



# XML 1.0 - Die Spezifikation

Schrittweise Erarbeitung  
Kommentare und Beispiele



## XML 1.0 - die Spezifikation



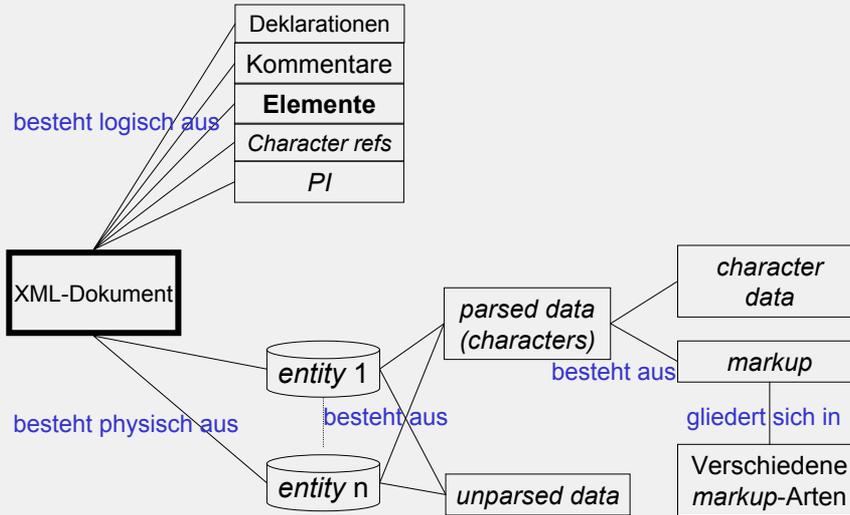
- Terminologie
  - XML 1.0 verwendet einige Begriffe in bestimmter Art und Weise. Bitte von deren umgangssprachlichen Gebrauch ggf. unterscheiden!
  - Begriffe/Ausdrücke mit präzisierter Bedeutung:
    - may, must, error, fatal error, at user option*
    - validity constraint, well-formedness constraint*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



- **Einschub: Zur 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!



- **Datenobjekt**
  - **XML-Dokument**, wenn „wohlgeformt“ im Sinne der XML 1.0 Spezifikationen
  - „gültiges“ XML-Dokument, wenn zusätzlich konsistent mit deklarierter DTD
- **XML-Dokument**
  - **Physischer Aufbau:**  
*Entities*
  - **Logischer Aufbau (Bestandteile):**  
Deklarationen  
Elemente  
Kommentare  
*character references*  
*processing instructions (PI)*



- 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, entity, 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 spezifischen Abschnitte
  - Man mache sich klar, dass wirklich jedes Zeichen, das nicht markup ist, irgendwo als character data auftauchen muss - z.B. auch Zeilenumbrüche!



## Das Visitenkarten-Beispiel

XML - ganz einfach (?)



## Das Visitenkarten-Beispiel



- <Visitenkarte layoutkey=>
  - Layoutschlüssel
  - Name
    - Vorname
    - Nachname
    - Akadem. Grad
    - Titel
    - MI, Gen qual.
  - Beruf
    - (job title)
  - Abteilung/Projektbezeichnung
  - Kontakt
    - Tel
    - Fax
    - email
    - WWW
- Visitenkarte (Forts.)
  - Firma
    - Firmeneinheit
    - Anschrift
      - Briefanschrift
      - Postfach
      - Ort
      - PLZ
      - Land
    - Hausanschrift
      - Straße, Hausnr.
      - Ort
      - PLZ
      - Land



- Element- und Attributnamen:
  - Was ist zulässig?
- Umlaute:
  - Wie codieren?
- Attribute:
  - Liste der zulässigen Codes?
- Elemente:
  - Wiederholungen? Inhalte?
- Leere Elemente / Attribute:
  - Was tun?
- Designfragen:
  - Element oder Attribut wählen?



# Grundlagen und erste Regeln der XML 1.0-Spezifikation

Tags und Referenzen  
Unicode und zulässige Zeichen in XML



## Regeln?



Beispiele:

1. <Beispiel>Hallo zusammen!</Beispiel>

2. <#Beispiel#>  
Halo zusammen!  
</#Beispiel#>

3. <#Beispiel Sprachschlüssel#=#"DE"##>  
Halo zusammen!  
<#/#Beispiel#>

Alles zulässig? Ggf: Wirkung?

XML benötigt  
präzise Regeln!



## Die Spezifikationsregeln



- Die XML 1.0 Spezifikationen sind in Form von **89 Regeln** („*productions*“) formal präzisiert.
  - Die 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*



## Die Spezifikationsregeln



- Regeln für „tags“:

```
[40] STag ::=
      '<' Name (S Attribute)* S? '>'
```

```
[41] Attribute ::= Name Eq AttValue
```

```
[42] ETag ::= '</' Name S? '>'
```

```
[44] EmptyElemTag ::=
      '<' Name (S Attribute)* S? '/>'
      [WFC: Unique Att Spec]
```

```
[25] Eq ::= S? '=' S?
```



## Die Spezifikationsregeln



- Also sind folgende tags korrekt:

```
- <Beispiel>,
- <Beispiel >,
- <Beispiel key = "value" >,
- </Beispiel > # OK
```

- aber jene sind FALSCH:

```
- < Beispiel>,
- </ Beispiel>,
- < /Beispiel>,
- < / Beispiel> # FALSCH!
```

- Noch zu klären:

```
- Regeln für: S, Name, AttValue
```



# Einschub: Unicode

... und andere Zeichensätze



## Vorbereitung: Zeichensatz-Angaben



- XML 1.0 basiert auf [Unicode/ISO10646](#). Daher werden konkrete Zeichen (*characters*) grundsätzlich über ihre Unicode-IDs (USC-4) spezifiziert. (Vergleiche dazu die Vorübung.)
- 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 u. dürfen ausgelassen werden.
- Beispiel:  
Der Buchstabe „A“ lässt sich z.B. wie folgt angeben:
  - `#x41`, `#x0041`, `#65`, `#0065`, ...



- Informationen:
  - <http://czyborra.com/>  
zu Zeichensätzen allgemein, insb. ISO-8859-x
  - <http://www.unicode.org/>  
Speziell zu Unicode
- Beispiel: Buchstabe „ü“
  - Codepage 437 (DOS): 0x81
  - ISO-8859-1: 0xFC
  - Unicode (composite): U+00FC
  - Unicode (combining): U+0075, U+0308
  - Unicode, UTF-8 (s.u.): U+00FC = 0xC3, 0xBC



- Basiszeichen
  - Unser normales Verständnis eines Zeichens
- 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)



# Unicode: Codierungen



- UCS-4:
  - Die allgemeine 4-Byte-Angabe: `U+dddddddd`
- UTF-8, UTF-16, UTF-32
- Unterscheidung im Fall UTF-16:
  - *high-endian* vs. *low-endian* mittels Sonderzeichen `xFEFF`
- UTF-8 Codierung:
  - `U+00000000` – `U+0000007F` `0xxxxxxx`
  - `U+00000080` – `U+000007FF` `110xxxxx 10xxxxxx`
  - `U+00000800` – `U+0000FFFF` `1110xxxx 10xxxxxx 10xxxxxx`
  - `U+00010000` – `U+001FFFFF` `11110xxx (10xxxxxx)3`
  - `U+00200000` – `U+03FFFFFF` `111110xx (10xxxxxx)4`
  - `U+04000000` – `U+7FFFFFFF` `1111110x (10xxxxxx)5`
  - 1 bis 6 Oktets pro Unicode-Zeichen (31 bits), niemals `xFE` oder `xFF`.
  - Stets klar, ob Folgebyte vorliegt und wieviele Folgebytes notwendig!



# Zulässige Zeichen in XML allgemein



- XML 1.0:
 

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF]
           | [#xE000-#xFFFF] | [#x10000-#x10FFFF]
           /* any Unicode character, excluding the
              surrogate blocks, FFFE, and FFFF. */
```

  - Beachte: NICHT alle Unicode-Zeichen!
- XML 1.1:
 

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#x7E]
           | #x85 | [#xA0-#xD7FF] |
           [#xE000-#xFFFF] | [#x10000-#x10FFFF]
           /* any Unicode character, excluding most ISO
              controls, the surrogate blocks, FFFE,
              and FFFF. */
```

  - Beachte: NICHT voll abwärtskompatibel zu XML 1.0!



#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: ...]	<i>Well-formedness or validity constraint</i>



# ...zurück zur Klärung der Regeln...



- XML 1.0:

[3] **S** ::= (#x20 | #x9 | #xD | #xA)+

- Also: Beliebige Zeichenfolgen aus *blank*, *form feed (FF)*, *carriage return (CR)* oder *line feed (LF)*

- XML 1.1:

- Vermutlich Erweiterung um #x85 (IBM: *NEL*) und #x2028 (Unicode *line separator char*)
- Bem.: CR-Phase lief am 28.2.03 aus...



- Einschränkung bei der Namenswahl:
  - XML fordert die Einhaltung bestimmter Regeln bei der Vergabe von z.B. Element- und Attributnamen.

[5] **Name** ::= (Letter | '\_' | ':') (NameChar)\*

[4] **NameChar** ::= Letter | Digit |  
'.' | '-' | '\_' | ':' |  
CombiningChar | Extender

[6] **Names** ::= Name (#x20 Name)\*  
/\* #x20: Beachte errata in [6] \*/



## XML names, name tokens



- Noch zu klären
  - Letter, Digit, CombiningChar, Extender
  - AttValue (!!)
- Dazu: Ein Blick in die Spezifikation!



## XML names, name tokens



- Achtung - Neues Konzept bei XML 1.1:

[5] **Name** ::= **NameStartChar** **NameChar**\*

[4] **NameStartChar** ::= ":" | [A-Z] | "\_" |  
[a-z] | [#xC0-#x2FF] | [#x370-#x37D] |  
[#x37F-#x1FFF] | [#x200C-#x200D] |  
[#x2070-#x218F] | [#x2C00-#x2FEF] |  
[#x3001-#xD7FF] | [#xF900-#xEFFFF]

[4a] **NameChar** ::= **NameStartChar** | '-' | '.' | [0-9] |  
#xB7 | [#x0300-#x036F] | [#x203F-  
#x2040]



## XML names, name tokens



[7] **Nmtoken** ::= (NameChar)+ /\* XML 1.0 \*/

[7] **Nmtoken** ::= NameChar+ /\* XML 1.1 \*/

[8] **Nmtokens** ::= Nmtoken (#x20 Nmtoken)\*

- Bemerkungen

- *names* fangen also immer mit einem „Buchstaben“, `_` oder Doppelpunkt an.
- *name tokens* unterliegen diesen Einschränkungen nicht.
- Namen, die mit ('x'|'X') ('m'|'M') ('l'|'L') beginnen, sind von XML **reserviert** (z.B. xml, Xml, xML, XML, ...)
- Der **Doppelpunkt** ist i.d.R. für XML-interne Zwecke reserviert - **meiden!**



## XML names: 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 my#Attr = '...'> ... </myElem>`

**#** ist kein erlaubtes Zeichen in einem *name*



## Markup: Referenzen

Zeichenreferenzen  
(*character references*)  
(*general*) *entity references*



## Character references



- Zeichenreferenzen (*char. references*):
  - Eine Methode zur Einbettung beliebiger (einzelner) Zeichen unter ausschließlicher Verwendung der ASCII-Codierung.
  - Ansatz: Angabe entweder des dezimalen oder des hexadezimalen Unicode-Wertes, eingebettet in speziellen XML *markup*:

```
[66] CharRef ::=
    '&#' [0-9]+ ';' |
    '&#x' [0-9a-fA-F]+ ';'
    [WFC: Legal Character]
```



## Character references



- Beispiel:

```
<someText>  
  &#x41; &#x0042; &#099; &#100;      /* ABcd */  
</someText>
```

- Bemerkungen

- Angaben entweder in dezimaler oder in hexadezimaler Notation – binär oder oktal sind nicht zulässig.
- Hex-Darstellung: Kleine wie große Ziffern-Buchstaben sind zulässig.
- WFC - *Well-formedness constraint*:  
Gemeint ist hier, dass das derart referenzierte Zeichen aus der Menge der in Regel [2] beschriebenen „Char“ stammt.  
Andere Zeichen führen zum Parser-Abbruch (*fatal error*)!



## Entity references



- *Entity*-Referenzen:

- Eine Methode zur Einbettung beliebiger Zeichenfolgen.
- Expansion durch Parser, analog zu Makros
- Kaskadierbar (aber ohne Rekursion)

```
[68] EntityRef ::= '&' Name ';' ;  
      [WFC: Legal Character]  
      [VC: Entity Declared]  
      [WFC: Parsed Entity]  
      [WFC: No Recursion]
```

- Beispiel:

```
<par>Dieser Text entstand am &heute; im  
Auftrag der Firma &Kunde;.</par>
```



## 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? Insbesondere gilt das für: **<**, **>**, **&**, **'**, und **"**
- Die simple Lösung:
  - Codierung über *character references*.
- Die elegantere Lösung:
  - Zugriff über symbolische Namen in „*entity references*“.



## Vordefinierte *general entities*



- XML kennt 4 bereits vordefinierte entities:  
**amp**, **lt**, **gt**, **apos**, **quot**
- Verwendung per Referenz:

<b>&amp;amp;</b>	&
<b>&amp;lt;</b> und <b>&amp;gt;</b>	< und >
<b>&amp;apos;</b> und <b>&amp;quot;</b>	' und "
- Beispiel: **A < B & C > B**  
**<myTag>**  
**A &lt; B &amp; C &gt; B**  
**</myTag>**



- Wunsch:
  - Ganze Symboltabellen, also Listen gängiger Unicode-Spezialzeichen, per *char. ref.* spezifiziert), deren Einträge einfach per *entity reference* verwendbar sind.
  - Möglichkeit, eigene *entities* wie „heute“ oder „Kunde“ einzuführen.
- Dazu notwendig:
  - Regeln zur Deklaration von *entities*
  - Methoden zur Einbindung externer Dateien
- **Nächstes Kapitel!**