



XML 1.0 - Die Spezifikation

Schrittweise Erarbeitung
Kommentare und Beispiele



- 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.



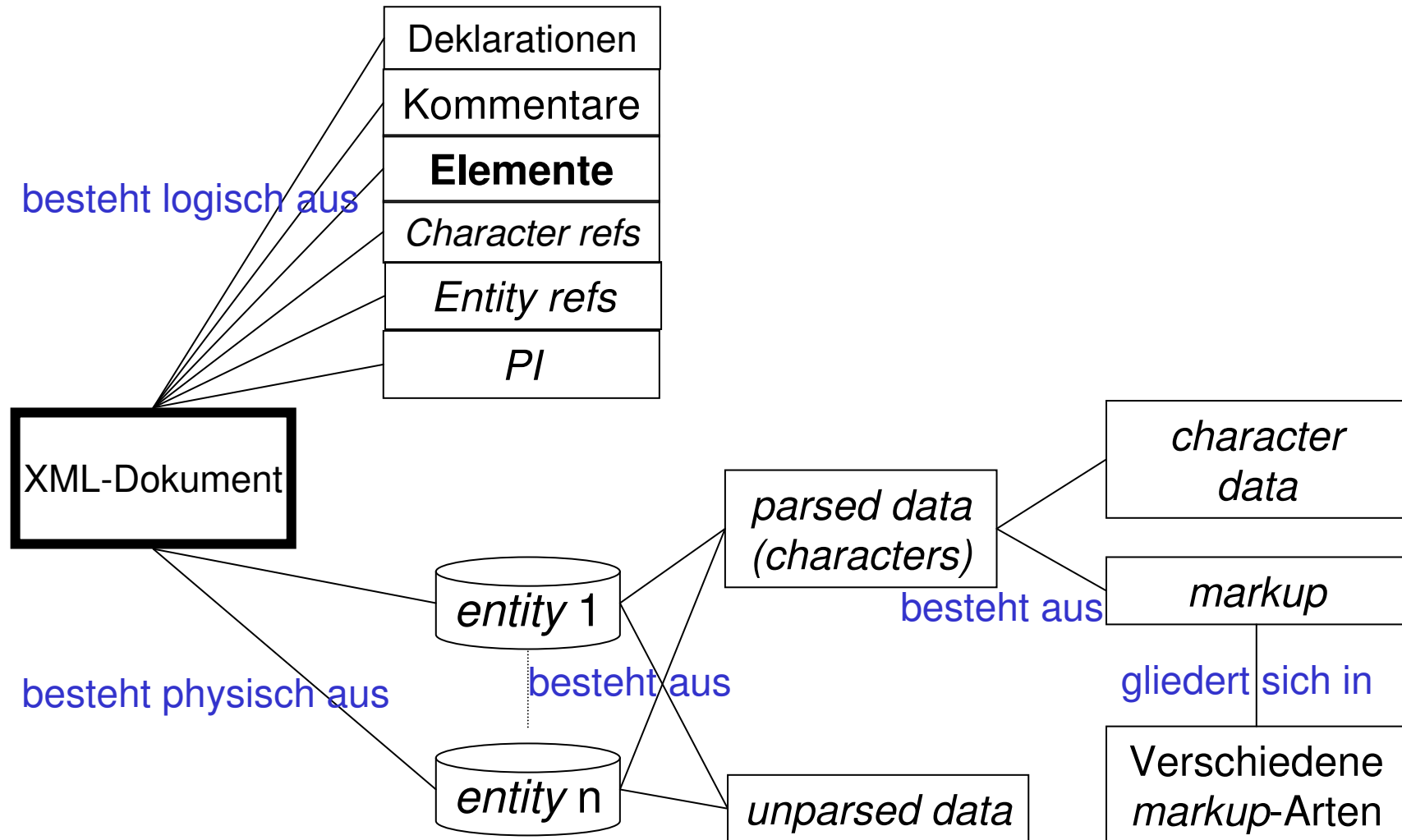
- Einschub: Zur Methodik
 - Die nächsten Abschnitte gehen **deduktiv** vor, denn
 - wir benötigen erst einmal ein formales Rüstzeug, um die Bestandteile von XML präzise beschreiben zu können.
 - Damit wird dann klar werden, was genau in XML erlaubt ist.
 - Zahlreiche Parserfehler liegen in Verletzungen 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 deklariertes DTD
- XML-Dokument
 - Physischer Aufbau:
Entities
 - Logischer Aufbau (Bestandteile):
Deklarationen
Elemente
Kommentare
character references
processing instructions (PI)



XML-Dokumente: Aufbau





- 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!



Grundlagen und erste Regeln der XML 1.0-Spezifikation

Tags und Referenzen

Unicode und zulässige Zeichen in XML

Beispiele:

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

2. `<#Beispiel>`

`Hallo zusammen!`

`</#Beispiel>`

3. `<Beispiel Sprachschlüssel="DE">`

`Hallo 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] **S**Tag ::= '<' Name (S Attribute)* S? '>'

[41] **A**tttribute ::= Name Eq AttValue

[42] **E**Tag ::= '</' Name S? '>'

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

[25] **E**q ::= 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/> (war mal offline), Ersatz+mehr:
<http://www.i18nguy.com/unicode/codepages.html>
zu Zeichensätzen allgemein
- <http://www.unicode.org/>
Speziell zu Unicode

- Beispiel: Buchstabe

- | | |
|--------------------------|---------------------|
| – Codepage 437 (DOS): | „ü“
0x81 |
| – ISO-8859-1: | 0xFC |
| – Unicode (composite): | U+00FC |
| – Unicode (combining): | U+0075, U+0308 |
| – Unicode, UTF-8 (s.u.): | U+00FC = 0xC3, 0xBC |



Unicode: Zeichenarten



- 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:

<code>U+00000000</code>	–	<code>U+0000007F</code>	<code>0xxxxxxx</code>	
<code>U+00000080</code>	–	<code>U+000007FF</code>	<code>110xxxxx</code>	<code>10xxxxxx</code>
<code>U+00000800</code>	–	<code>U+0000FFFF</code>	<code>1110xxxx</code>	<code>10xxxxxx</code> <code>10xxxxxx</code>
<code>U+00010000</code>	–	<code>U+001FFFFF</code>	<code>11110xxx</code>	<code>(10xxxxxx)</code> ₃
<code>U+00200000</code>	–	<code>U+03FFFFFF</code>	<code>111110xx</code>	<code>(10xxxxxx)</code> ₄
<code>U+04000000</code>	–	<code>U+7FFFFFFF</code>	<code>1111110x</code>	<code>(10xxxxxx)</code> ₅

Folgebyte

Anzahl Folgebytes

- 1 bis 6 Oktetts pro Unicode-Zeichen (31 bits), niemals `xFE` oder `xFF`.
- Stets klar, ob Folgebyte vorliegt und wie viele Folgebytes notwendig!



Zulässige Zeichen in XML allgemein



- **XML 1.0:**

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF]
          | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

/ any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. */*

- Beachte: NICHT alle Unicode-Zeichen!

- **XML 1.1:**

```
[2] Char ::= [#x1-#xD7FF] | [#xE000-#xFFFD] |
           [#x10000-#x10FFFF]
```

```
[2a] RestrictedChar ::= [#x1-#x8] | [#xB-#xC] |
                        [#xE-#x1F] | [#x7F-#x84] | [#x86-#9F]
```

- RestrictedChar: Nur als Zeichenreferenz zulässig.
- Weitere Unicode-Zeichen vermeiden; siehe Spec. (2.2)
- Beachte: NICHT voll abwärtskompatibel zu XML 1.0!



Notationen – zum Nachlesen...



#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:**

- Weitere „Zeilenende“-Zeichen: #x85 (IBM: NEL) und #x2028 (Unicode *line separator char*)
- Regel [3] blieb aber unverändert!



XML names, name tokens



- **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-#xD6] | [#xD8-#xF6] |
[#xF8-#x2FF] | [#x370-#x37D] |
[#x37F-#x1FFF] | ...
(weitere 7 Intervalle)

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



XML names, name tokens



[7] **Nmtoken** ::= (NameChar) +

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

- Bemerkungen

- *names* fangen also immer mit einem „Buchstaben“ (im allgemeinen Unicode-Sinn), mit `_` 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!**
- *Best practice*-„Vorschläge“ für XML *names* etc.: **Anhang I**



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 Zeichenreferenzen (*character references*)
- Die elegantere Lösung:
 - Zugriff über symbolische Namen in „*entity references*“.



Vordefinierte *general entities*



- XML kennt 5 bereits vordefinierte entities:
amp, lt, gt, apos, quot

- Verwendung per Referenz:

<i>&amp;</i>	&
<i>&lt;</i> und <i>&gt;</i>	< und >
<i>&apos;</i> und <i>&quot;</i>	' und ''

- Beispiel: $A < B \& C > B$

<someMath>

A < B & amp; C > B

</someMath>



- Weiterführender Wunsch:
 - Benutzung vorbereiteter Symboltabellen (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:
 - Methoden zur Einbindung externer Dateien
 - Regeln zur Deklaration von *entities*

→ Nächstes Kapitel!



Pragmatischer Vorgriff 1:

Entity-Deklarationen,
intern & extern

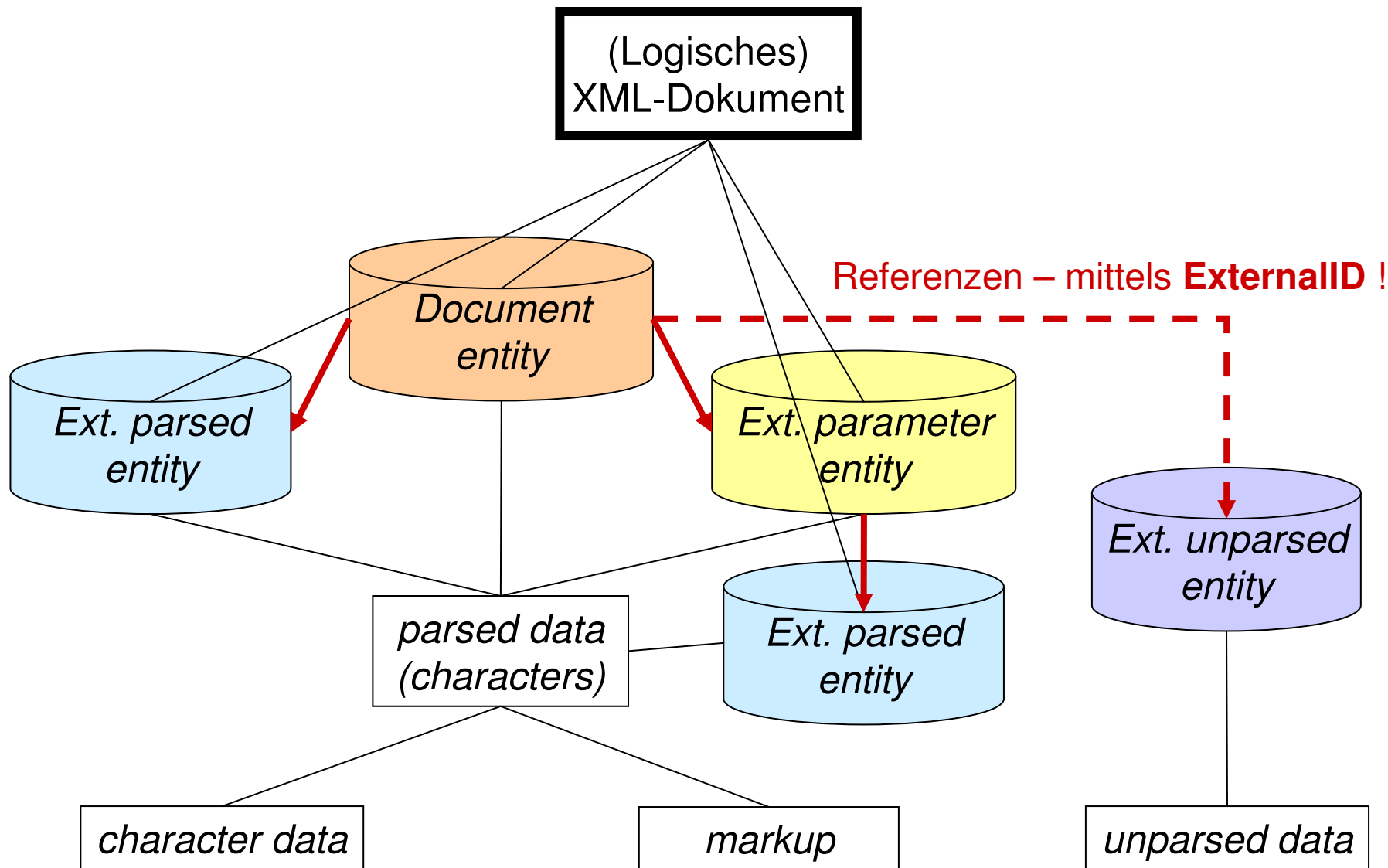


- **Warum ein Vorgriff?**
 - Erste Eindrücke vom Ziel, noch vor dem Theorie-Teil
 - Vorbereitung für das Praktikum

- **Wie erfolgt der Vorgriff?**
 - Durch Beispiel
 - ... und einige wenige Produktionsregeln
 - Auf der Basis stichwortartiger Folien
 - Mit ausführlichen mündlichen Erläuterungen



Verweise auf externe *entities*





Beispiel 1: Mit interner DTD

```
<?xml version="1.0"?>  
<!DOCTYPE greeting [  
  <!ELEMENT greeting (#PCDATA)>  
>  
<greeting>Hello, world!</greeting>
```

Beispiel 2: Mit externer DTD

```
<?xml version="1.0"?>  
  <!DOCTYPE greeting SYSTEM "hello.dtd">  
<greeting>Hello, world!</greeting>
```



```
<?xml version='1.0'?>
<!DOCTYPE Test [
<!ELEMENT Test (par+) >
<!ELEMENT par (#PCDATA) >
<!ENTITY heute '02.04.2008'>
<!ENTITY Kunde 'XML Inc.'>
]>
<Test>
  <par>
    Dieser Text entstand am &heute; im Auftrag
    der Firma &Kunde;.
  </par>
  <par>
    Noch ein Beispielsatz, erstellt am &heute; .
  </par>
</Test>
```



Entity-Deklarationen: Externer Fall



```
<?xml version='1.0'?>
<!DOCTYPE Test [
<!ELEMENT Test (par+) >
<!ELEMENT par (#PCDATA) >
<!-- Dynamische Erzeugung von Inhalten: -->
<!ENTITY heute SYSTEM
        "http://ww.mydomain.xy/cgi-bin/get_date.cgi">
<!ENTITY Kunde 'XML Inc.'>
]>
<Test>
  <par>
    Dieser Text entstand am &heute; im Auftrag
    der Firma &Kunde;.
  </par>
</Test>
```



Parameter Entity-Deklaration



```
<?xml version='1.0'?>
<!DOCTYPE Test [
<!ELEMENT Test (par+) >
<!ELEMENT par (#PCDATA) >
<!-- Externes Parameter-Entity -->
<!ENTITY % myExtDecls SYSTEM "boilerplate.ent">
<!-- Expansion z.B. zu Entity-Deklarationen: -->
%myExtDecls;
]>
<Test>
  <par>
    Dieser Text entstand am &heute; im Auftrag
    der Firma &Kunde;.
  </par>
</Test>
```



Entity-Deklarationen: Nützlich!



Gliederung und Modularisierung mit externen *entities*:

```
<?xml version='1.0'?>
<!DOCTYPE mythesis SYSTEM "mythesis.dtd" [
<!ENTITY ch01 SYSTEM 'chapter01.ent'>
<!ENTITY ch02 SYSTEM 'chapter02.ent'>
<!ENTITY ch03 SYSTEM 'chapter03.ent'>
]>
<!-- Wrapper document -->
<mythesis>
  &ch01; <!-- Put content of external entity here -->
  &ch02; <!-- By putting some chapters in comments, -->
  &ch03; <!-- we can develop long docs in parts -->
</mythesis>
```




Pragmatischer Vorgriff 2:

XML-Deklaration,
Zeichensätze / *encoding*



```
[23] XMLDecl ::=
    '<?xml' VersionInfo EncodingDecl? SDDDecl? S? '?>'

[24] VersionInfo ::= S 'version' Eq
    ('"' VersionNum '"' | "'" VersionNum "'")

[26] VersionNum ::= '1.0' /* vgl. errata */
```

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"
    standalone="no"?>
```

- Versionsangabe
 - Muß ggf. angegeben werden
 - „1.0“ oder „1.1“ zulässig.



- Regel für die *encoding*- Deklaration:

```
[80] EncodingDecl ::= S 'encoding' Eq  
      ('"' EncName '"') | ("'" EncName "'")
```

```
[81] EncName ::= /* Encoding name contains */  
      [A-Za-z] ([a-zA-Z0-9_.] | '-')+  
      /* only Latin characters */
```

Bemerkungen

- I.d.R. die bei der IANA-CHARSETS registrierten Namen
- Namen proprietärer *charsets* mit Präfix „x-“ angeben.
- Standardwerte für die gängigen Unicode-Darstellungen:
 „**UTF-8**“, „**UTF-16**“,
 „**ISO-10646-UCS-2**“ und „**ISO-10646-UCS-4**“



- **UTF-8** und **UTF-16** muss jeder XML-Prozessor unterstützen.
- **#xFEFF** („*encoding signature*“)
 - leitet eine UTF-16 codierte Datei ein. Dieses Zeichen („*non-breakable zero-length space*“, „*byte order mark*“) zählt dann weder zum *markup* noch zu den *char data*, sondern steuert die Erkennung der Codierung (UTF-16) sowie die der Byte-Reihenfolge (*little-endian vs. big-endian processors*).
- XML-Prozessoren sollen die *encoding*-Werte unabhängig von Klein-/Großschrift erkennen.
- Weitere gängige *encoding*-Werte:
 - **ISO-8859-*n*** (*n*=1, 2, ..., 9; 15)
 - **ISO-2022-JP, Shift-JIS, EUC-JP**
 - **Windows-1252** (ISO-8859-1 Obermenge), **Windows-125*n*** (*n*=0...8)
- Hintergrundinformationen zu Zeichensätzen:
 - Siehe Vorübung und „Einschub: Unicode“



- Regel für die *standalone* Dokumentdeklaration:

```
[32] SDDecl ::= S 'standalone' Eq  
              (("'" ('yes' | 'no') "'") |  
               ('"' ('yes' | 'no') '"'))
```

- Bemerkungen
 - Zulässige Werte sind nur “yes“ und “no“, *default* ist “no“
 - Der Wert “yes“ bedeutet, dass das XML-Dokument keine externen *markup*-Deklarationen aufweist, die die vom Parser an die Anwendung geleiteten Informationen betreffen.
 - Externe Attribute mit *default*-Werten würden z.B. “no“ erfordern.