Kapitel 16: Einführung in XML und SGML

References:

- Liora Alschuler: ABCD ... SGML — A User's Guide to Structured Information. International Thomson Computer Press (ITP), 1995, ISBN 1-850-32197-3, 414 pages.
- Charles F. Goldfarb, Yuri Rubinsky: The SGML Handbook. Clarendon Press, 1990.
- Henning Lobin: Informationsmodellierung in XML und SGML. Springer-Verlag, 1999.
- C. M. Sperberg-McQueen and Lou Burnard (Eds.): A Gentle Introduction to SGML. [http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html]
- On SGML and HTML (in the HTML 4.01 Specification). [http://www.w3.org/TR/html401/intro/sgmltut.html]
- Yuri Rubinsky, SoftQuad: The SGML Primer. [http://www.softquad.com/top_frame.sq?page=resources/content_sgml_primer.html]
- Martin Bryan (The SGML Centre): An Introduction to the Standard Generalized Markup Language (SGML). [http://www.personal.u-net.com/~sgml/sgml.htm]
- Harvey Bingham: SGML Syntax Summary. [http://www.oasis-open.org/cover/sgmlsyn/contents.htm]
- Charles F. Goldfarb's SGML Source Home Page: [http://www.sgmlsource.com/]
- Boc DuCharme: XML — The Annotated Specification. Prentice Hall, 1999.
- Tim Bray, Jean Paoli, C.M. Sperberg-McQueen: Extensible Markup Language (XML) 1.0, 1998. [http://www.w3.org/TR/REC-xml] See also: [http://www.w3.org/XML].

Lernziele

Nach diesem Kapitel sollten Sie folgendes können:

- die Beziehung zwischen SGML, XML, HTML und XHTML erklären.
- syntaktisch korrekte XML Dokumente schreiben.
- einfache XML DTDs lesen.
- XML mit relationalen Datenbanken vergleichen.

Inhalt

Einführung (1)

- SGML ("Standard Generalized Markup Language") ist ein ISO-Standard seit 1986.
- Früher wurden Manuskripte von Büchern auf der Schreibmaschine geschrieben, "Markup" waren die zusätzlichen, handschriftlichen Anweisungen an den Setzer (z.B. "Fettdruck").

  Anderes Beispiel: Wichtige Teile von Texten werden mit Textmarkern hervorgehoben, gelegentlich sogar verschiedene Arten von Informationen mit verschiedenen Farben. Der Name für die Vorläufersprache GML (1969) wurde wohl nach den Projektmitarbeitern Goldfarb, Mosher und Lorie gewählt. Charles F. Goldfarb gilt als Vater von SGML.

Einführung (2)

- In SGML/XML werden Bestandteile von Texten mit “Tags” markiert, z.B.

$$\underbrace{\underbrace{\texttt{<title>}}_{\text{Start-Tag}}\ \underbrace{\texttt{Mein erstes HTML-Dokument}}_{\text{Inhalt}}\ \underbrace{\texttt{</title>}}_{\text{End-Tag}}}_{\text{Element}}$$

- XML ist eine Teilmenge (vereinfachte Version) von SGML, die 1998 vom W3C als Empfehlung verabschiedet wurde.

  Den Browser-Herstellern war das vollständige SGML zu kompliziert.

Einführung (3)

- SGML hat zwei Ebenen:
  - ◇ Einerseits ist es ein spezieller Syntaxformalismus: Man kann in Form einer DTD (“Document Type Definition”) festlegen, welche Tags erlaubt sind, wie sie geschachtelt werden müssen, u.s.w.
  - ◇ Andererseits ist es ein Datenformat: Man kann Texte oder Daten mit Tags entsprechend einer gegebenen DTD strukturieren.
- Das gilt entsprechend auch für XML.

  Allerdings kann man XML auch ohne vorab festgelegte DTD als Datenformat benutzen (“well-formed XML”). Bei SGML geht das nicht.

Einführung (4)

- HTML (HyperText Markup Language) ist die Syntax, in der Web-Seiten verfasst sind.

  Tatsächlich sind an einer Webseite, wie sie im Browser angezeigt wird, noch weitere Sprachen und Datenformate beteiligt. Z.B. enthält HTML für jedes Bild nur einen Verweis auf die entsprechende Bilddatei (in einem Format wie GIF, JPEG, PNG). Manche Webseiten enthalten auch Programmcode in Javascript, in vielen Fällen wird das genaue Aussehen von Stylesheets in CSS beeinflusst. Aber der eigentliche Text und die wesentliche Struktur der Webseite ist in HTML verfasst.

- HTML ist eine Anwendung von SGML, d.h. SGML für eine spezielle DTD.

  Entsprechend ist XHTML eine Anwendung von XML.

Einführung (5)

- Man kann Syntax (Dokument-Strukturen) in SGML /XML definieren, aber nicht Semantik (Bedeutung).
- Z.B. kann man definieren, daß es ein Tag “`<it>`” gibt, aber daß dies Kursivschrift (“italics”) bedeutet, kann man nicht in SGML/XML festlegen.

  Es gibt zusätzliche Standards: DSSSL, CSS, XSLT/XSL FO.

- Die HTML Spezifikation enthält daher außer einer DTD noch beschreibenden Text, der Bedeutung und Aussehen der Markierungen mit Tags erläutert.

  DTDs sind nicht sehr mächtig, deswegen sind auch einige zusätzliche Syntaxeinschränkungen nur textuell beschrieben.

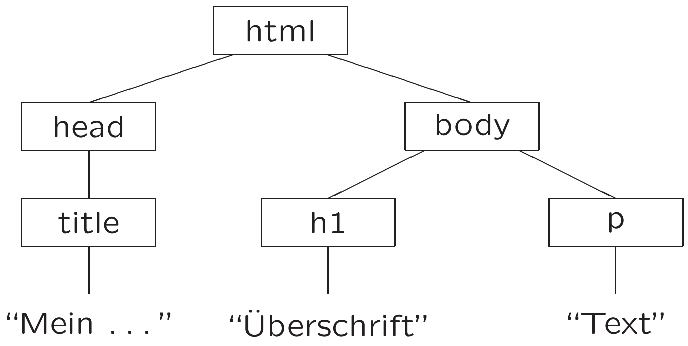
Einführung (6)

HTML-Dokument (Beispiel eines SGML Dokumentes):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <title>Mein erstes HTML Dokument</title>
    </head>
    <body>
        <h1>Überschrift</h1>
        <p>Text</p>
    </body>
</html>
```


Einführung (7)

- Durch Schachtelung ergibt sich eine Baumstruktur:



Einführung (8)

- SGML/XML sind keineswegs darauf eingeschränkt, nur Strukturen in Texten zu markieren.
- Man kann auch Daten mit XML/SGML strukturieren, die man in ähnlicher Weise in einer relationalen Datenbank abspeichern könnte.
- Im folgenden Beispiel kommen auch Attribute vor:

  $$<\underbrace{\texttt{BEW}}_{\text{El.-Typ}}\ \underbrace{\texttt{JAHR}}_{\text{Attribut}} = \underbrace{\texttt{"2006"}}_{\text{Wert}}>\underbrace{\texttt{Na ja, geht so}}_{\text{Inhalt}}</\texttt{BEW}>$$

- Bei leerem Inhalt kann man Start- und Endtag verschmelzen (“leeres Element Tag”, endet in />).

Einführung (9)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Feuerwerksartikel>
    <Bombenrohr Herst="Diamond" Name="Silver Star">
        <Angebot Händler="Röder" Preis="5.49"/>
        <Bew Jahr="2005">Goldpalme! Toll.</Bew>
        <Bew Jahr="2006">Na ja, geht so.</Bew>
    </Bombenrohr>
    <Batterie Herst="Weco" Name="Tanz der Vampire"
            Schuss="12" Hoehe="45" Dauer="30">
        <Angebot Händler="Roeder" Preis="8.50"/>
        <Angebot Händler="Preisw-FW" Preis="7.69"/>
        <Bew Jahr="2006">Rote Blinker. Hübsch.</Bew>
    </Batterie>
</Feuerwerksartikel>
```

Einführung (10)

ARTIKEL						
ID	HERST	NAME	TYP	HOEHE	SCHUSS	DAUER
1	Diamond	Silver Star	BO	60		
2	Weco	Tanz der Vampire	BA	45	12	30

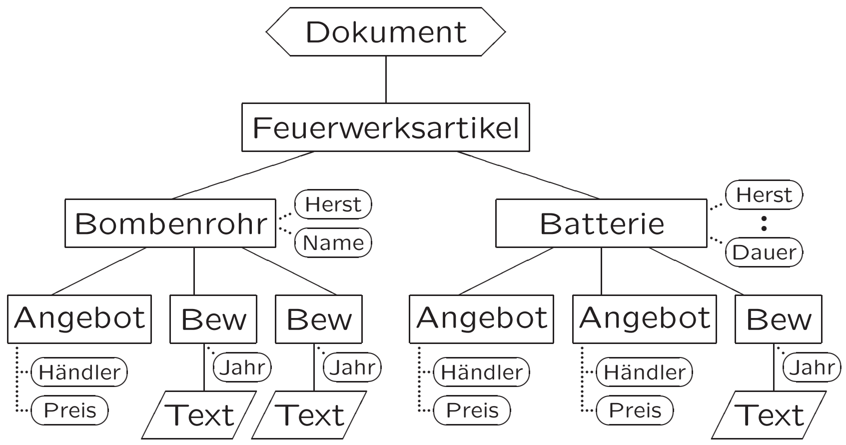
ANGEBOT		
ID	HAENDLER	PREIS
1	Röder	5.49
2	Röder	8.50
2	Preisw.F.	7.69

BEWERTUNG		
ID	JAHR	TEXT
1	2005	Goldpalme! Toll.
1	2006	Na ja, geht so.
2	2006	Rote Blinker. Hübsch.

Einführung (11)

- Das Wesentliche bei XML ist nicht die Syntax, sondern die interne Baustruktur, die sich durch Schachtelung der Elemente ergibt.
- Man kann XML als ein Datenmodell (neben dem relationalen Modell und dem ER-Modell) ansehen.
- Es gibt auch eine eigene Anfragesprache: XQuery.
- Das XQuery Data Model (XDM) abstrahiert von syntaktischen Details und stellt den Inhalt eines XML Dokumentes als Baum mit Element-Knoten, Attribut-Knoten, Text-Knoten u.s.w. dar.

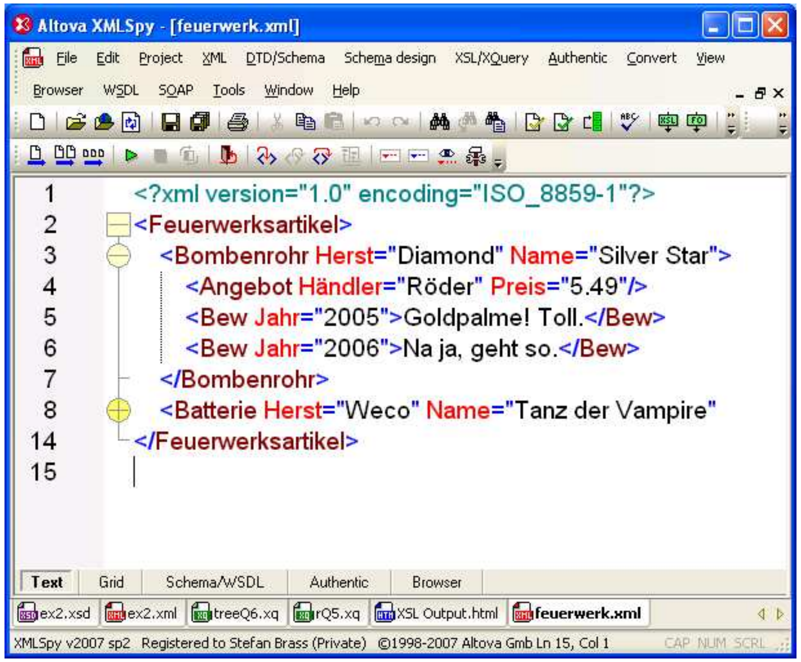
Einführung (12)

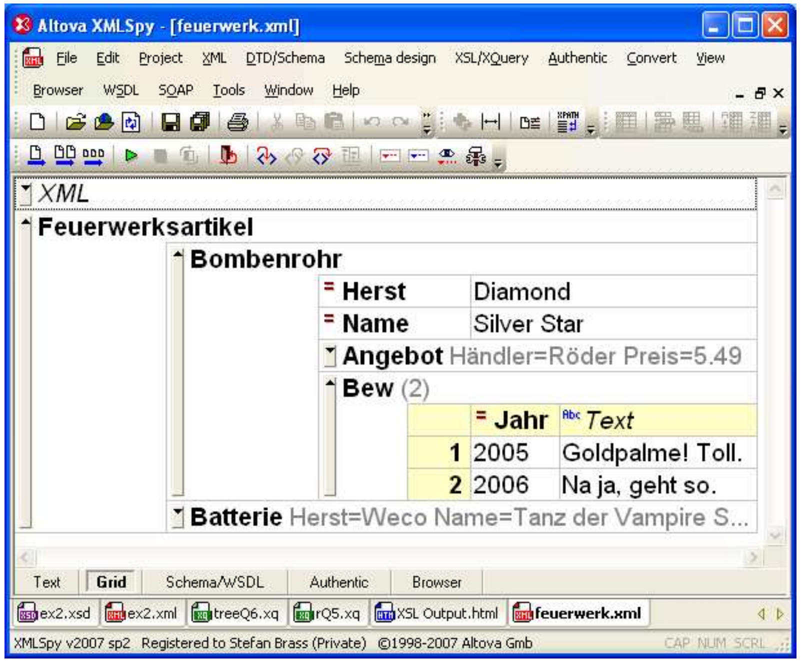


Einführung (13)

- Selbstverständlich kann man SGML/XML Dokumente mit einem beliebigen Texteditor erstellen.
- SGML enthält viele Abkürzungsmöglichkeiten, um den Tippaufwand zu verringern.
- Da man heute bei syntaxgestützten Editoren nicht jedes Zeichen tippen muß, schien das bei XML nicht mehr so wichtig (einfachere, aber längere Syntax).
- Ein Beispiel zeigen die nächsten Folien.

  Die unterschiedliche Visualisierung des gleichen Dokumentes (zum Teil mit Tabellenstrukturen) unterstreicht noch einmal die Bedeutung der internen Datenstruktur im Vergleich zum externen Text.





Markup-Arten (1)

- Wie oben schon erwähnt, war das erste “Markup” Anweisungen für den Setzer, z.B. “18pt Schrift”.
- Dies ist “darstellungsorientiertes Markup”.
- Es funktioniert, solange die einzige Anwendung des Textes das Ausdrucken in genau diesem Format ist.
- Man kann Texte aber auch anders nutzen:
  - ◇ Rechtschreib-Prüfung
  - ◇ Automatische Erstellung Inhaltsverzeichnis
  - ◇ Suchen und Ersetzen
  - ◇ Andere Formate, andere Medien (z.B. Web).

Markup-Arten (2)

- Z.B. kann man Kapitel-Überschriften als groß, fett, zentriert markieren.
- Da es aber vielleicht auch andere große, fette, zentrierte Teile im Text gibt (etwa spezielle Warnungen), kann man aus den vorhandenen Daten nicht automatisch ein Inhaltsverzeichnis konstruieren.

  Außerdem eventuell einzelne Kapitelüberschriften leicht anders (nicht ganz einheitlich): z.B. einmal Zentrierung vergessen, oder etwas kleinerer Zeichensatz, damit die Überschrift in eine Zeile passt.
- WYSIWYG (“What you see is what you get”): “What you see is all you've got” (Brian Kernighan).

Markup-Arten (3)

- Wie auch sonst bei der Erfassung von Daten, muß man sich vorher überlegen, was man mit dem Dokument später machen will.
- Alternative zu darstellungs-orientiertem Markup: Inhaltsorientiertes Markup. Hier wird dieser Textteil als Kapitelüberschrift markiert.

  Die Warnung, die im Beispiel gleich gedruckt wird, wird explizit als Warnung/Hinweis markiert. Bei inhaltsorientierten Markup unterscheidet man also Dinge, die für das reine Ausdrucken nicht unterschieden werden müßten, um so auch andere Anwendungen zu erlauben.

Markup-Arten (4)

- Die genaue Darstellung wird dann (getrennt) in einem Stylesheet festgelegt.

  Es kann z.B. verschiedene Stylesheets für verschiedene Ausgabemedien geben. Wie Kapitelüberschriften aussehen, ist nur an einer Stelle festgelegt, statt bei jeder Überschrift. Es gibt daher eine größere Konsistenz und die festlegung ist leichter änderbar.
- In HTML sind darstellungsorientierte und inhaltsorientierte Elemente vermischt.

  Rein darstellungsorientierte Dinge werden zurückgedrängt, nachdem Cascading Stylesheets verfügbar sind.
- Perfekt inhaltsorientiert geht nicht, da HTML für beliebige Arten von Dokumenten verwendet wird.

Markup-Arten (5)

- Z.B. hat HTML keine Tags für Angebote, Warenbezeichnungen, Preise im Web. Das wäre aber für Preisvergleichsdienste nützlich.
- Diese prinzipielle Einschränkung jeder festen Menge von Tags war eine Motivation für die Einführung von XML.
- Umgekehrt kann man XML anzeigen, indem man mit einem XSLT-Stylesheet eine Übersetzung in HTML festlegt.

  Für HTML ist die Darstellung ja definiert, für beliebiges XML nicht.

Datenaustausch-Format

- XML wird häufig zum Datenaustausch zwischen Firmen verwendet (z.B. Bestellungen, Rechnungen).
- Dazu wäre es natürlich gut, sich auf einen einheitlichen Satz von Tags (eine DTD) für die jeweilige Anwendung zu einigen.
- Eine besondere Stärke von SGML/XML ist aber auch die leichte Konvertierbarkeit.

  Mit einem XSLT Strylesheet kann man ein Dokument bezüglich einer DTD recht einfach in ein Dokument bezüglich einer anderen DTD umwandeln (oder auch in ein Nicht-XML Format).

Inhalt

Elemente (1)

- Ein XML/SGML Dokument ist ein Text, in dem Worte, Phrasen, Abschnitte mit "Tags" markiert sind, z.B.

```
<title>Mein erstes HTML Dokument</title>
```

- Tags sind in "<" und ">" eingeschlossen.

  "Tag" (engl.) heißt u.a. "Etikett", "Schildchen".

- "<title>" ist ein Beispiel für ein Start-Tag.
- "</title>" ist ein Beispiel für ein End-Tag.

  Das End-Tag erkennt man an dem "/" vor dem Namen im Tag.

Elemente (2)

- Der Textteil vom Beginn eines Start-Tags bis zum Ende des zugehörigen End-Tags heißt ein Element.
- Der Name im Start-Tag und im End-Tag heißt Element-Typ. Im Beispiel: "`title`".

  Manche Autoren sagen auch "Element-Name" statt "Element-Typ".

- Oft wird "Tag" gesagt, wo eigentlich "Element" oder "Element-Typ" formal richtiger wäre.

  Z.B. "Jedes HTML Dokument muß ein `title`-Tag enthalten". Es stimmt zwar, daß jedes HTML-Dokument ein Tag "<title>" enthalten muß (und auch ein Tag "</title>"), aber gemeint ist doch: "Jedes HTML-Dokument muß ein `title`-Element enthalten".

Elemente (3)

- Namen (Bezeichner, u.a. für Elementtypen) können Buchstaben, Ziffern, Punkte ".", Bindestriche "-", Unterstriche "_" und Doppelpunkte ":" enthalten.

  Und noch einige zusätzliche Zeichen aus dem Unicode Zeichensatz. Sie müssen mit einem Buchstaben, einem Unterstrich "_", oder einem Dopplepunkt ":" beginnen. Der Doppelpunkt sollte nur in Übereinstimmung mit der Namespace Spezifikation verwendet werden. Alle namen, die mit "`xml`" beginnen, sind reserviert.

- In XML wird Groß- und Kleinschreibung unterschieden, in SGML ist dies wählbar.

  In HTML ist die Groß-/Kleinschreibung egal, außer für Entities (s.u.). In XHTML nicht.

Elemente (4)

- Der Text zwischen Start-Tag und End-Tag (ohne die Tags selbst) ist der Inhalt des Elementes. Im Beispiel von Folie 16-26 ist dies der Text

```
Mein erstes HTML Dokument
```

- Die DTD legt fest, was genau als Inhalt erlaubt ist. Z.B. muß in XHTML eine "unordered list" `ul` (Liste nicht numerierter Punkte) eine Folge von Elementen des Typs "list item" `li` enthalten:

```
<ul><li>First</li><li>Second</li></ul>
```

- Aufgabe: Zeichnen Sie die Baumstruktur dafür.

Elemente (5)

- Elemente können sich nicht partiell überlappen.

  Für je zwei Elemente A und B ist A entweder vollständig in B enthalten, oder B in A, oder die beiden überlappen sich überhaupt nicht.

- Start-Tags und End-Tags müssen daher korrekt geschachtelt werden. Z.B. ist Folgendes erlaubt:

```
<h1><code>...</code></h1>
```

  Das Folgende ist dagegen ein Syntaxfehler:

```
<h1><code>...</h1></code>
```

- Dies entspricht öffenenden und schließenden Klammern verschiedener Typen: ([ ]) ist ok, [( ]) nicht.

Elemente (6)

- Es gibt vier Arten von Element-Typen:
  - ◇ Elemente, die nur reinen Text enthalten können.

    Beispiel: `title`.
  - ◇ Elemente, die nur andere Elemente enthalten.

    Diese "Kind-Elemente" können natürlich ihrerseits wieder Text enthalten. Beispiel: `ul` muß Folge von `li`-Elementen enthalten.
  - ◇ Elemente, die eine Mischung von Text und anderen Elementen enthalten.

    "Mixed content model". Beispiel: `<p>Hello, <em>world</em>!</p>`
  - ◇ Elemente, die immer leeren Inhalt haben.

    Sie markieren dann eine Position im Text (Beispiel: `br` in HTML: Zeilenumbruch) oder enthalten Daten in Form von Attributen.

Leere Elemente

- In XML muß es für jedes öffnende Tag (Start-Tag) ein zugehöriges schließendes Tag (End-Tag) geben.

  SGML erlaubt dagegen, unter gewissen Voraussetzungen Tags wegzulassen, die sich aus dem Kontext eindeutig rekonstruieren lassen.

- Da `<br></br>` nicht gut aussieht (und das schließende Tag keine zusätzliche Information enthält), wurden spezielle Tags für leere Elemente in XML eingeführt: "`<br/>`" ist äquivalent zu "`<br></br>`".

  Dies war einer der wenigen Punkte, bei dem XML zunächst keine Teilmenge von SGML war. Bei SGML bestand das Problem nicht, da man dort das schließende Tag immer weggelassen hätte. SGML wird nun entsprechend erweitert.

Attribute (1)

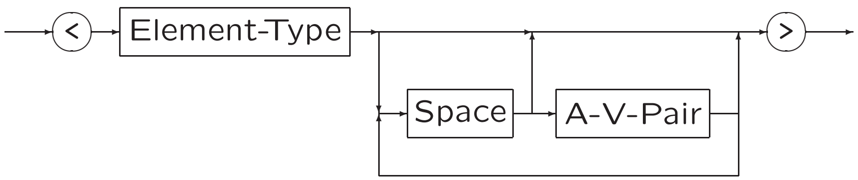
- Im Start-Tag können optional Attribut-Wert Paare angegeben werden.
- Z.B. werden in HTML/XHTML Verweise auf andere Dokumente mit dem Element `a` ("anchor") markiert:

```
XML was developed by the
<a href="http://www.w3.org">W3C</a>.
```

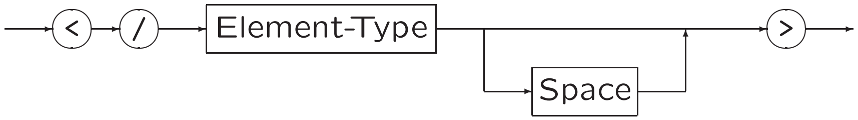
- Der Text des Verweises ("Label") ist im Element-Inhalt gegeben, die URI der referenzierten Webseite im Attribut "`href`".

Attribute (2)

Start-Tag:

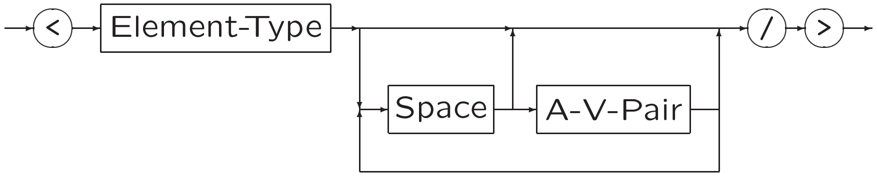


End-Tag:



Attribute (3)

Empty Element Tag:

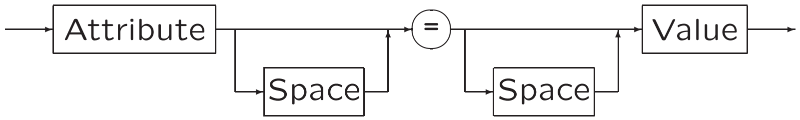


- "Space" (white space) besteht aus ein oder mehr Leerzeichen, Zeilenumbrüchen, Tabulatorzeichen.

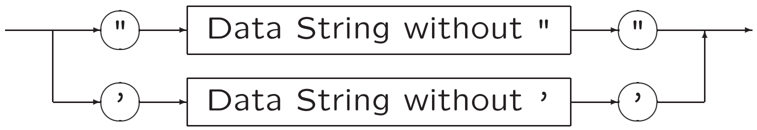
  Dies entspricht den ASCII-Codes 32, 13, 10, 9.

Attribute (4)

A-V-Pair:



Value:



Attribute (5)

- Attributwerte können in " oder ’ eingeschlossen werden. Das jeweils andere Zeilen kann dann im Attributwert benutzt werden.

  Falls man beide Arten von Anführungszeichen benötigt, muß man ein Entity oder eine Zeichen-Referenz verwenden (s.u.).

- Attributwerte können keine Elemente enthalten.
- Das Zeichen “<” ist in Attributwerten verboten.

  Wenn nötig, kann man es aber mit einem Entity oder einer Zeichen-Referenz darstellen (s.u.). Durch das Verbot von “<” in Attributwerten werden Fehler früher gefunden (z.B. ein fehlendes ").

Attribute (6)

- Das Zeichen “`&`” hat in SGML/XML eine besondere Bedeutung (Zeichen/Entity Referenz, s.u.), das gilt auch in Attributwerten.
- Attributwerte können sich über mehrere Zeilen erstrecken. Tabulatorzeichen und Zeilenenden werden dabei intern in Leerzeichen umgewandelt.

  Abhängig vom in der DTD deklarierten Attributtyp wird Leerplatz ggf. normalisiert: Er wird dann am Beginn und Ende des Attributwertes entfernt, mehrere aufeinanderfolgende Leerzeichen werden durch ein einzelnes ersetzt. Für “`CDATA`” Attribute geschieht dies nicht.

- Die Reihenfolge, in der mehrere Attribut-Wert Paare in einem Start-Tag angegeben werden, ist egal.

Zeichen-Referenzen (1)

- Man muß unterscheiden zwischen
  - dem Zeichenvorrat, der intern benutzt wird (Daten, die ein XML Parser der Anwendung liefert)
  - der Codierung dieser Zeichen in Dateien für den Datenaustausch (externe Repräsentation).
- Intern benutzt XML den Unicode Zeichensatz.
- Extern kann man z.B. auch die ISO 8859-1 (ISO Latin 1) Zeichen-Codes verwenden (Teilmenge der Unicode Zeichen für westeuropäische Sprachen).

  Andere Codierungen sind z.B. spezialisiert auf kyrillische Zeichen.

Zeichen-Referenzen (2)

- Die XML-Deklaration ganz am Anfang des Dokumentes (s.u.) legt die Codierung fest.

  Die Codierung kann auch bei der Übertragung des Dokumentes mit HTTP spezifiziert werden. In SGML gibt es eine extra Parameterdatei (SGML Deklaration), die u.a. die Zeichencodierung festlegt.

- Alle Zeichen des ISO Latin 1 Zeichensatzes (der seinerseits ASCII umfasst) sind auch im Unicode Zeichensatz enthalten und haben den gleichen Zahlenwert als Code (siehe aber nächste Folie).

  ASCII ist der “American Standard Code for Information Interchange” (enthält z.B. keine deutschen Umlaute). ASCII ist sehr verbreitet.

Zeichen-Referenzen (3)

- Von der externen Codierung her ist Unicode (mit UTF-8) aber nicht kompatibel mit ISO Latin 1.

  Im Unicode sind 17 "Planes" vorgesehen, die sich in jeweils 16 bit codieren lassen, damit können über eine Million Zeichen codiert werden. Es sind mehrere externe Codierungen definiert, eine ist UTF-16, in der Zeichen der wichtigen "Basic Multilingual Plane" mit 16 bit codiert sind, und andere (sehr seltene) Zeichen mit zwei 16 bit Einheiten hintereinander. Sehr verbreitet ist UTF-8. Dort sind die Zeichen mit 1 bis 4 Bytes codiert. ASCII-Zeichen (Zahlwerte bis 127) benötigen nur ein Byte, insofern ist UTF-8 voll kompatibel zu ASCII. Die deutschen Umlaute (im Bereich zwischen 128 bis 255) benötigen aber schon zwei Bytes. In der normalen ISO Latin 1 Codierung benötigen sie nur ein Byte. Der Grund für den Unterschied ist, daß UTF-8 Bits benötigt, um anzuzeigen, daß es noch Folgebytes gibt. Dies ist bei ISO Latin 1 nicht nötig, da seine Codes immer in ein Byte passen.

Zeichen-Referenzen (4)

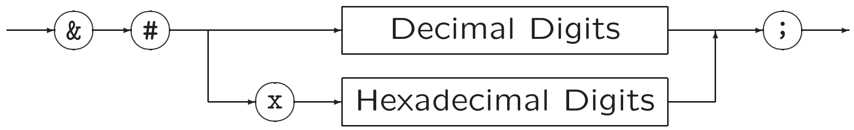
- Zeichen können als "Zeichen-Referenz" über ihren numerischen Code angegeben werden:

  `ä`

  ist z.B. ein "ä".
- Hexadezimale Notation ist auch möglich: `ä`
- Die Zahlwerte beziehen sich auf den Zeichenvorrat (z.B. Unicode), nicht auf die externe Codierung.
- Man kann z.B. extern (in der Datei) nur ASCII verwenden (besonders portabel), und dennoch alle Unicode-Zeichen darstellen.

Zeichen-Referenzen (5)

Character Reference:



- In DTDs können symbolische Namen ("Entities") für diese Zeichen-Referenzen definiert werden.
- Dann braucht man sich die Zeichencodes nicht zu merken.

  In HTML kann man z.B. "`ä`" ("a Umlaut") für "ä" schreiben (falls man extern reines ASCII benutzen will). In XML muß man sich solche Namen selbst definieren.

Zeichen-Referenzen (6)

- Zeichen-Referenzen können auch benutzt werden, um Zeichen einzugeben, die sonst eine spezielle Bedeutung in XML/SGML hätten.

  Man muß diese Zeichen "escapen", um ihnen ihre spezielle Bedeutung zu nehmen, so daß sie als normale Daten behandelt werden.
- Z.B. leitet das Zeichen "&" (ASCII 38) normalerweise eine Zeichen/Entity-Referenz ein. Wenn man es als gewöhnliches Zeichen in den Daten haben will, muß man `&` schreiben.

  Das Ergebnis einer Zeichen-Referenz sind immer normale Daten. Anderes Beispiel: Ein Anführungszeichen " (ASCII 34) ist in einem Attributwert enthalten, der mit " begrenzt ist: Als "`"`" schreiben.

Entity-Referenzen (1)

- Entities sind benannte Bausteine von Dokumenten.
- Mit einer Entity-Referenz kann den Inhalt eines Entities in ein Dokument einfügen.
- Entities können als Abkürzung verwendet werden. Z.B. kann man ein Entity “`ora`” mit dem Wert “`Oracle 8.1.6`” definieren (in der DTD).
- Anschließend wird die Entity-Referenz

  `&ora;`

  im Dokument durch “`Oracle 8.1.6`” ersetzt.

Entity-Referenzen (2)

Entity Reference:

In SGML ist das Semikolon “;” optional, wenn das folgende Zeichen nicht Teil des Entity-Namens sein kann (z.B. ein Leerzeichen). Wenn ein Zeilenende dem Entity-Namen folgt, wird es gelöscht. In XML ist das Semikolon “;” nötig.

Entities können nur im Inhalt eines Elementes oder im Wert eines Attributes oder im Ersetzungstext für ein anderes Entity verwendet werden. Man kann mit einem Entity also z.B. nicht den Elementtyp definieren oder den Namen eines Attributes.

Entity-Referenzen (3)

- SGML und besonders XML stellen sicher, daß sich durch “Expansion” einer Entity-Referenz (Ersetzen durch den Inhalt des Entities) keine syntaktischen Überraschungen ergeben.

  Wenn die öffnende Klammer “<” eines Tags, eines Kommentars, u.s.w. im Ersetzungstext für ein Entity enthalten ist, muß auch die passende schließende Klammer im Ersetzungstext des gleichen Entities enthalten sein. Bei XML muß sogar das Element (auch mit dem schließenden Tag) vollständig im Entity enthalten sein. XML Dokumente sollen ja auch ohne Kenntnis der DTD verarbeitet werden können, der Ersetzungstext ist also eventuell gar nicht bekannt.

  Wenn eine Entity-Referenz in einem Attributwert enthalten ist, werden eventuelle Anführungszeichen im Ersetzungstext nicht interpretiert, d.h. sie können nicht den Attributwert vorzeitig beenden.

Entity-Referenzen (4)

- In XML sind die folgenden fünf Entities vordefiniert (um Zeichen mit spezieller Bedeutung leichter als normale Daten eingeben zu können):
  - `&` für “&” (“ampersand”, “und”).
  - `<` für “<” (“less than”, “kleiner als”).
  - `>` für “>” (“greater than”, “größer als”).
  - `'` für “ ' ” (Apostroph).
  - `"` für “"” (“quotation mark”, Anführungsz.).

  In SGML sind diese Entities nicht vordefiniert. Deswegen wird aus Kompatibilitätsgründen empfohlen, sie auch in einer XML DTD explizit zu deklarieren.

Entity-Referenzen (5)

- Entities sind in SGML/XML ein recht mächtiges Konzept. Sie können z.B. auch für den Inhalt einer Datei stehen.

  Entities gelten in SGML/XML als physische Einheiten eines Dokumentes, Elemente als die logischen Einheiten.

- Es gibt auch Parameter-Entities, die nur in DTDs verwendet werden. Sie werden mit einem Prozentzeichen “%” statt einem “und”-Zeichen “&” angesprochen.

  Dies funktioniert nur in DTDs. Im eigentlichen Dokument-Inhalt hat das Prozentzeichen keine spezielle Bedeutung.

Markierte Abschnitte (1)

- Angenommen, ein XML/SGML-Dokument soll ein Teilstück in einer anderen Syntax (als reinen Text) enthalten. Eventuell enthält dieses Stück aber viele Vorkommen der Spezialzeichen “<”, >”, “&”.
- Es wäre sehr aufwendig, jedes Zeichen einzeln mit einer Zeichen/Entity-Referenz zu codieren.
- Für diesen Fall sind `CDATA`-Abschnitte gedacht. Sie können die Zeichen “<”, “>” und “&” als normalen Text enthalten (nicht als Markup interpretiert).

```
<![CDATA[...]]>
```


Markierte Abschnitte (2)

- `CDATA` Abschnitte kann man nicht schachteln.

  Der einzige Markup, der in einem `CDATA` Abschnitt erkannt wird, ist die Ende-Markierung “]]>”. Der Parser würde den Beginn eines geschachtelten Abschnittes nicht bemerken, aber beim ersten Vorkommen der Ende-Markierung den gesamten Abschnitt beenden. Die ungewöhnliche Syntax wurde gewählt, damit Ende-Markierungen nicht zufällig in den Daten vorkommen.

- `CDATA` Abschnitte können z.B. verwendet werden, um Beispiele von XML/HTML Code in eine XML-Datei einzufügen. In diesem Fall soll der Markup ja nicht interpretiert werden.

Kommentare (1)

- Kommentare können verwendet werden, um Erklärungen für den Leser der SGML/XML-Quelldatei in das Dokument einzufügen.
- Kommentare werden von Programmen ignoriert, die XML/SGML-Daten verarbeiten. Sie erscheinen also nicht in der formatierten Ausgabe.

  Der XML Standard erlaubt, daß ein XML Parser Kommentare an das Anwendungsprogramm liefert, aber er verlangt das nicht.

- Ein Kommentar in SGML/XML hat die Form

```
<!-- This is a comment -->
```

Kommentare (2)

- Kommentare können sich über mehrere Zeilen erstrecken.

  D.h. sie müssen nicht auf der gleichen Zeile geschlossen werden.

- Innerhalb eines Kommentars darf man nicht zwei aufeinander folgende Bindestriche schreiben: “--”.

  In SGML, erstreckt sich ein Kommentar eigentlich von “--” bis “--”. Er darf aber nur innerhalb von Markup Deklarationen verwendet werden. Eine Deklaration beginnt mit “<!” und endet mit “>” (sie darf ansonsten leer sein). Das erklärt die komplizierte Syntax für den Kommentar. In SGML können Kommentare auch in richtigen Element-, Attribut- u.s.w. Deklarationen verwendet werden, in XML geht dies nicht. XML erlaubt nur “<!-- ... -->”. In SGML würde “--” innerhalb des Kommentares den Kommentar beenden.

Kommentare (3)

- Offiziell sind Tags innerhalb von Kommentaren erlaubt (uninterpretiert), aber sie verwirren manche Browser.

  Zumindest bei HTML-Dateien versuchen Browser Syntaxfehler zu korrigieren (bei XML/XHTML müssen sie eigentlich einen Fehler melden). Wenn sie ein Tag sehen, vermuten sie eventuell, daß der Benutzer die Kommentarende-Markierung vergessen hat.

- Man kann Kommentare überall außerhalb von anderem Markup verwenden.

  Also z.B. nicht in Tags.

Kommentare (4)

- Man beachte, daß Kommentare in HTML Seiten dem Leser nicht wirklich verborgen sind:
  - ◇ Sie werden zwar vom Web-Browser normalerweise nicht angezeigt,
  - ◇ aber der Benutzer kann den Menüpunkt “View Document Source” des Browsers wählen.

    Falls dies nicht vorhanden ist, könnte er die Seite auch ohne Browser direkt mit `telnet` abrufen (siehe Kapitel über HTTP).

    Kommentare werden also nicht auf dem Web-Server ausgewertet (entfernt), sondern mit dem HTML Quelltext ausgeliefert. In ähnlicher Weise sind “versteckte Felder” in Formularen ungeeignet für geheime Passworte und ähnliches.

Processing Instructions

- “Processing Instructions” sind Anweisungen an Programme, die SGML/XML-Daten verarbeiten.
- Processing Instructions beginnen mit “<?” und enden mit “?>” (XML) bzw. “>” (SGML).

  SGML ist stark parametrisiert, und man kann natürlich auch “?>” für den Parameter “`pic`” (processing instruction close) wählen. Es gibt aber eine “Reference Concrete Syntax” mit der obigen Setzung.

- In XML identifiziert das erste Wort in der Processing Instruction das Programms, an die sich der Rest der Anweisung wendet (“target”).

  Andere Programme ignorieren die PI wie einen Kommentar.

XML Deklaration (1)

- XML Dokumente sollten mit einer XML Deklaration beginnen, die die XML Version angibt:

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

- Durch die explizite Angabe der verwendeten Version kann ein eventueller zukünftiger XML-Prozessor in einen Kompatibilitätsmodus schalten, um alte XML Dateien zu verarbeiten.
- Version `1.0` ist sehr verbreitet, und es wird empfohlen, sie zu verwenden, wenn man nicht die neuen Möglichkeiten von Version `1.1` benötigt.

XML Deklaration (2)

- Da viele Standards auf der XML Syntax aufbauen, sind größere Änderungen nicht zu erwarten.
- Auch die Unterschiede zwischen Version 1.0 und Version 1.1 sind minimal.

  Es gibt neue Auflagen der W3C Empfehlung für XML 1.0, aber sie korrigieren nur Fehler oder stellen kritische Punkte klarer. Die W3C Empfehlung für XML 1.0 wurde am 10.02.1998 veröffentlicht, eine zweite Auflage erschien am 06.10.2000, die dritte Auflage der XML 1.0 Empfehlung erschien am 04.02.2004 zusammen mit der ersten Auflage der Empfehlung für XML 1.1. Die aktuelle, vierte Auflage der XML 1.0 Empfehlung erschien am 16.08.2006 zusammen mit der zweiten Auflage der XML 1.1 Empfehlung. Beide wurden am 29.09.2006 noch einmal geändert. [`http://www.w3.org/XML/`].

XML Deklaration (3)

- Änderungen von Version 1.0 zu Version 1.1:
  - ◇ Mehr Zeichen sind in Namen zulässig.

    In XML 1.0 wurden die gültigen Zeichen in Elementtypen etc. explizit aufgezählt. Da der Unicode Standard noch erweitert wird, schien es nun besser, die verbotenen Zeichen aufzulisten. Zeichen werden nur noch verboten, wenn es einen Grund dafür gibt.
  - ◇ Zeilenenden von IBM Großrechnern unterstützt.
  - ◇ Die Regeln für Steuerzeichen ändern sich etwas.

    Zeichen-Referenzen für Steuerzeichen im Bereich `x01`–`x1F` sind nun erlaubt, Steuerzeichen im Bereich `x7F`–`x9F` (außer Leerplatz) müssen nun als Zeichen-Referenzen geschrieben werden.
  - ◇ Normalisierung erlaubt binären Vergleich.

XML Deklaration (4)

- Für SGML Prozessoren ist die XML Deklaration einfach eine Processing Instruction.

  Alte Browser können dadurch aber verwirrt werden, weil sie nicht einen allgemeinen SGML Parser enthalten, sondern nur die in HTML übliche Syntax verstehen. Moderne Browser unterstützen natürlich XHTML.
- Die XML Deklaration ist (auch in XML Dateien) optional (bei Verwendung von UTF-8 / UTF-16).
- Wenn man sie aber verwendet, muß sie ganz am Anfang der XML Datei stehen.

  Nicht einmal Leerplatz / Kommentare sind davor erlaubt. Der Grund ist, daß sie auch zur Erkennung der externen Zeichen-Codierung dient.

XML Deklaration (5)

- XML Prozessoren müssen als Zeichen-Codierung mindestens UTF-8 und UTF-16 von Unicode verarbeiten können (UTF-8 beinhaltet ASCII).

  UTF-16 codierte Dateien müssen mit dem "Byte Order Mark" `#xFEFF` beginnen. Dadurch ist eine automatische Unterscheidung von UTF-8 möglich, sowie eine Erkennung der Bytereihenfolge der 16 Bit Zahlen.

- Für andere Codes (z.B. ISO Latin 1) ist eine XML-Deklaration mit der Angabe der Codierung nötig:

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

  Wenn die Codierung für die in der XML Deklaration verwendeten Zeichen (reines ASCII) mit UTF-8 kompatibel ist, kann der XML Prozessor das `encoding` lesen, ohne die genaue Codierung zu kennen.

DOCTYPE Deklaration (1)

- Für jedes SGML-Dokument muß es eine DTD geben, die die Syntax des Dokumentes beschreibt.

  Die DTD muß nicht explizit im Dokument genannt sein, sie kann auch in die Software (z.B. den Web Browser) eingebaut sein.

- In XML ist die DTD optional. Man unterscheidet zwei Klassen von XML Dokumenten:
  - ◇ Wohlgeformte Dokumente ("well-formed XML") erfüllen die allgemeinen XML-Syntaxregeln.
  - ◇ Valide Dokumente haben eine zugehörige DTD und erfüllen die Syntaxregeln dieser DTD.

DOCTYPE Deklaration (2)

- Die Prüfung der Syntax eines XML-Dokumentes bezüglich einer DTD heißt entsprechend "validieren" des Dokumentes.
- Selbst wenn es eine DTD gibt, muß ein XML Prozessor sie nicht lesen und das Dokument validieren.

  Entsprechend unterscheidet die XML Spezifikation zwischen "validating" und "non-validating XML processors".

  Im Gegensatz dazu benötigt die Syntaxanalyse für SGML unbedingt eine DTD, weil die Markup Minimierung (optionale Start- und End-Tags) davon abhängt.

DOCTYPE Deklaration (3)

- Der Verweis auf eine zugehörige DTD geschieht am Anfang des Dokumentes (ggf. nach der XML Deklaration und eventuell Kommentaren etc.):

```
<!DOCTYPE FEUERWERKSARTIKEL SYSTEM "fw.dtd">
<FEUERWERKSARTIKEL>
    ...
</FEUERWERKSARTIKEL>
```

- Die Datei "`fw.dtd`" enthält dann die Deklaration von Elementen, Attributen, Entities.
- Der Dokument-Typ muß immer so heißen wie das äußerste Element ("document element").

DOCTYPE Deklaration (4)

- Für allgemein bekannte DTDs kann man auch einen “Public Identifier” benutzen, in XML benötigt man aber zusätzlich einen System Identifier (eine URI).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
    ...
</HTML>
```

Typischerweise hat so ein System eine Konfigurationsdatei, in der Public Identifier auf lokale Dateien mit den DTDs abgebildet werden (falls die Software nicht ohnehin auf eine ganz bestimmte DTD beschränkt ist). Dann wäre für die Syntaxanalyse kein Netzwerk-Zugriff nötig, es würde die lokale Datei benutzt.

DOCTYPE Deklaration (5)

- Man kann die DTD auch im Dokument selbst definieren:

```
<!DOCTYPE FEUERWERKSARTIKEL [
    <!ELEMENT FEUERWERKSARTIKEL ...>
    ...
]>
<FEUERWERKSARTIKEL> ... </FEUERWERKSARTIKEL>
```

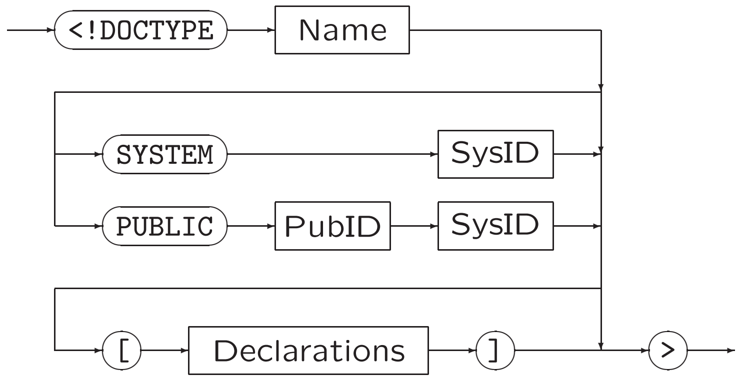
- Auch eine Mischung von beidem ist möglich:

```
<!DOCTYPE FEUERWERKSARTIKEL SYSTEM "fw.dtd" [...]>
```

Z.B. kann man im “internal subset” [...] Entity-Definitionen aus der Datei überschreiben.

DOCTYPE Deklaration (6)

DOCTYPE Declaration:



XML Dokument

- Ein XML Dokument besteht aus:
  - ◇ Einer XML Deklaration (optional, empfohlen).
  - ◇ Kommentaren, Processing Instructions, Leerplatz (optional).
  - ◇ Einer Dokumenttyp Deklaration (`DOCTYPE`, opt.).
  - ◇ Kommentaren etc. (optional).
  - ◇ Einem Element (“document element”, nötig).

    Darin geschachtelt alle anderen Elemente, Text, u.s.w.
  - ◇ Kommentare etc. (optional).

Aufgabe

Welche Syntax-Fehler enthält die folgende Datei?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GradesDB3>
    <student sid='101' first='Ann' last="Smith"
        <result cat='H' eno = 1  points=10>
        <result cat='H' eno ='1' points='8'>
        <result cat='M" eno ="1" points='12'
            >
    </ student >
    <!------------ Exercises ------------>
    <ex cat='H' eno='1' note='<em>difficult</em>'>
        Rel&#97 tional Algebra</ex>
</Grades-DB>
```


Software (1)

- Although browsers are very generous with syntax errors in HTML documents, they show all errors in XML documents.

  E.g. Internet Explorer, Firefox.

- They check only the syntax of well-formed XML, they do not validate documents against a DTD.
- If no style sheet is given, the document tree is displayed (child nodes are indented under the parent).

  It is possible to collapse/expand subtrees by clicking on the -/+ in front of the elements.

Software (2)

- Xerces from the Apache Software Foundation is an example for a validating parser for XML (supporting DTDs and XML Schema).

  See [http://xerces.apache.org/]. It has a DOM and a SAX interface for accessing the parsed data. It comes with a test program domprint, which can be used for checking the syntax (it is an unparser, i.e. it outputs the result of parsing again as XML, but probably differently formatted). There is a C++ and a Java version, and a Perl interface to the C++ version.

- There are also validation services on the web, e.g.
  - ◇ [http://www.stg.brown.edu/service/xmlvalid/]
  - ◇ [http://www.validome.org/xml/]

Overview

1. Introduction
2. XML Documents (Syntax)
3. Document Type Definitions (DTDs)
4. DOCTYPE, XML Declaration
5. Entities, Notations, Marked Sections

Example

Simple DTD for a HTML-Subset:

```
<!ELEMENT html  (head, body)>
<!ELEMENT head  (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body  ((#PCDATA|p|em|ul)*)>
<!ELEMENT p     ((#PCDATA|em|ul)*)>
<!ELEMENT em    (#PCDATA)>
<!ELEMENT ul    (li+)>
<!ELEMENT li    ((#PCDATA|p|em|ul)*)>
```


Element-Type Declarations (1)

An Element-Type Declaration consists of:

- "<!" followed by the keyword "ELEMENT".

  In SGML, one could define a different string instead of "<!". This is the parameter MDO ("Markup Deklaration Open Delimiter") in the SGML declaration. Correspondingly, ">" is called MDC ("Markup Declaration Close Delimiter"). XML is based on a fixed SGML declaration, so one cannot change these delimiters.

- Name of the element type to be declared.

  Such names are officially called "Generic Identifiers".

- Then one specifies what is permitted as content of this type of elements ("content model").
- ">".

Element-Type Declarations (2)

Element-Type Declaration:

White space is required between "<!ELEMENT" and the name, and between the name and the content specification. It is permitted but not required between content specification and the ">".

Names in XML must start with a letter, an underscore "_" or a colon ":", and can otherwise contain letters, digits, periods ".", hyphens "-", underscores "_", colons ":", or certain special Unicode characters. Names starting with "xml" in any capitalization are reserved, the colon is treated specially by the XML namespace standard.

The element type declaration is SGML is more complicated: There, also specifications for markup minimization are required (if the markup minimization parameter OMITTAG is set), "exclusions" and "inclusions" are possible, several element types can be declared together, etc.

Content Specifications (1)

- The building blocks of content specifications are
  - ◇ Names X of element types: This pattern matches exactly one element of type X, i.e. basically <X>...</X>.
  - ◇ The keyword #PCDATA: Pure textual data without tags (but possibly character/entity references).

    #PCDATA stands for "Parsed Character Data". The text is syntactically analyzed in order to check that it does not contain tags and in order to resolve entity and character references. In SGML (but not in XML) there is also "CDATA", which is not syntactically analyzed (like "verbatim" in LaTeX). The use of #PCDATA in model groups is very restricted in XML, see below.

Content Specifications (2)

- One can specify the optionality/multiplicity of elements and groups by attaching occurrence indicators:
  - A?: Optional, non repeatable (0 or 1 time).
  - A*: Optional, repeatable (0 or more times).
  - A+: Required, repeatable (1 or more times).

Content Specifications (3)

- Content specifications can be connected with
  - (A | B): "A or B".

    The content must match A or B.
  - (A , B): "First A, then B" ("A followed by B").

    A prefix of the content must match A, the rest B.
- In SGML, there is also (not supported in XML):
  - (A & B): "A and B".

    A and B must both appear, but in arbitrary sequence. This is equivalent to ((A,B) | (B,A)). Therefore, & is not strictly necessary. But rewriting an "and" with many components in this way becomes clumsy. There are also restrictions because of the deterministic parsing requirement, see below.

Content Specifications (4)

- Model groups consisting of more than two components are also possible:
  - (A_1|...|A_n): "Alternative" / "Choice" (one of the A_i).
  - (A_1,...,A_n): "Sequence" (all A_i in the given sequence).
- The A_i are
  - An element type (possibly with ?/*/+).
  - #PCDATA (in XML with restrictions, see below).
  - A nested model group (possibly with ?, *, +).

Content Specifications (5)

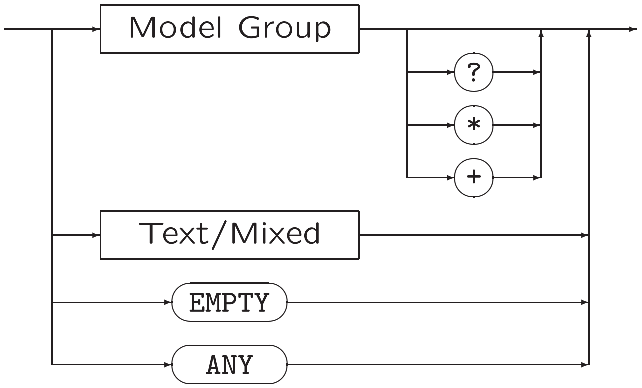
- A content specification ("content model") is
  - A model group (possibly only of one element),

    Element types must always be specified within parentheses. XML has special restrictions for mixed content, see below.
  - the keyword EMPTY: No content permitted.
  - The keyword ANY: Character data and elements of arbitrary type.
- SGML has also CDATA and RCDATA.

  For CDATA, even the special characters <, >, & can be used. They are not interpreted. RCDATA is the same, but & is interpreted (i.e. one can use character/entity references). In XML, one can use a CDATA-Section (see below) to include arbitrary text data in an XML document.

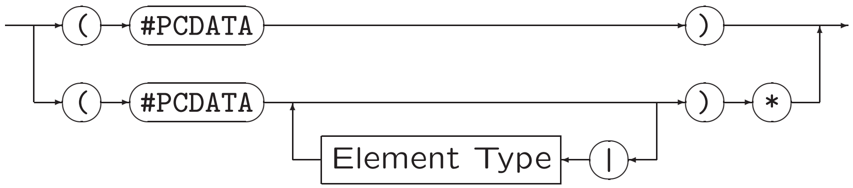
Content Specifications (6)

Content:



Content Specifications (7)

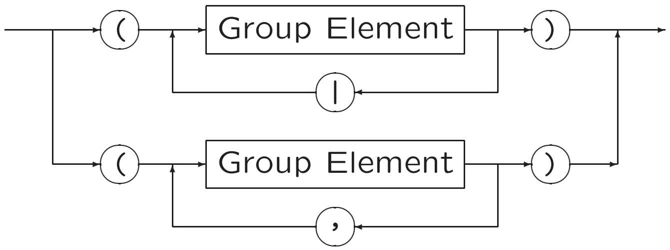
Text/Mixed:



- In XML, the only content models that can contain `#PCDATA` are (SGML has no such restriction):
  - ◇ `(#PCDATA)`
  - ◇ `(#PCDATA | Element-Type | ... | Element-Type)*`

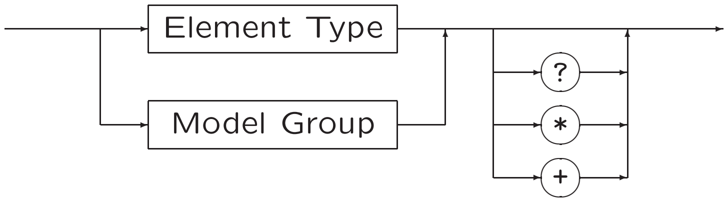
Content Specifications (8)

Model Group:



Content Specifications (9)

Group Element:



Content Specifications (10)

- In SGML and XML, the possible occurrence of white space is defined by the grammar.
- It is permitted but not required between each two tokens ("word symbols") in content models, except before the occurrence indicators "?", "*", "+".
- The keyword "#PCDATA" requires the symbol "#" (RNI, "Reserved Name Indicator") in order to distinguish it from an element type named "PCDATA".

  Other keywords like "EMPTY" do not use it, since in the element type declaration, they appear outside of parentheses, while user-defined names must appear inside parentheses.

Content Specifications (11)

- In SGML and in XML, content models must be not ambiguous. E.g. the following is forbidden:

  ```
  <!ELEMENT E ((A, B?), B)>
  ```

  When the parser has read an A and sees a B, it is not clear whether this is the optional B in the middle or already the required B at the end.

  The parser could solve this problem by looking ahead to see whether after the B in question there is another B. However, the SGML standard explicitly states: "an element or character string that occurs in the document instance must be able to satisfy only one primitive content token [in the content model] without looking ahead in the document instance." A primitive content token is an element type or #PCDATA.

Content Specifications (12)

- Another example for an ambiguous content model:

  ```
  <!ELEMENT E ((A, B) | (A, C))>
  ```

  When the parser sees the element A, it does not know which path to follow in the content model.
- This requirement simplifies the task of checking the input with respect to a given DTD.

  There are standard techniques for generating a nondeterministic finite automaton for a given regular expression. Normally, one would need to translate this into a deterministic automaton, which can lead to an exponential increase in the number of states. SGML and XML are restricted in such a way that the constructed automaton is already deterministic.

Attribute Declarations (1)

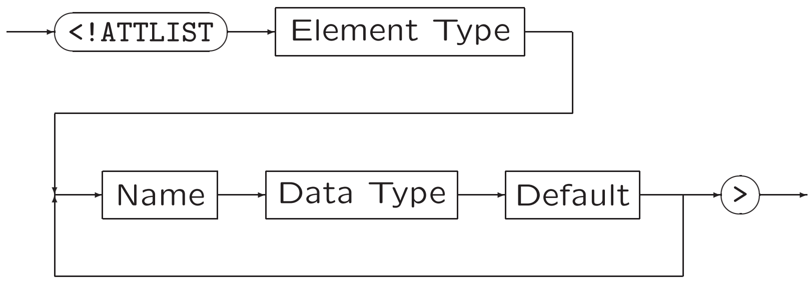
- Example (symbol used for marking list items):

  ```
  <!ATTLIST UL type (disc|square|circle) #IMPLIED>
  ```

  In HTML 4.01 Strict this attribute was removed.
- Several attributes (of one element type) can be declared in a single ATTLIST command.
- E.g. some attributes of images in HTML:

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

Attribute Declarations (2)



- For each attribute, three things are defined: Name, data type, and default value.

  White space is required between each two components of the `ATTLIST` command, except before the final “>”, where it is optional.

Attribute Declarations (3)

- The same attribute name can appear in an `ATTLIST` declaration only once.

  This is clear: There cannot be conflicting definitions for an attribute in the same declaration.

- If there are several `ATTLIST` declarations for the same element type, they are merged. The first declaration for an attribute becomes effective, all other declarations for the same attribute are ignored.

  This might be useful if a DTD is constructed in several pieces. It is however recommended (required in SGML?) that for every element type, there is only one `ATTLIST` declaration which defines all its attributes.

Attribute Data Types (1)

- E.g. `(yes|no)`: Enumeration type.

  The attribute value must be one of the listed values. Each value is a “name token” (NMTOKEN), i.e. a sequence of characters that can appear anywhere in identifiers (letters, digits, and certain special characters). E.g. a sequence of digits would be valid. In SGML, it is forbidden that same enumeration type value is used for two attributes of the same element type. In XML, this is recommended “for interoperability”.

- `CDATA`: Sequence of arbitrary characters.

  The character “`&`” is interpreted, i.e. one can use character and entity references in the attribute values. In XML, “<” is forbidden in attribute values (so that missing quotes are easier found), and “>” is not interpreted (treated as data). In SGML, “<” and “>” are valid, but not interpreted. Thus, attribute values still cannot contain elements.

Attribute Data Types (2)

- `ID`: A name that uniquely identifies this element (within the entire document).

  The syntax is the same as for element type names (sequence of letters and digits plus `_`, `:`, `.`, `-`, starting with letter or `_`, `:`, `.`). Two elements must not have the same value for an attribute of type `ID`. This even holds for elements of different type. The same element type cannot have two attributes of type `ID`. One should use the same name for all attributes of type `ID`, and the attribute name “`ID`” is very common.

- `IDREF`: A name that appears as value of an `ID`-attribute somewhere in the document.

- `IDREFS`: List of `IDREF`-values.

  The single values are separated by white space.

Attribute Data Types (3)

- NMTOKEN: Sequence of name characters.

  An arbitrary sequence of letters, digits, “_”, “-”, “.”, and “:”.

- NMTOKENS: List of NMTOKEN-values.
- ENTITY: Name of an entity.

  Entities are a kind of macros or include files (see below). An attribute of type ENTITY takes as value the name of a declared unparsed entity.

- ENTITIES: List of ENTITY-values.
- NOTATION (N_1|...|N_m): One of the notations N_i.

  The N_i must be declared as notations (data formats). Only one attribute of an element type can have the type NOTATION. This attribute defines the format of the content of the element.

Attribute Data Types (4)

- In summary, XML supports the following attribute data types:
  - Enumeration types,
  - CDATA,
  - ID, IDREF, IDREFS,
  - NMTOKEN, NMTOKENS,
  - ENTITY, ENTITIES,
  - enumerations of notations.
- SGML has in addition NAME, NAMES, NUMBER, NUMBERS, NUTOKEN, NUTOKENS. E.g. NUMBER: sequence of digits.

Default Values (1)

- One must specify what should happen if an element of the type has not defined a value for the attribute.
- One possibility is to specify a default value:

```
<!ATTLIST UL type (disc|square|circle) "disc">
```

  The quotation marks around the default value are not required in SGML, but they are required in XML. This is a bit inconsistent, since in accordance with SGML, there are no quotation marks in the enumeration of possible values. In SGML, attribute values that are NMTOKENS do not need quotes.

- Then the tag <UL> in the document is equivalent to

  <UL type="disc">.

Default Values (2)

- Instead of a default value, one can also specify:
  - #IMPLIED: The attribute is optional.

    I.e. the default value is a “null value” different from all possible normal values. The name for the keyword was chosen because it is assumed that the application program can compute a value for the attribute. E.g. a chapter number is usually the number of the last chapter plus 1.

  - #REQUIRED: An attribute value must be specified.
  - #FIXED "Value": The attribute can have only this single value that is specified in the DTD.

    This is e.g. used when many/all element types have an attribute with the same name, and for each element type a (possibly different) value is declared in the DTD.

Exercise (1)

Please find syntax errors:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE GradesDB4 [ <!-- contains syntax errors -->
    <!ELEMENT GradesDB4 (STUDENT, RESULT)*>
    <!ELEMENT STUDENT RESULT+>
    <!ATTLIST STUDENT FIRST CDATA #REQUIRED
                      LAST  CDATA #REQUIRED>
    <!ELEMENT RESULT #EMPTY>
    <!ATTLIST RESULT EX_ID  IDREF   #REQUIRED
                     POINTS NMTOKEN #REQUIRED>
    <!ELEMENT EXERCISE #PCDATA>
    <!ATTLIST EXERCISE ID ID #REQUIRED>
]> <!-- continued on next slide -->
```


Exercise (2)

Please validate against DTD on last slide:

```
<GradesDB4>
    <student sid='101' first='Ann' last='Smith'>
        <email>smith@acm.org</email>
        <result ex_id='H1' points='A+'/>
        <result ex_id='2' points='8'/>
        <result ex_id='M1' points='12 points'/>
    </student>
    <student first='Maria' last='Brown'/>
    <exercise id='H1'>Relational Algebra</exercise>
    <exercise id='2'>SQL</exercise>
</GradesDB4>
```


Exercise (3)

Please develop a DTD for this document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smallbusiness>
    <product id='P01' name='Apple' price='0.40'>
        Really <em>delicious</em>Apples!</product>
    <product id='P02' name='Banana' price='0.50'>
        The best bananas!</product>
    <order id='R100' customer='Ann Smith'/>
        <item prodid='P01'/>
        <item prodid='P02' quantity='5'/> </order>
    <order id='R100' customer='Maria Brown'/>
        <item prodid='P01' quantity='3'/> </order>
</smallbusiness>
```