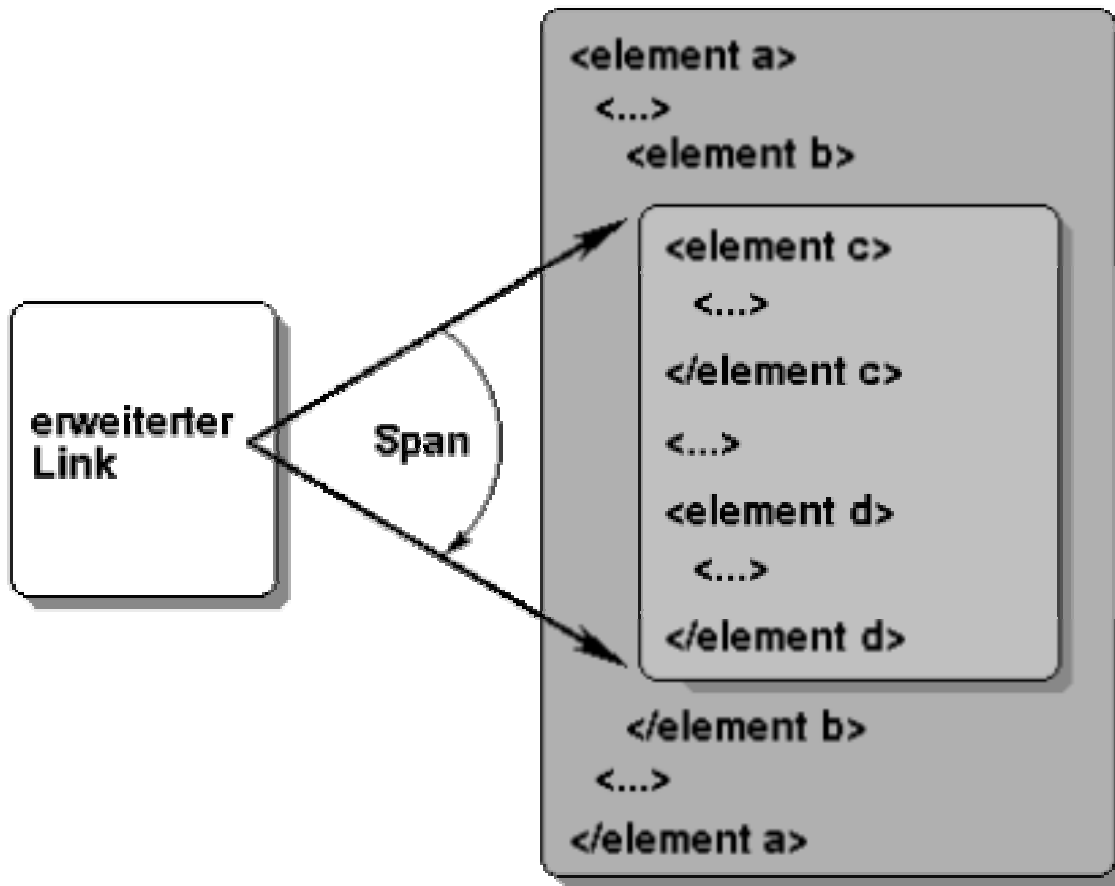
```
child(4, kapitel) .child(2, abschnitt)  
id(1001) .child(3, div) .child(2, p)  
root() .descendant(-2, DIV)
```

Suche im vierten Kapitel nach dem zweiten Abschnitt.

Im Element, das ein Attribut vom Typ `id` enthält, wird nach dem dritten mit `div` bezeichneten Element gesucht - darin wiederum nach dem zweiten Absatz (`p`).

4.2.3. Verweisausdrücke für Bereiche

Ein Bereichsverweisausdruck (»spanning location term«) verweist durch das Schlüsselwort `span` und zwei XPointer auf einen Bereich, der mehrere Elemente umfasst.



Beispiel:

```
root().descendant(-2,DIV).span(child(1),child(2))
```

Umfasst ausgehend vom Wurzel-Element, innerhalb der vorletzten `DIV` das erste und zweite Kindelement.

4.2.4. **Attributverweisausdrücke**

Attributverweisausdrücke haben das Schlüsselwort `attr`, das den Wert des Attributs enthält.

Beispiel:

```
attr("1ba")
```

4.2.5. **Stringverweisausdrücke**

Mit dem **Stringverweisausdruck** ist es möglich, einen Verweis auf beliebige Stellen in Zeichendaten zu platzieren. Ein solcher Ausdruck kann folgende Parameter enthalten:

```
string(wievieltetesVorkommen, String, Position, Laenge)
```

Der erste Parameter enthält eine Zahl, die das n-te Vorkommen eines Strings benennt. Bei negativem Wert wird vom Ende des Dokuments rückwärts gerechnet. Falls der Parameter den Wert `all` enthält, kommen alle Strings, auf die der zweite Parameter passt, für die Verarbeitung in Frage.

Die Angaben für Position und Länge des bezogenen Strings erlauben es, auf kleinste Bereiche in Textdaten zu verweisen.

Beispiel:

```
id("1ba").string(3,"")
```

verweist auf das dritte Zeichen
des Elements mit id="1ba".

```
root().string(all,"!!!",1,200)
```

Verweist auf alle Textstellen, die drei Ausru-

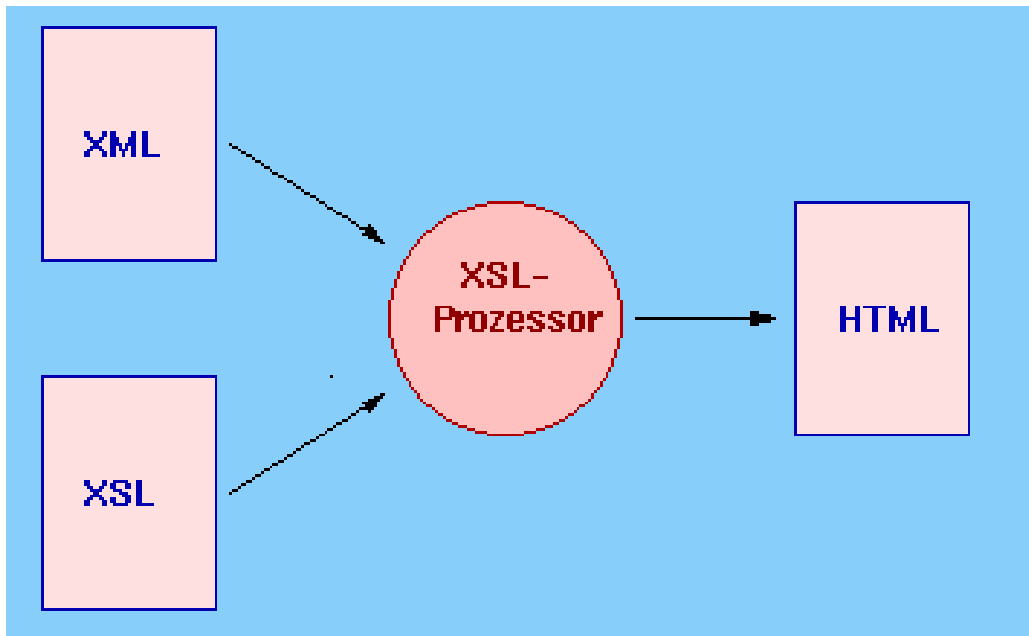
XPointers ist eine Liste der gefundenen Text-
stellen, die eine Anwendung entsprechend

5. XSL

Mit **Cascading Style Sheets** (CSS) für **HTML** ist es möglich, das Basisdokument und die Art und Weise, wie es formatiert werden soll, voneinander getrennt zu halten und zu pflegen. Außerdem erlauben CSS Web-Autoren, »ein« Style Sheet für eine beliebige Menge von HTML-Dokumenten zu nutzen -- und so für ein einheitliches Aussehen der Website (»Corporate Identity«) zu sorgen.

Durch **Extensible Style Language** (XSL) können XML-Dokumente ausgabefertig formatiert werden. Im Regelfall des Webs handelt es sich dabei um eine Umsetzung einer konkreten XML-Sprache in HTML.

Die Vorgehensweise dabei verdeutlicht folgende Abbildung:



XSL wird nicht näher betrachtet, da wir es für die Veranstaltung „Web Services“ nicht benötigen.