XML 1.0 - Die Spezifikation

Schrittweise Erarbeitung
Kommentare und Beispiele

XML 1.0 - die Spezifikation

- Terminologie
  - XML 1.0 verwendet einige Begriffe in bestimmter Art und Weise. Bitte von deren umgangssprachlichen Gebrauch ggf. unterscheiden!
  - Begriffe/Ausdrücke mit präzisierter Bedeutung:
    - *may, must, error, fatal error, at user option*
    - *validity constraint, well-formedness constraint*
    - *match*
    - *for compatibility, for interoperability*
  - Bem.: Diese Liste hier soll nur sensibilisieren für reservierte Begriffe. Definitionen ggf. direkt in den Spezifikationen nachlesen.

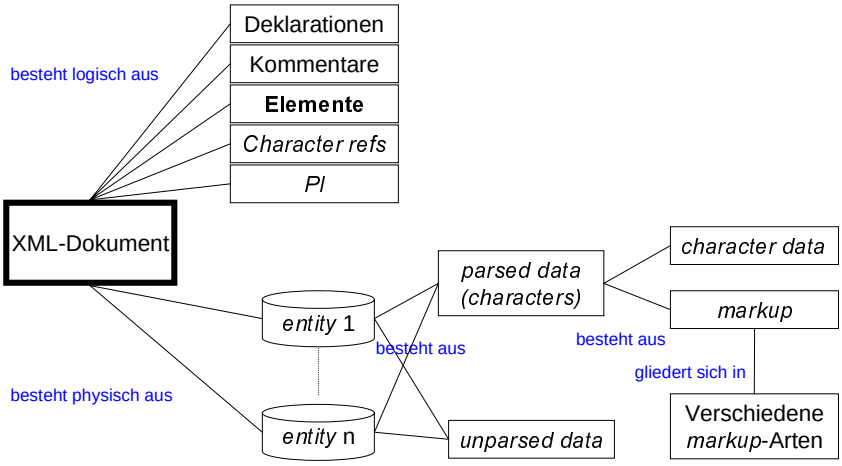
XML 1.0 - die Spezifikation

- Methodik
  - Die nächsten Abschnitte gehen deduktiv vor, denn
  - wir benötigen erst einmal ein formales Rüstzeug, um die Bestandteile von XML präzise beschreiben zu können.
  - Damit wird dann klar werden, was genau in XML erlaubt ist.
  - Zahlreiche Parserfehler liegen in Verletzungen von XML-Regeln begründet, die ohne deren genaue Kenntnis sehr schwer zu beseitigen sind.
  - Ist diese „Durststrecke“ erst überwunden, läßt sich mit XML-Dokumenten umso leichter arbeiten.
  - Früh eingeführte, aber erst später definierte Begriffe dienen der Systematik und dem späteren Nachschlagen der Zusammenhänge. Also: Nicht wundern, wenn sie zunächst nicht verständlich sind.
  - Bei Gefahr des „Verdurstens“ bitte mit Fragen unterbrechen!

XML-Dokumente: Aufbau

- Datenobjekt
  - XML-Dokument, wenn „wohlgeformt“ im Sinne der XML 1.2 Spezifikationen
  - „gültiges“ XML-Dokument, wenn zusätzlich konsistent mit deklarierter DTD
- XML-Dokument
  - Physischer Aufbau:
    - *Entities*
  - Logischer Aufbau:
    - (siehe Forts.)
- XML-Dokument (Forts.)
  - Logischer Aufbau (Bestandteile):
    - Deklarationen
    - Elemente
    - Kommentare
    - *character references*
    - *processing instructions (PI)*

XML-Dokumente: Aufbau



XML-Dokumente: *Markup*-Arten

- XML-Dokument
  - *Markup*
  - *character data* (der „Rest“)
- *Markup*-Arten
  - ***Tags**: start-tags, end tags, empty-element tags*
  - ***References**: Entity refs., character refs.*
  - ***Comments***
  - *CDATA section delimiters*
  - ***Declarations**: document type, element, attribute, notation, text, XML decl.*
  - ***Processing instructions***
  - *White space* außerhalb des *(vor dem) root element*

- Bemerkungen
  - Die nebenstehende Liste aller *markup*-Arten wird im Folgenden nach und nach vorgestellt.
  - Interessant ist hier ihre Vollständigkeit. Letztlich sollte man jede *markup*-Art kennengelernt haben.
  - Die Liste dient als Leitfaden durch die spezifschen Abschnitte
  - Man mache sich klar, dass wirklich jedes Zeichen, das nicht *markup* ist, irgendwo als *character data* auftauchen muss - z.B. auch Zeilenumbrüche!

Grundlagen zur XML 1.0-Spezifikation

Allgemeines zu den Regeln Zeichensätze

Die Spezifikationsregeln

- Vorbemerkungen
  - Die XML 1.0 Spezifikationen sind in Form von 89 Regeln („*productions*“) formal präzisiert.
  - Diese formale Grammatik von XML wird in einer einfachen „*Extended Backus-Naur Form“ (EBNF*) beschrieben.
  - Die Notation ist in Abschnitt 6 am Ende der Spezifikationen definiert.
  - Genereller Aufbau:
    [n] *symbol ::= expression*
  - Besonderheiten
    - Die Regeln werden durchnummeriert ([n]).
    - Symbole, die den Ausgangspunkt einer „regulären Sprache“ bilden, fangen mit Großbuchstaben an.
    - *literal strings are quoted*
  - Beispiel
    - [99] meinBspSymbol ::= ‘Dieser Ausdruck ist ein konstanter String‘

Die Spezifikationsregeln

- Zeichensatz-Angaben
  - Da XML 1.0 generell auf Unicode basiert, werden konkrete Zeichen (*characters*) grundsätzlich über ihre Unicode-IDs (USC-4) spezifiziert.
    - Erinnerung: Unicode ist in UTF-8 Notation zu 7-bit ASCII abwärtskompatibel
  - Der numerische ID-Wert eines Zeichens wird - meist in hexadezimaler Form - wie folgt angegeben:
    #xN
  - mit N stellvertretend für eine beliebige Folge hexadezimaler Ziffern
    - Hexadezimal-Ziffern sind 0, 1, …, 9, A, B, C, D, E, F
    - Führende Nullen sind nicht signifikant und dürfen ausgelassen werden.
  - Beispiel:
    Der Buchstabe „A“ läßt sich z.B. wie folgt angeben:
    - #x41 oder auch
    - #x0041

Die Spezifikationsregeln

- Man beachte Besonderheiten von Unicode:
  - Basiszeichen
    - Unser normales Verständnis eines Z.
  - Ideographische Zeichen
    - z.B. fernöstliche wie Kanji-Zeichen
  - *combining characters*
    - „Pünktchen“, Akzentzeichen u.a.
    - Sie ergeben zusammen mit ihrem jeweiligen Vorläuferzeichen in einem String das endgültige Symbol
    - Beispiel: à = a`
    - Diese Zeichenkombinationen ergänzen die bereits vorhandenen Spezialzeichen
    - Die Kombinationsmethode schafft mit relativ wenigen Unicode-Einträgen eine große Vielfalt an möglichen Symbolen.
  - *extenders*
    - (Unicode-Spezialthema, hier nicht behandelt)

Die Spezifikationsregeln

Notationen

#xN	einfaches Zeichen
[a-zA-Z], [#xN-#xN]	Zeichenbereiche und ...
[abc],[#xN#xN#xN]	… Zeichenlisten. Listen und Bereiche sind mischbar!
[^a-z], [^#xN-#xN]	Auszuschließende Zeichenbereiche
"*string*", '*string*'	Konstante Strings
(*expression*)	Klammerung von Ausdrücken
A B	Ausdruck A gefolgt von B
A | B	A oder B
A - B	A ohne B (Mengendifferenz)
A?	String passt höchstens einmal zu A
A+	String passt ein- oder mehrmals zu A
A*	String passt beliebig oft zu A
/* … */	Kommentar (in der Grammatik)
[wfc: …], [vc: …]	*Well-formedness or validity constraint*

Zeichen und Zeichenketten

Zulässige Zeichen, *white space*
character references, character data
predefined entitites
names, name tokens, literals

XML characters, White Space

- Zeichen (characters):
  ```
  [2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] |
      [#xE000-#xFFFD] | [#x10000-#x10FFFF]
  /* any Unicode character, excluding the surrogate
  blocks, FFFE, and FFFF. */
  ```
  unterteilt in
  - Buchstaben (letters)
  - Ziffern (digits)
  - Andere Zeichen (other char)
- „White Space“:
  ```
  [3] S ::= (#x20 | #x9 | #xD | #xA)+
  ```
  - Also: Beliebige Zeichenfolgen bestehend aus *blank, form feed* (FF), *carriage return* (CR) oder *line feed* (LF)

Character references

- Formale Definition
  ```
  [66] CharRef ::=
      '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'
      [WFC: Legal Character]
  ```
- Bemerkungen
  - Hier die allgemeine Regel!
  - Angaben entweder in dezimaler oder in hexadezimaler Notation
  - Andere sonst gebräuchliche Notationen (binär, oktal) sind nicht zulässig.
  - Bei hex-Darstellung sind kleine wie große Ziffern-Buchstaben zulässig.
  - WFC - *Well-formedness constraint*:
    - Gemeint ist hier, dass das derart referenzierte Zeichen aus der Menge der in [2] beschriebenen „Char“ stammt
    - Nicht darin vorgesehene Zeichen führen zum Parser-Abbruch (*fatal error*)!

Character data

- `[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)`
- Bemerkungen
  - Innerhalb des Dokuments versteht man darunter alle Zeichen, die nicht *markup* sind.
  - Umschreibung, Fall 1: Außerhalb einer CDATA *section* (Normalfall)
    - Alle Zeichen außer den einleitenden Zeichen eines *tags* (<) bzw. einer *entity (&)*.
    - Beispiel: <greeting>Hello, world!</greeting>
  - Umschreibung, Fall 2: Innerhalb einer CDATA *section*
    - Alle Zeichen außer der Zeichenfolge, die CDATA *sections* beendet (]]>), danach „normal“ weiter.
    - Vorgriff: CDATA sections sind ein Konstrukt zur Aufnahme „wörtlicher“ Textpassagen, z.B. wenn man XML-Codebeispiele zitieren will.
    - LaTeX-Analogon: Die *verbatim*-Umgebung

Vordefinierte *general entities*

- Das Problem:
  - XML *parser* erkennen *markup* anhand bestimmter Zeichen (siehe die CharData-Definition)
  - Was tun, wenn eines dieser Zeichen als normales Textzeichen verwendet werden soll?
- Die Lösung:
  - Derartige Zeichen lassen sich indirekt - als *public entities* - in *character data* einbetten.
  - Analogie zu HTML
  - Die Methode gilt generell, nicht nur für die reservierten Zeichen.
    Tip: Liste gängiger Unicode-Spezialzeichen per *entity* zugänglich machen.
- Liste der (einzigen) in XML vordefinierten *entities*:
  - & &
  - < und > < und >
  - ' und " ‘ und “
  - Beispiel: „A<B & C>B“<Ungleich> A < B & C &gt B </Ungleich>

XML names, name tokens

- Einschränkung bei der Namenswahl
  - XML fordert die Einhaltung bestimmter Regeln bei der Vergabe von z.B. Element- und Attributnamen.
  - *name tokens* unterliegen nur geringen Einschränkunen, *names* dagegen mehr:
- `[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender`
- `[5] Name ::= (Letter | '_' | ':') (NameChar)*`
- `[6] Names ::= Name (S Name)*`
- `[7] Nmtoken ::= (NameChar)+`
- `[8] Nmtokens ::= Nmtoken (S Nmtoken)*`
- Bemerkungen
  - *names* fangen also immer mit einem Buchstaben, _ oder Doppelpunkt an
  - Namen, die mit (‘x‘|‘X‘) (‘m‘|‘M‘) (‘l‘|‘L‘) beginnen, sind von XML reserviert!
  - Der Doppelpunkt ist i.d.R. für XML-interne Zwecke reserviert - meiden!

XML names, name tokens

- Beispiele
  - <myElem myAttr = ‘value‘> … </myElem>
    korrekt
  - <myElem:1><myElem:2>…<myElem:2><myElem:1>
    Doppelpunkt im Namen - möglichst NICHT verwenden
  - <XML-Basis>hier mein Text zu diesem Inhalt…</XML-Basis>
    Verletzung der Reservierungsregel - „XML“ am Anfang des Elementnamens
  - <_myElem -myAttr = ‘…‘> … </myElem>
    Attributname fängt mit einem unzulässigen Zeichen an
  - <_myElem my-Attr = ‘…‘> … </_myElem>
    korrekt
  - <myElem .myAttr = ‘…‘> … </myElem>
    Punkt is kein erlaubtes Zeichen in einem (Attribut-)Namen

XML *literals*

```
[9] EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"' |
                    "'" ([^%&'] | PEReference | Reference)* "'"

[10] AttValue ::= '"' ([^<&"] | Reference)* '"' |
                  "'" ([^<&'] | Reference)* "'"

[11] SystemLiteral ::= ('"' [^"]* '"') | ("'" [^']* "'")

[12] PubidLiteral ::= '"' PubidChar* '"' |
                      "'" (PubidChar - "'")* "'"

[13] PubidChar ::=    #x20 | #xD | #xA | [a-zA-Z0-9] |
                      [-'()+,./:=?;!*#@$_%]
```

- *Literals* sind Strings (ohne die *quotation marks*)
- Sie werden verwendet als
  - Inhalte interner *entities (EntityValue),*
  - Attributwerte (*AttValue*)
  - Externe *identifier (SystemLiteral, PubidLiteral)*
- Man beachte die Doppel-Beschreibung für die beiden *quotation marks!*

XML *literals*

- Bemerkungen:
  - *Entities* werden mit & oder % eingeleitet, daher sollten ihre Werte mit diesen Zeichen gerade nicht beginnen.
  - Attributwerte sollten weder wie ein *tag* (<) noch wie ein *entity* (&) anfangen.
  - *System literals* dürfen aus allem bestehen, nur nicht aus dem jeweils verwendete *quotation mark* selbst.
  - *Public ID literals* lassen andere Zeichen zu als *name tokens*. Das wird verständlich, wenn man sich z.B. URLs darunter vorstellt.
  - Die Definition von *PubidLiteral* ist bez. der beiden *quotation marks* nur deshalb asymmetrisch, weil nur eines der beiden (') in der Menge der explizit erlaubten *PubidChar*-Zeichen auftaucht - und daher eine Sonderbehandlung benötigt.

Spezielle *markup*-Arten

Kommentare
Processing instructions
CDATA sections
Conditional sections

Markup: Übersicht nach Erscheinungsbild

Nicht-essentielle *markup*-Arten, Gegenstand dieses Kapitels:

`<!-- … -->`
Kommentar

`<?…?>`
XML-Deklaration, Text-Deklaration, PI (*processing instruction*)

`<![NAME[ … ]]>`
Besondere Code-Abschnitte; NAME aus {CDATA, INCLUDE, IGNORE}

Kern der XML-*Markups*, im folgenden Kapitel vorzustellen:

`<name attr="value"> … </name>, <name attr="value"/>`
Elemente und Attribute

`<!DOCTYPE NAME … [ … ] >`
Dokumententyp-Deklaration (hier: *root element* heißt „NAME“)

`<!NAME … >`
Markup-Deklaration. Einzelfälle:
NAME aus {ENTITY, ELEMENT, ATTLIST, NOTATION}

Kommentare

- Bemerkungen:
  - Kommentare sehen ähnlich aus wie in HTML. Sie sind nicht schachtelbar.
  - Sie zählen als markup und dürfen außerhalb anderen *markups* erscheinen.
  - Innerhalb von Kommentaren wird anderer *markup* überlesen.
  - XML Prozessoren dürfen Kommentare an Anwendungen durchreichen, sie müssen dies aber nicht.
  - ACHTUNG:
    - Kommentare sollen nicht zur Steuerung von Anwendungen mißbraucht werden!
    - ‚--' innerhalb Kommentarinhalten ist nicht zulässig!
    - Das letzte Zeichen des Kommentarinhalts darf kein ‚-' sein!
- Formale Definitionen:

```
[15] Comment ::= '<!--' ((Char - '-') |
                 ('-' (Char - '-')))* '-->'
```

- Beispiel:

```
<!-- Dies ist ein tag: <tag> -->          OK
<!-- B+, B, und B--->                     NICHT ZULÄSSIG!
<!-- eins -- zwei -- drei -->             NICHT ZULÄSSIG!
```


Processing instructions (PI)

- Bemerkungen:
  - Mit PIs schafft XML einen Standard zur Steuerung von Anwendungen.
  - Proprietäre Lösungen z.B. durch Konventionen innerhalb von Kommentaren sollen so vermieden werden. Beispiel Apache, SSI.
  - Dennoch: PI-Einsatz sollte man auf das Nötigste beschränken!
  - PIs gehören nicht zu den *character data*, sondern zum *markup*
  - PIs müssen vom *Parser* an die Anwendung durchgereicht werden.
- Formale Definitionen:

```
[16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'

[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

- Bemerkungen:
  - *Parameter entity references* werden innerhalb von PIs nicht expandiert.
  - PITarget identifiziert die Anwendung; siehe auch *notations*.
  - Nicht mit der XML Deklaration zu verwechseln - diese ist keine PI!
  - Meist folgt dem *PITarget* eine attribut-artige Liste von *key/value*-Paaren. Im Gegensatz zu Attributen ist die Reihenfolge der PI-Argumente aber wichtig!
- Beispiel:

```
<?xml-stylesheet href="http://..." type="text/xsl"?>
```

CDATA *sections*

- Bemerkungen:
  - CDATA = Character data
  - Ein reservierter Textbereich, den der Parser nicht interpretiert.
  - *Entities* und anderer *markup* werden innerhalb einer CDATA *section* nicht interpretiert - < würde nicht nach < übersetzt!
  - Praktisch z.B. wenn XML Quellcode selbst zur Anzeige als Text gebracht werden soll, da man das escaping der zahlreichen Markup-Zeichen vermeidet
- Formale Definitionen:

```
[18] CDSect ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20] CData ::= (Char* - (Char* ']]>' Char*))
[21] CDEnd ::= ']]>'
```

- Beispiel:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```


CDATA *sections*

- Beispiel für einen Konflikt mit *CDEnd*:

```
<![CDATA[
 Javascript code: if( a[c[5]]> 7 ) then …
]]>
```

- zu lösen etwa durch:

```
<![CDATA[
 Javascript code: if( a[c[5]]]]><![CDATA[> 7 ) then …
]]>
```

- oder - falls das Leerzeichen akzeptabel ist, einfach durch:

```
<![CDATA[
 Javascript code: if( a[c[5]] > 7 ) then …
]]>
```

–Leerzeichen eingefügt!

Conditional sections

- Bemerkungen
  - Diese Konstrukte werden nur in externen Deklarationsteilen verwendet.
  - Es ist damit möglich, DTDs systematisch für mehrere Zwecke in verschiedenen Varianten zu pflegen.
  - Formal ähnliche Notation wie bei CDATA, daher hier vorgestellt.
- Formale Regeln:

```
[61] conditionalSect ::= includeSect | ignoreSect

[62] includeSect ::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
                     [VC: Proper Conditional Section/PE Nesting]

[63] ignoreSect ::= '<![' S? 'IGNORE' S? '['
                     ignoreSectContents* ']]>'
                     [VC: Proper Conditional Section/PE Nesting]

[64] ignoreSectContents ::= Ignore ('<!['
                     ignoreSectContents ']]>' Ignore)*

[65] Ignore ::= Char* - (Char* ('<![' | ']]>') Char*)
```


Conditional sections

- Beispiel:

```
<!ENTITY % draft 'INCLUDE' >   <!-- Parameter entity:   -->
<!ENTITY % final 'IGNORE' >    <!-- typisch in docdecl. -->

<![ %draft; [    <!-- Wird hier zu INCLUDE expandiert -->
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![ %final; [    <!-- Wird hier zu IGNORE  expandiert -->
<!ELEMENT book (title, body, supplements?)>
]]>
```

- Bemerkungen
  - Je nach Definition von „draft“ bzw. „final“ (an einer zentralen Stelle) lassen sich so verschiedene Definitionen von „book“ in angepassten Varianten vorhalten - in derselben DTD.
  - Besonders in größeren / komplexen DTDs zu finden.
  - Vorsicht: Im Unterschied zu CDATA dürfen INCLUDE und IGNORE von *white space* umgeben sein.