

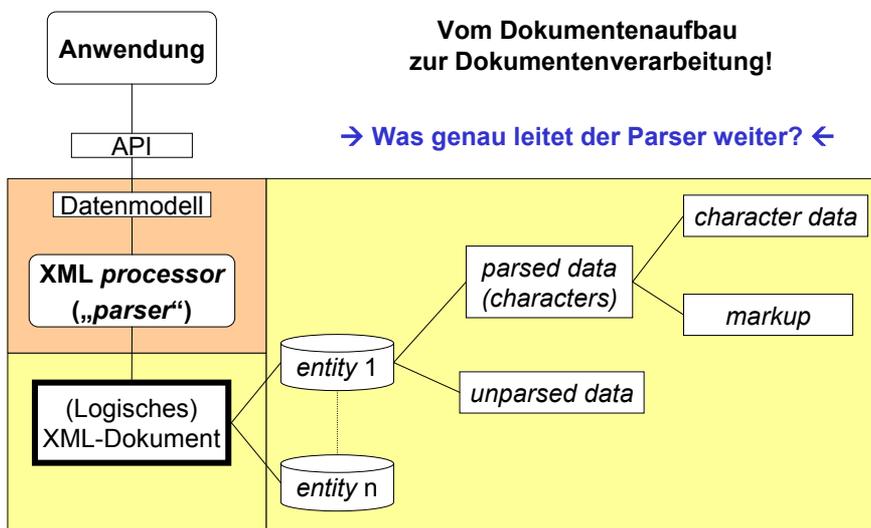


Attribut-Konventionen und Normierungsregeln

xml:lang, xml:space
Zeilenenden, Attributwerte



Perspektivwechsel





Attribut-Konventionen - xml:lang



- Zur Kennzeichnung der Sprache eines Elementinhalts sollte folgendes Standardverfahren eingehalten werden. Beispiel:

```
<p xml:lang="en">The quick brown fox jumps over the
  lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
</sp>
```

- Zur Konvention, die das reservierte Attribut „xml:lang“ verwendet, gehört ein Vererbungskonzept: Sprachschlüsselwerte vererben sich implizit vom definierenden Element auf alle Kind-Elemente, bis dass sie von einem lokalen Wert überschrieben werden.



Attribut-Konventionen - xml:lang



- „Gültige“ XML-Dokumente müssen das reservierte Attribut **xml:lang** in allen Elementen, die es verwenden, normal deklarieren.
- „**xml:lang**“ lässt zwar nur bestimmte Sprach- bzw. Länder-Codes zu (Details: Siehe XML 1.0, Kap. 2.12), aber diese müssen nicht explizit - etwa per NOTATION – deklariert werden, sondern verstehen sich als Teil der Konvention.
- Beispiel für eine einfache Deklaration:

```
xml:lang CDATA #IMPLIED
```

- Vollständiges Beispiel:

```
<!ATTLIST poem xml:lang CDATA 'fr'>
<!ATTLIST gloss xml:lang CDATA 'en'>
<!ATTLIST note xml:lang CDATA 'en'>
```

- Im Beispiel wird ein Gedicht in französischer Sprache unterstellt, das Ergänzungen/Anmerkungen in Englisch erhalten soll. Daher verwendet die DTD kontextabhängig *default*-Werte für **xml:lang**.



- Das Problem:
 - Grundsätzlich müssen XML-Prozessoren alle Zeichen, die nicht *markup* sind, an die Anwendung weiterreichen.
 - Validierende *Parser* müssen darüberhinaus der Anwendung mitteilen, welche Zeichen eines Elementinhalts *white space* sind (!).
 - Andererseits verwenden XML-Autoren gerne *white space* nur zur besseren Lesbarkeit der XML-Rohdaten.
 - **Wie kann man nun „significant white space“ (z.B. in Gedichten) von diesem unterscheiden?**
- Der Lösungsansatz:
 - Per Konvention wird das Attribut **xml:space** zu dieser Unterscheidung eingeführt. Es kann nur die Werte „*default*“ und „*preserve*“ annehmen.
 - Dieses Attribut ist ein Signal an die Anwendung - nicht an den *Parser*!
 - Für **xml:space** gilt eine Vererbungsregel analog zu **xml:lang**.



- Deklaration
 - „Gültige“ XML-Dokumente müssen das reservierte Attribut **xml:space** in allen Elementen, die es verwenden, normal deklarieren.
- Beispiele:

```
<!ATTLIST poem xml:space  
          (default|preserve) 'preserve'>
```

```
<!ATTLIST pre xml:space  
          (preserve) #FIXED 'preserve'>
```



Normierungsregeln

... für Zeilenenden

... für Attributwerte



Normierungsregeln für Zeilenenden



- Situation:
 - Verschiedene Betriebssysteme verwenden unterschiedliche Konventionen zur Kennzeichnung von Zeilenumbrüchen (DOS/Windows: (#xD, #xA); Unix: nur #xA, MacOS 9, OS/9: nur #xD).
- Ziel
 - XML-basierte Anwendungen sollen unabhängig von der Herkunft der XML-Dokumente arbeiten können.
- Lösung:
 - XML 1.0 legt fest, dass XML-Prozessoren alle *external parsed entities* noch vor Beginn des *parsing* wie folgt normiert:
 - Jede 2-Zeichen Sequenz #xD #xA wird umgesetzt in ein #xA
 - Jedes #xD, das nicht direkt von einem #xA gefolgt wird, wird ersetzt durch ein #xA.
 - D.h. beim Einlesen werden alle Zeilenenden auf den Unix-Standard normiert.



- Situation:
 - *White space* in Attributwerten kann
 - Bedeutungsträger sein (CDATA),
 - zur *Token*-Bildung benötigt werden (bei Listentypen wie NMTOKENS),
 - nur der besseren Lesbarkeit halber eingegeben worden sein,
 - schlicht die Anwendung stören, zu vielfältig oder gar ungültig sein.
- Ziele:
 - Die Prüfung auf Einhaltung der deklarierten Attributtypen soll möglichst einfach erfolgen können.
 - Die Anwendung soll nur „signifikante“ Leerzeichen erhalten.
- Lösungsansatz:
 - *White space* in Attributwerten wird vor der Validierung und Weiterleitung an die Anwendung vom XML Prozessor kontextabhängig normiert.



- **Der Normierungsalgorithmus für Attributwerte:**
 - 1) Zeilenenden werden als bereits normiert vorausgesetzt (s.o.).
 - 2) Die Normierung eines Attributwerts beginnt mit einem leeren String
 - 3) Für jedes Zeichen, *entity reference*, oder *char reference* im noch nicht normierten Attributwert:
 - a) Falls *char reference*:
Leite das Ersetzungszeichen weiter (expandiere).
 - b) Falls *entity reference*:
Expandiere, wende Regel (3) rekursiv auf den Ersetzungstext an.
 - c) Falls *white space*-Zeichen:
Leite ein *space char* (Leerzeichen, #x20) weiter.
 - d) Falls ein anderes Zeichen:
Leite das Zeichen unverändert weiter.
- Anschließend - falls das Attribut nicht vom Typ CDATA ist:
 - Entferne führende und endende Leerzeichen (*leading & trailing blanks*).
 - Ersetze Leerzeichenketten durch einzelne Leerzeichen.



Normierungsregeln: Attributwerte



- Beispiele (vgl. Kap. 3.3.3 der XML 1.0-Spezifikationen)
 - Vorbereitende *entity*-Deklarationen:


```
<!ENTITY d "&#xD;">
<!ENTITY a "&#xA;">
<!ENTITY da "&#xD;&#xA;">
```
 - Attributspezifikation 1: `att="`

```

x y z"
– Normiert, NMTOKENS: x y z
– Normiert, CDATA: #xD #x20 x y z
```
 - Attributspezifikation 2: `att="&d;&d;A&a; &a;B&da"`

```

– Normiert, NMTOKENS: A #x20 B
– Normiert, CDATA: #x20 #x20 A #x20 #x20 #x20 B #x20 #x20
```
 - Attributspezifikation 3: `att=`

```

"&#xD;&#xD;A&#xA;&#xA;B&#xD;&#xA;"
– Normiert, NMTOKENS: #xD #xD A #xA #xA B #xD #xA
– Normiert, CDATA: #xD #xD A #xA #xA B #xD #xA
```



Normierungsregeln: Attributwerte



- Bemerkungen zu Fall 1 ...:
 - Die beiden Leerzeilen zu Beginn des Attributwertes wurden gemäß (3c) in `#x20`-Zeichen umgewandelt.
 - Da diese zu Beginn stehen, wurden sie im Fall NMTOKENS schließlich ganz entfernt
- ... zu Fall 2 ...:
 - Die *entity references* auf *char references* von *white space* wurden rekursiv aufgelöst: (3b), (3a), (3c) und in `#x20`-Zeichen umgewandelt.
 - Im Fall NMTOKENS wurden die zu Beginn und am Ende ganz entfernt und die drei mittleren zu einem zusammengefasst.
- ... und zu Fall 3:
 - Ersatzzeichen der *char references* wurden gemäß (3a) weitergereicht.
 - Da diese *white space*, aber keine Leerzeichen sind, griff die letzte Regel nicht.
 - Anmerkungen aus den Spezifikationen:
Fall NMTOKENS ist wohlgeformt, aber nicht gültig!



Abschließendes kleines Beispiel eines XML-Dokuments mit DTD



Einleitendes Beispiel-Dokument



```
<Dozent>  
  <Name>  
    <Vorname MI= 'W'>Heinz</Vorname>  
    <Nachname Titel="Dr">Werntges</Nachname>  
  </Name>  
  <Beschäftigungsverhältnis Art="Prof"/>  
</Dozent>
```

- Aufgabe:
 - Nun zu vervollständigen um Prolog und insbesondere um eine DTD. Ziel: Validierung ermöglichen.



```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE Dozent [
  <!ELEMENT Dozent (Name, Beschäftigungsverhältnis)>
  <!ELEMENT Name (Vorname, Nachname)>
  <!ELEMENT Vorname (#PCDATA)>
  <!ELEMENT Nachname (#PCDATA)>
  <!ELEMENT Beschäftigungsverhältnis EMPTY>
  <!ATTLIST Beschäftigungsverhältnis
    Art (LB | Prof | Stud) #REQUIRED>
  <!ATTLIST Nachname Titel NMTOKEN #IMPLIED>
  <!ATTLIST Vorname MI
(A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|Ä|Ö|Ü) # IMPLIED>
]> <!-- Hier endet der hinzugefügte Prolog -->
<Dozent>
  <Name>
    <Vorname MI= 'W'>Heinz</Vorname>
    <Nachname Titel="Dr">Werntges</Nachname>
  </Name>
  <Beschäftigungsverhältnis Art="Prof"/>
</Dozent>
```



- Errata zu 1.0
 - Die 2004-Ausgabe von XML 1.0 enthält bereits alle früheren Errata.
 - Neue „Errata“ sollten ggf. herangezogen werden, wenn man einen Fehler in den Spezifikationen vermutet sich ein Parserverhalten partout nicht erklären kann.
- XML 1.1
 - Alle Errata aus 1.0 sind berücksichtigt
 - Berücksichtigt Änderungen zwischen Unicode 2.0 und 4.0
 - Flexibler in den Regeln zu *Name(s)* und *Nmtoken(s)*
 - Änderungen bei Steuerzeichen (neue kommen hinzu), aber auch:
 - Nicht abwärtskompatibel bez. Zeichen #x7F - #x9F (**umstritten!**)
 - Normierung der Zeilenenden verallgemeinert: #x85, #x2028 neu



Anhang

Markup-Übersicht XML *literals*



Markup: Übersicht nach Erscheinungsbild



„Nicht-essenzielle“ *markup*-Arten:

`<!-- ... -->`

Kommentar

`<?...?>`

XML-Deklaration, Text-Deklaration, PI (*processing instruction*)

`<![NAME[...]]>`

Besondere Code-Abschnitte; $NAME \in \{CDATA, INCLUDE, IGNORE\}$

Kern der XML-*Markups*:

`<name attr="value">...</name>, <name attr='value'/>`

Elemente und Attribute

`<!DOCTYPE docname ... [...] >`

Dokumententyp-Deklaration (hier: *root element* heißt „docname“)

`<!NAME ... >`

Markup-Deklarationen. Einzelfälle:

$NAME \in \{ENTITY, ELEMENT, ATTLIST, NOTATION\}$

```
[9] EntityValue ::= '"' ([^%&"] | PReference | Reference)* '"' |
    "'" ([^%&' ] | PReference | 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 *identifizier* (*SystemLiteral*, *PubidLiteral*)
- Man beachte die Doppel-Beschreibung für die beiden *quotation marks*!

- 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 verwendeten *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 - daher Sonderbehandlung!