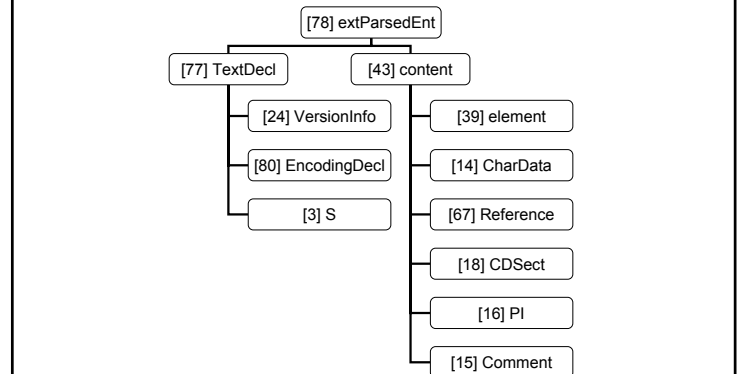
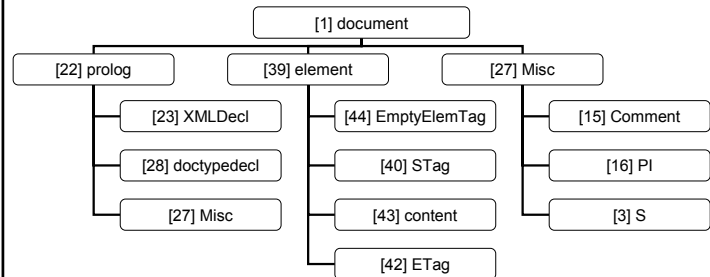
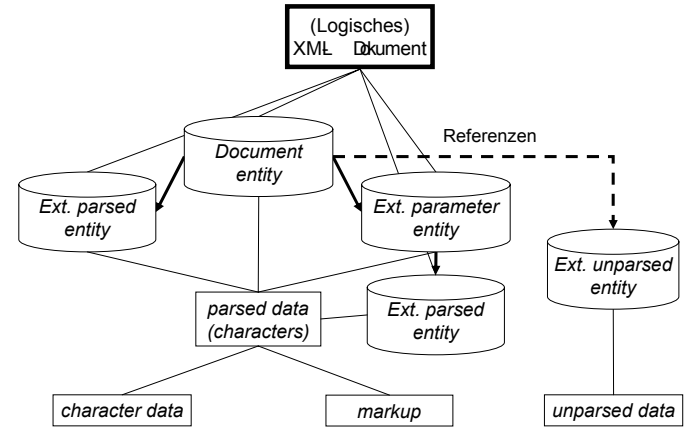


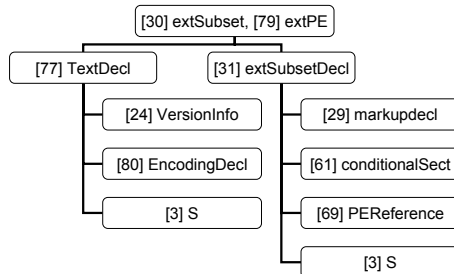
XML 1.0 - Die Spezifikation

Formaler Aufbau eines XML-Dokuments

Der Prolog, incl. DTD
 Das *root*-Element; Unter-Elemente
 Markup am „Ende“



Ebene 1: external subset / parameter entity



Dokumenten-Grobstruktur

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Vorstellung mittels XML-Text -->
<?xml-stylesheet href="demo.css" type="text/css"?>
<!DOCTYPE Dozent SYSTEM "demo.dtd" [
<ENTITY % ISO-Latin1-chars SYSTEM "iso8859-1.ent">
%ISO-Latin1-chars;
<ENTITY Abk "Besch&auml;ftigungsverh&auml;ltnis">
]>
<Dozent>
  <Name>
    <Vorname MI='W'>Heinz</Vorname>
    <Nachname Titel="Dr">Werntges</Nachname>
  </Name>
  <&Abk; Art="Prof"/> <!-- *SYNTAXFEHLER* -->
</Dozent>
  element
<!-- Zum Abschluss noch whitespace und eine PI: -->

<?someApp param1 key2="someValue" moreParams ?>
  Misc
  
```

Formaler Aufbau eines XML-Dokuments

- Regel:


```
[1] document ::= prolog element Misc*
```
- Beispiel:

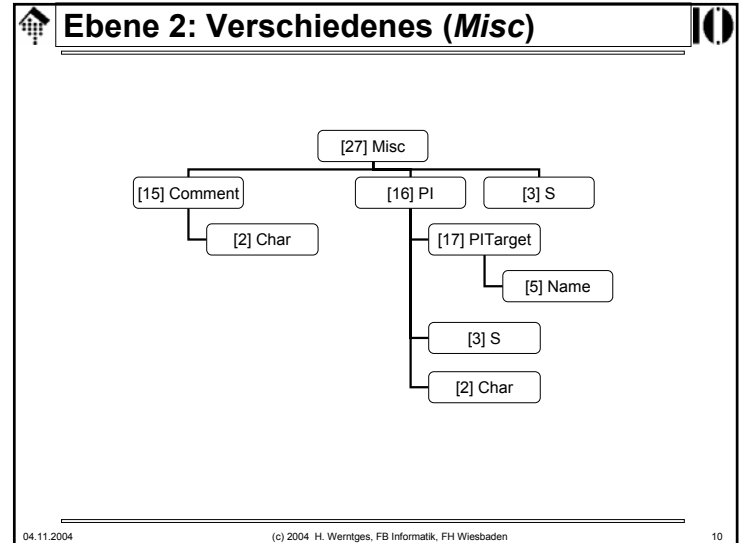
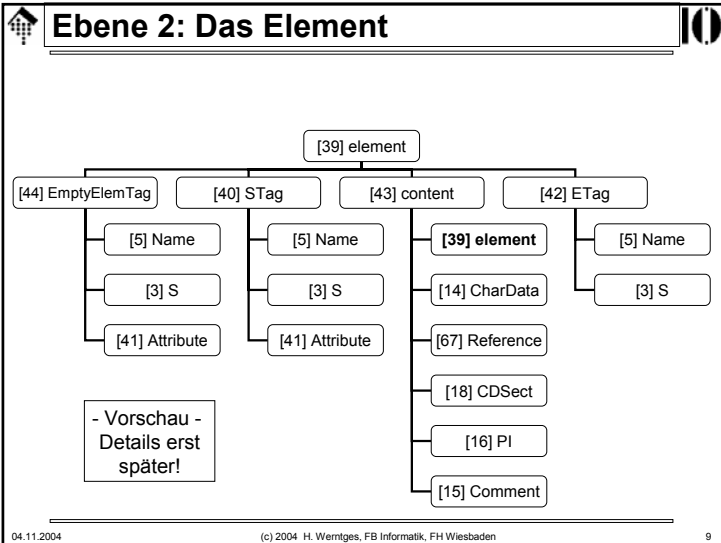

```
<hello>Hello, world!</hello>
```

 (bereits ein (wohlgeformtes) XML-Dokument)
- Bemerkungen:
 - Ein XML-Dokument besteht aus dem Prolog, einem Element, und verschiedenen „Anhängen“.
 - Das Element heißt *document element* oder *root element*.
 - Es besteht wiederum aus Unter-Elementen u.a. *markup*.
Typische Begriffsbildung: *parent elements*, *child elements*
 - Definitionen von *prolog*, *element*, *Misc* erfolgen noch
 - Vorab: Der Prolog darf auch leer sein bzw. fehlen.

Formaler Aufbau eines XML-Dokuments

- Regelvariante in XML 1.1:


```
[1] document ::= prolog element Misc*
      - Char* RestrictedChar Char*
```
- Bemerkungen:
 - Der „Zusatz“ in XML 1.1 schließt die direkte Verwendung eines *RestrictedChar* im Dokument aus.
 - Zeichenreferenzen auf *RestrictedChars* sind dagegen zulässig.



„Verschiedenes“

- Regel:
[27] Misc ::= Comment | PI | S
- in Worten:
 - Kommentare,
 - *processing instructions* ("Verarbeitungsanweisungen"),
 - oder *white space* (bereits behandelt)

04.11.2004 (c) 2004 H. Wernigtes, FB Informatik, FH Wiesbaden 11

Kommentare

- Regel:
[15] Comment ::= '

Kommentare

- Bemerkungen:
 - Kommentare sehen ähnlich aus wie in HTML, unterliegen aber strengeren Regeln.
 - Sie sind nicht schachtelbar.
 - Sie zählen als *markup* und dürfen außerhalb anderen *markups* erscheinen.
 - Innerhalb von Kommentaren wird *markup* überlesen.
 - XML Prozessoren dürfen Kommentare an Anwendungen durchreichen - sie müssen dies aber nicht!

Kommentare

- ACHTUNG:
 - Innerhalb Kommentarinhalten ist '--' nicht zulässig!
 - Das letzte Zeichen des Kommentarinhalts darf kein '-' sein!
 - Kommentare sollen nicht zur Steuerung von Anwendungen missbraucht werden!

XML stellt für solche Zwecke die PI `parat` – s.u.

Es besteht bewusst keine Garantie, dass eine Anwendung die Kommentare auch „durchgereicht“ bekommt!

Processing instructions (PI)

- Zweck:
 - Ein Standard zur Steuerung von Anwendungen.

- Regeln:

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

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

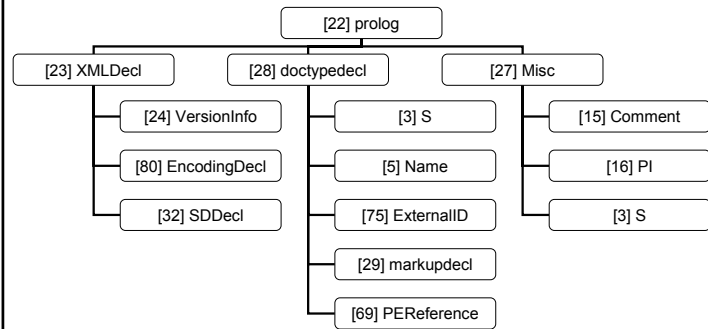
- Beispiel:

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

Processing instructions (PI)

- Bemerkungen:
 - Proprietäre Lösungen z.B. durch Konventionen innerhalb von Kommentaren sollen so vermieden werden. Beispiel Apache, SSI. Dennoch gilt:
 - PI-Einsatz sollte man auf das Nötigste beschränken!
 - Pis müssen vom Parser an die Anwendung durchgereicht werden.
 - *Parameter entity references* werden innerhalb von PIs nicht expandiert.
 - PITarget identifiziert die Anwendung (siehe auch *notations*).
 - Nicht mit der XML Deklaration verwechseln (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!

Ebene 2: Prolog



Ebene 2: Prolog

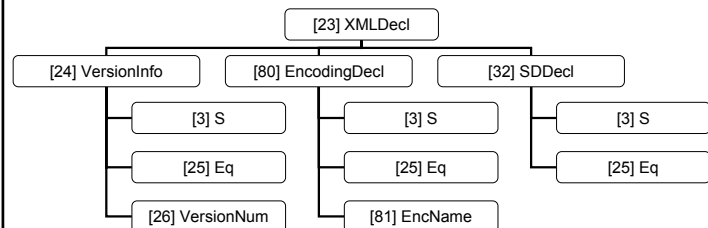
- Regel:
`[22] prolog ::= XMLDecl? Misc* (doctypeDecl Misc*)?`
- Beispiel:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- PI zur Stylesheet-Einbindung -->
<?xml-stylesheet href="mystyle.xml" rel="text/xsl"?>
<!DOCTYPE mydoc SYSTEM "mydoc.dtd" <!-- DTD -->
<mydoc>...</mydoc> <!-- Kein Prolog mehr -->

```
- Bemerkungen:
 - Die XML Deklaration muß ggf. den Dateianfang bilden.
 - Sowohl die XML Deklaration als auch die Dokumenttyp-Deklaration dürfen fehlen.
 - Kommentare, *processing instructions* und *white space* dürfen „eingestreut“ werden.

Ebene 3: XML-Deklaration



Ebene 3: XML-Deklaration

- [23] XMLDecl ::= `'<?xml' VersionInfo EncodingDecl? SDDDecl? S? '?'`
- [24] VersionInfo ::= `S 'version' Eq ("'" VersionNum "' | "'" VersionNum "'')`
- [26] VersionNum ::= `'1.0' /* XML 1.0 */`
- [26] VersionNum ::= `'1.1' /* XML 1.1 */`

Beispiel:

```

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

```

- Versionsangabe
 - Muss angegeben werden

Die XML-Deklaration

- 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“

Die XML-Deklaration: Encoding

- 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-125n (*n*=0, ..., 8)
- Hintergrundinformationen zu Zeichensätzen zu finden unter:
 - <http://www.unicode.org>, <http://czyborra.com> (Ersatz: vgl. Vorübung)

Die XML-Deklaration

- Regel für die *standalone* Dokumentdeklaration:

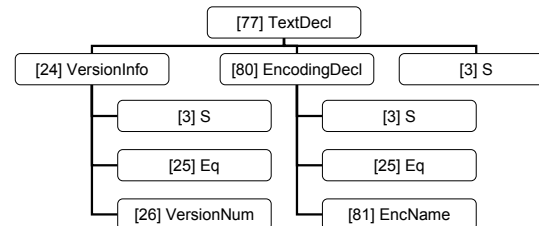
```
[32] SDDecl ::= S 'standalone' Eq
  (('"' ('yes' | 'no') '"') |
  (('"' ('yes' | 'no') '"'))
```

```
[32] SDDecl ::= #x20+ 'standalone' Eq /* XML 1.1 */
  (('"' ('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, z.B. Attributdefaults und Entity-Deklarationen.
- Externe Attribute mit *default*-Werten würden z.B. „no“ erfordern.

Ebene 2: Text-Deklaration



Die Text-Deklaration

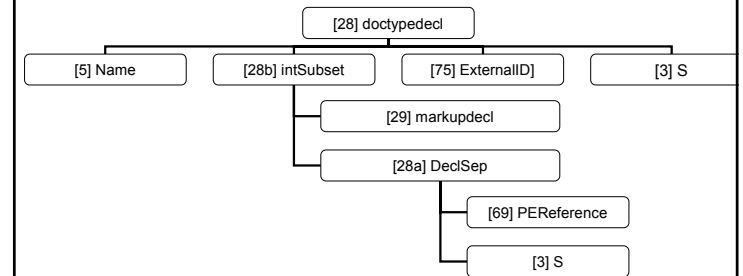
- Regel:
[77] **TextDecl** ::=
'<?xml' VersionInfo? EncodingDecl S? '?>'
- Bemerkungen
 - KEIN Prologteil - hier dennoch vorgestellt, denn:
 - Die Textdeklaration sieht sehr ähnlich aus wie die XML-Deklaration.
 - Die Versionsangabe ist hier aber optional, dafür ist die encoding-Deklaration vorgeschrieben.
 - Die Textdeklaration bildet ggf. die erste Zeile eines jeden externen entity (außer dem *document entity* selbst - dort ist die XML-Deklaration vorgeschrieben).
 - Zweck ist die Mitteilung des verwendeten Zeichensatzes bzw. seiner Codierung eines jeden *entity* an den *Parser*.
 - Der *Parser* kann jedes *entity* getrennt beim Lesen normieren, es ist also möglich, jedem entity seinen eigenen Zeichensatz zuzuordnen.

04.11.2004

(c) 2004 H. Wernigtes, FB Informatik, FH Wiesbaden

25

Ebene 3: Dokumententyp-Deklaration



04.11.2004

(c) 2004 H. Wernigtes, FB Informatik, FH Wiesbaden

28

Die Dokumententyp-Deklaration

- Regeln (beachte *errata*):
[28] **doctypeDecl** ::= '<!DOCTYPE' S Name
(S ExternalID)? S?
('[' intSubset ']' S?)? '>'

[VC: Root Element type]
[WFC: External Subset]
[28a] **DeclSep** ::= PEReference | S
[WFC: PE Between Declarations]
[28b] **intSubset** ::= (markupdecl | DeclSep)*
[29] **markupdecl** ::=
elementdecl | AttlistDecl | EntityDecl |
NotationDecl | PI | Comment
[VC: Proper Declaration / PE Nesting]
[WFC: PEs in Internal Subset]
[69] **PEReference** ::= '% Name ';' (Vorgriff)

04.11.2004

(c) 2004 H. Wernigtes, FB Informatik, FH Wiesbaden

27

Die Dokumententyp-Deklaration

Beispiel 1: Mit externer DTD

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

Beispiel 2: Mit interner DTD

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

Beispiel 3: Ohne DTD (zulässig!)

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

04.11.2004

(c) 2004 H. Wernigtes, FB Informatik, FH Wiesbaden

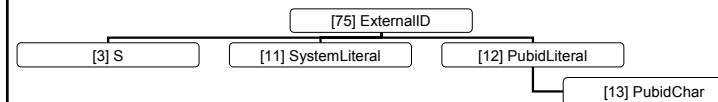
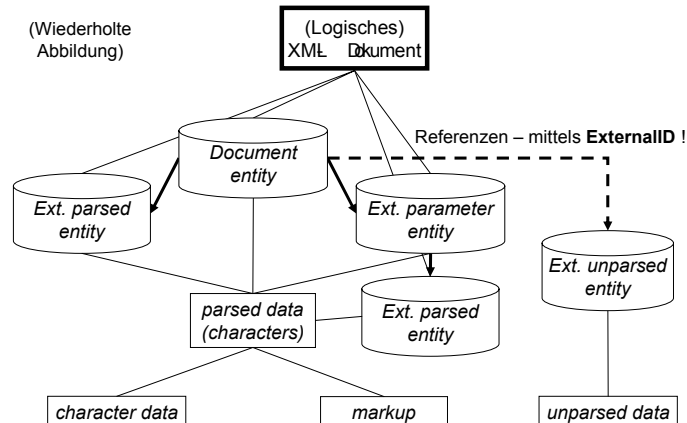
28

- Der *Name* deklariert den Typ des *document* bzw. *root elements*. Diese Deklaration stellt somit den Anfangspunkt der meisten folgenden *markup*-Deklarationen dar.
- Die Deklaration sieht formal ähnlich aus wie andere *markup*-Deklarationen, allerdings besitzt sie noch einen „Zusatzteil“ in eckigen Klammern ([...]), das *interne Subset*.
- Dem Namen des *document element* wird die *Document Type Definition (DTD)* (i.w. eine Liste von *markup*-Deklarationen, siehe die folgenden Abschnitte) zugewiesen. Diese DTD ist Grundlage jeder Validierung.
- Die DTD kann sowohl als externes *entity* (vgl. ExternalID) als auch intern (vgl. die Inhalte der [...] -Sektion) angegeben werden.
- Grundregel: **Interne Deklarationen haben Vorrang vor externen.**
- Eine DOCTYPE-Deklaration ganz ohne DTD-Angabe ist zulässig! Allerdings ist ein XML-Dokument ohne DTD vielleicht wohlgeformt, aber sicher nicht validierbar bzw. „gültig“.

Einschub: Verweise auf externe entities

PUBLIC vs. SYSTEM
FPI
XML-Kataloge

(Wiederholte Abbildung)



Aus HTML bekannt:

```
<!doctype html public "-//W3C//DTD HTML 4.0
  Transitional//EN">
<html>
  <head>
    ...
  </head>
</html>
```

SGML-Lesart!

Erneuter Versuch, nun nach XML-Regeln:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN" "mySysURI">
<html>
  <head>
    ...
  </head>
</html>
```

04.11.2004 (c) 2004 H. Werniges, FB Informatik, FH Wiesbaden 33

ExternalID: SYSTEM vs. PUBLIC

- XML unterscheidet zwei Arten externer Verweise
 - SYSTEM
 - URI (URL oder URN)
 - Insbesondere typisch bei Verwendung lokaler Pfadnamen
 - PUBLIC
 - Verweise auf öffentliche, i.d.R. nicht lokale *entities*.
 - Stammt bereits aus SGML, auch bekannt aus HTML.
- Die beiden Teile eines PUBLIC *identifiers*
 - Fixed Public Identifier* (FPI)
 - Global einheitlich zu verwendender Name für das gemeinte Objekt
 - System identifier*
 - Vom Parser zu verwenden, wenn er FPI nicht auflösen kann
 - In SGML nicht vorgesehen (optional?)
 - Formal wie ein SYSTEM *identifier* mit vorgelagertem FPI

04.11.2004 (c) 2004 H. Werniges, FB Informatik, FH Wiesbaden 34

ExternalID

- Formale Regeln:


```
[75] ExternalID ::=
    'SYSTEM' S SystemLiteral |
    'PUBLIC' S PubidLiteral S SystemLiteral

[83] PublicID ::= 'PUBLIC' S PubidLiteral

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

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

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


```
PUBLIC "-//OASIS//DTD DocBook V3.1//EN"
  "docbook/3.1/docbook.dtd"

SYSTEM "http://nwalsh.com/docbook/xml/1.3/db3xml.dtd"
```

04.11.2004 (c) 2004 H. Werniges, FB Informatik, FH Wiesbaden 35

FPI-Struktur, catalog

- Formal public identifiers (FPI): Aufbau
 - prefix
 - entweder ‚+‘ (registriert) oder ‚-‘ (nicht registriert)
 - Manchmal „ISOxxx“ (nur für ISO möglich)
 - Registrierung erfolgt durch die *Graphics Communication Association* (GCA, www.gca.org) - die GCA weist eine weltweit eindeutige Zeichenkette zu.
 - owner-identifier
 - Identifiziert die Person oder Organisation, die den *identifier* besitzt.
 - Der *identifier* sollte global eindeutig sein. Heute bieten sich Internet-Adressen an, mit Präfix IDN (*Internet Domain Name*).
 - text-class text-description
 - text-class*: beschreibt die Art der Textklasse. Beispiele:
 - DOCUMENT: SGML oder XML Dokument
 - DTD: DTD bzw. ein Ausschnitt davon
 - ELEMENTS, ENTITIES, NONSGML: wie der Name sagt...

04.11.2004 (c) 2004 H. Werniges, FB Informatik, FH Wiesbaden 36

FPI-Struktur, *catalog*

- **Formal public identifiers (FPI):** Aufbau, Fortsetzung
 - **text-description:**
Freiform-Beschreibung des Dokuments
 - **language**
Kennzeichnet die Sprache des Dokuments
Es wird empfohlen, ISO-Standard 2-Buchstabencodes zu verwenden
 - **display-version (eher selten verwendet)**
Unterscheidungskennzeichen für verschiedene Versionen desselben Dokuments, z.B. wenn mit unterschiedlichen Zeichensätzen dargestellt.
- **Formale Bildung:**
 - Die o.g. Bestandteile werden in dieser Reihenfolge mittels „/“ verkettet.
- **Beispiele**
-//OASIS//DTD DocBook V3.1//EN
-//W3C//DTD HTML 4.01//EN
-//W3C//DTD XHTML 1.0 Transitional//EN

04.11.2004

(c) 2004 H. Werntges, FB Informatik, FH Wiesbaden

37

FPI-Struktur, *catalog*

- **catalog, URN**
 - Unter dem *catalog* versteht man im SGML-Kontext die „Wegbeschreibung“ von der *ExternalID* zur konkreten Datei.
 - Beispiele:

```
PUBLIC "-//OASIS//DTD DocBook V3.1//EN"
      "docbook/3.1/docbook.dtd"
SYSTEM "http://nwalsh.com/docbook/xml/1.3/db3xml.dtd"
```
 - *Catalogs* können kaskadiert werden:
Ganze Projekte (mit eigener *catalog*-Datei) werden einfach per Verweis in den Hauptkatalog eingebunden. Dies verringert den Pflegeaufwand.
 - URN (Universal Resource Name) - hier nicht behandelt
Ein aktuellerer, konzeptionell mit FPI und *catalogs* verwandter Ansatz verwendet die Internet-Infrastruktur; noch nicht sehr verbreitet.
 - Quellenangaben
www.docbook.org (Online-Version), Kapitel 2.3: „Public Identifiers, System Identifiers, and Catalog Files“; RFC2141 („URN Syntax“), IETF, 1997

04.11.2004

(c) 2004 H. Werntges, FB Informatik, FH Wiesbaden

38

XML *catalog*

- **XML catalogs**
 - Neben den altbekannten SGML-Katalogen gibt es nun spezielle XML-Kataloge - natürlich in XML erstellt – und inzwischen von einigen wichtigen Werkzeugen auch unterstützt.
- **Ausgangspunkt unter Unix:**
`/etc/xml/catalog # eine XML-Datei`
- **Inhalt (Bsp., Teil 1 – Prolog und *root element* ohne Inhalt):**

```
<?xml version="1.0"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD Entity
Resolution XML Catalog V1.0//EN"
"http://www.oasis-
open.org/committees/entity/release/1.0/catalog.dtd">
<catalog xmlns=
  "urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <!-- Hier Inhalt, siehe Teil 2 -->
</catalog>
```

04.11.2004

(c) 2004 H. Werntges, FB Informatik, FH Wiesbaden

39

XML *catalog*

- **Inhalt (Teil 2, Unter-Elemente von „catalog“):**

```
<delegatePublic publicIdStartString="-//OASIS//ELEMENTS DocBook"
catalog="file:///usr/share/sgml/docbook/xml/dtd/4.2/catalog.xml"/>
<delegatePublic publicIdStartString="-//OASIS//ENTITIES DocBook"
catalog="file:///usr/share/sgml/docbook/xml/dtd/4.2/catalog.xml"/>
<delegatePublic publicIdStartString="-//OASIS//DTD DocBook XML"
catalog="file:///usr/share/sgml/docbook/xml/dtd/4.2/catalog.xml"/>
<delegateSystem systemIdStartString="http://www.oasis-
open.org/docbook/"
catalog="file:///usr/share/sgml/docbook/xml/dtd/4.2/catalog.xml"/>
<delegateURI uriStartString="http://www.oasis-open.org/docbook/"
catalog="file:///usr/share/sgml/docbook/xml/dtd/4.2/catalog.xml"/>
<delegatePublic publicIdStartString="ISO 8879:1986//ENTITIES"
catalog="file:///usr/share/sgml/entities/xml-iso-entities-
8879.1986/catalog.xml"/>
<delegatePublic publicIdStartString="-//Norman Walsh//DTD Website"
catalog="file:///usr/share/sgml/docbook/custom/website/2.4.1/catalog.xml"
/>
<rewriteURI
uriStartString="http://docbook.sourceforge.net/release/xsl/current"
rewritePrefix="usr/share/sgml/docbook/stylesheets/xsl/nwalsh"/>
```

04.11.2004

(c) 2004 H. Werntges, FB Informatik, FH Wiesbaden

40

Einschub: Spezielle Markup-Arten

CDATA section,
Conditional sections;
Nachtrag: Character data

- Benutzen Sie XML-Dokumententyp „sampletype“, um das Dokument selbst als *character data* zu erfassen. Das Ergebnis soll ein „gültiges“ XML-Dokument sein:

```
<?xml version="1.0"?>
<!DOCTYPE samplecode [
<!ELEMENT samplecode (#PCDATA)>
]>
<!-- Ein Kommentar -->
<samplecode>
    Hier mein Beispiel-Code incl. markup!
</samplecode>
```

- Einfache, aber umständliche Lösung:

```
...
<?xml version="1.0"?>
<!DOCTYPE samplecode [
<!ELEMENT samplecode (#PCDATA)>
]>
<!-- Ein Kommentar -->
<samplecode>
    Hier mein Beispiel-Code incl. markup!
</samplecode> ...
```

- Elegante Lösung:** Per CDATA section !

```
... <![CDATA[
<?xml version="1.0"?>
<!DOCTYPE samplecode [
<!ELEMENT samplecode (#PCDATA)>
]>
<!-- Ein Kommentar -->
<samplecode>
    Hier mein Beispiel-Code incl. markup!
</samplecode>
]]> ...
```

CDATA sections

- Zweck:
 - Behandlung von Markup als *character data*, analog etwa zur verbatim-Umgebung von LaTeX.
- Regeln:

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

```
<par>Und hier etwas XML-Quelltext:
<![CDATA[<greeting>Hello,
world!</greeting>]]> .</par>
```

04.11.2004

(c) 2004 H. Wernitges, FB Informatik, FH Wiesbaden

45

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.
- Fazit: 3 Methoden zur Darstellung reservierter Zeichen:
 - Zeichenreferenz,
 - Entity-Referenz,
 - CDATA *section*

04.11.2004

(c) 2004 H. Wernitges, FB Informatik, FH Wiesbaden

46

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:

Leerzeichen eingefügt!

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

04.11.2004

(c) 2004 H. Wernitges, FB Informatik, FH Wiesbaden

47

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.

04.11.2004

(c) 2004 H. Wernitges, FB Informatik, FH Wiesbaden

48

- Bemerkungen
 - Konstrukte nur in externen Deklarationsteilen verwendbar.
 - 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:

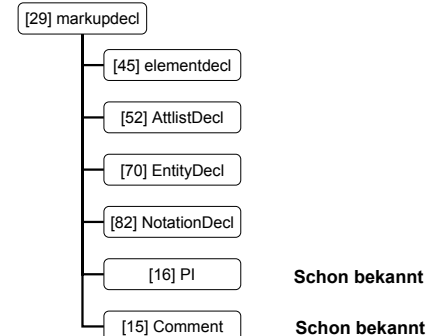
```
[61] conditionalSect ::= includeSect | ignoreSect
[62] includeSect ::= '<![[' S? 'INCLUDE' S? '['
    extSubsetDecl ']]>' [VC: Proper Conditional Section/PE Nesting]
[63] ignoreSect ::= '<![[' S? 'IGNORE' S? '['
    ignoreSectContents* ']]>'
    [VC: Proper Conditional Section/PE Nesting]
[64] ignoreSectContents ::= Ignore ('<![['
    ignoreSectContents ']]>' Ignore)*
[65] Ignore ::= Char* - (Char* ('<![[' | ']]>') Char*)
```

[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. eines *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.

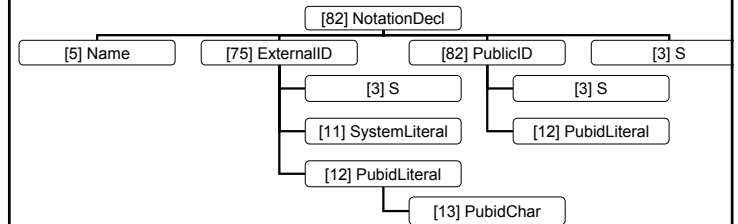
Markup-Deklarationen

NOTATION, ENTITY,
ELEMENT, ATTLIST



Die **NOTATION**-Deklaration

Notations
ExternalID: SYSTEM vs. PUBLIC
 FPI-Struktur, *catalogs*



- Sinn und Zweck von *notations*
 - *Notations* identifizieren über einen Namen:
 - das Format eines *unparsed entity* (wie z.B. das einer Bilddatei).
 - das Format von Elementen, die ein *notation attribute* besitzen (wird später behandelt)
 - die Anwendung, auf die sich eine PI bezieht
 - Eine *notation*-Deklaration schafft den Bezug zwischen dem *notation name* und einer (ausführlichen) ID zur näheren Beschreibung des referenzierten Objekts.
- Eindeutigkeit
 - Innerhalb eines XML-Dokuments darf der Name einer *notation* nur einmal vergeben werden.

- Beispiele:
 - Verweise zu externen Dokumentationen, die die Formate ISODATE und EUDATE näher beschreiben:


```
<INOTATION ISODATE SYSTEM "http://www.iso.ch/date_specification">
<INOTATION EUDATE SYSTEM "http://www.eu.eu/date_specification">
```
 - Nutzung dieser notations bei der Attributdeklaration (Vorgriff):


```
<!ELEMENT Today (#PCDATA)>
<!ATTLIST Today DATE-FORMAT NOTATION (ISODATE|EUDATE)
#REQUIRED>
```

Bemerkungen:
 Die Wirkung: Pflicht-Attribut „DATE-FORMAT“ von Element „Today“ ist vom Typ NOTATION. Es darf nur in den beiden zuvor deklarierten Notationen ISODATE oder EUDATE verwendet werden.
 Achtung: Der Parser ist nicht in der Lage, die Einhaltung dieser Regel zu prüfen. Sie hat rein dokumentarischen Charakter!
 - Verweis zu einer (lokalen) Hilfsanwendung


```
<INOTATION GIF SYSTEM "gifviewer.exe">
```

- Formale Regel:

```
[82] NotationDecl ::=
  '<!NOTATION' S Name S (ExternalID |
  PublicID) S? '>'
  [VC: Unique Notation Name]
```

ENTITY-Deklarationen

general - parameter
internal - external
parsed - unparsed

Ein einfaches Beispiel mit interner Deklaration:

```
<?xml version='1.0'?>
<!DOCTYPE test [
<!ELEMENT test (#PCDATA) >
<!ENTITY % xchardefs SYSTEM "sonderzeichendecl.ent">
%xchardefs;
<!ENTITY Abk 'Abk&ue;rzung'>
]>
<!-- Macro style -->
<test>Meine &Abk;!</test>
```

Ein einfaches Beispiel 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>
```

Entity-Klassifizierung

- XML kennt 5 verschiedene *entity*-Arten. Diese lassen sich durch 3 Begriffspaare abgrenzen.
- Von den dadurch „aufgespannten“ $2^3=8$ Triplets sind 3 nicht sinnvoll, so dass also 5 *entity*-Arten resultieren.
- Die *external parsed entities* werden hier in zwei Unterarten unterschieden: SYSTEM und PUBLIC.

ENTITY	parsed		unparsed (nur <i>general</i>)
	general	parameter	
internal	(ja)	(ja)	(ex. nicht!)
external	SYSTEM	SYSTEM	SYSTEM NDATA
	PUBLIC	PUBLIC	PUBLIC NDATA

Die 3 Entity-Begriffspaare

- *general / parameter*
 - *General entities* kennen wir z.B. vom Umgang mit Sonderzeichen. Sie können fast überall im Dokument auftauchen - daher „general“ - und werden i.d.R. vom Autor des Dokuments vergeben.
 - *Parameter entities* sind reserviert für Zwecke innerhalb der DTD. Sie werden von *general entities* unterschieden, da sie von DTD-Autoren vergeben werden. Durch den separaten Namensraum besteht keine Kollisionsgefahr mit den Arbeiten der Dokument-Autoren.
- *internal / external*
 - *Internal entities* werden innerhalb des *document entity* deklariert.
 - *External entities* werden über URL u.ä. referenziert und müssen vom *Parser* erst einmal geholt und separat gelesen werden - was nicht validierende *Parser* nicht immer unterstützen!
- *parsed / unparsed*
 - *Parsed entities* enthalten XML-Daten,
 - auf *unparsed entities* wird nur verwiesen, i.d.R. per *notation*.

ENTITY-Deklarationen

Formale Regeln:

```
[70] EntityDecl ::= GEDecl | PEDecl
[71] GEDecl ::=
    '<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl ::=
    '<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef ::=
    EntityValue | (ExternalID NDataDecl?)
[74] PEDef ::= EntityValue | ExternalID
[76] NDataDecl ::= S 'NDATA' S Name
    [VC: Notation Declared]
```

Entity-Regeln

- Die Regeln in Worten:
 - Entweder deklariert man ein *general entity* oder ein *parameter entity*.
Die Unterscheidung trifft allein das „%“-Zeichen!
 - Ein *parameter entity* wird entweder (intern) über seinen Wert oder über ein *ExternalID* definiert.
 - Bei einem *general entity* darf die *ExternalID* darüberhinaus noch vom Typ NDATA (*unparsed data*) sein.
Diese *unparsed data* werden mit Hilfe einer *notation* deklariert (die natürlich existieren muss).

Entity-Referenzen, constraints

[67] **Reference** ::= **EntityRef** | **CharRef**

[68] **EntityRef** ::= '&' **Name** ';' ;
[WFC: Entity Declared] , [VC: Entity Declared]
[WFC: Parsed Entity] , [WFC: No Recursion]

[69] **PEReference** ::= '% ' **Name** ';' ;
[VC: Entity Declared] , [WFC: No Recursion] , [WFC: In DTD]

- Deklarierte(!) *entities* werden über ihren Namen, gefolgt von einem Semikolon, referenziert. Sie dürfen selbst wiederum *entity*-Referenzen enthalten, aber diese dürfen nicht zu Rekursionen führen!
- *Parameter entity*-Referenzen beginnen mit einem '%'. Diese Referenzen können nur innerhalb der DTD verwendet werden, denn außerhalb des Prologs verliert '%' die *markup*-Eigenschaften.
- *General entity*-Referenzen beginnen mit einem '&', analog zu *character*-Referenzen. Nur *parsed entities* lassen sich expandieren und können daher Referenzen besitzen.

„Trickreiche“ Entity-Referenzen

- Während *character*-Referenzen als auch *parameter-entity* Referenzen in Werten von Deklarationen (DTD-Teil) expandiert werden, gilt das nicht für *general-entity* Referenzen. Dazu ein Beispiel aus XML 1.0, D:
- Deklaration von *example*:

```
<!ENTITY example " <p>An ampersand (&#38;#38;) may be
escaped numerically (&#38;#38;#38;) or with a general
entity (&amp;).</p>" >
```

- Wert von *example* (nach dem Parser-Lauf):

```
<p>An ampersand (&#38;) may be escaped numerically
(&#38;#38;#38;) or with a general entity (&amp;).</p>
```

- Eine Referenz &example; im Dokumenttext wird expandiert zu:

```
<p>An ampersand (&) may be escaped numerically (&#38;#38;)
or with a general entity (&amp;).</p>
```

„Trickreiche“ Entity-Referenzen

Ein komplizierteres Beispiel:

```
<?xml version='1.0'?>
<!DOCTYPE test [
<!ELEMENT test (#PCDATA) >
<!ENTITY % xx '&#37;zz;' >
<!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
%xx;
]>
<test>This sample shows a &tricky; method.</test>
```

Wie wird „&tricky;“ in Element „test“ expandiert?

Bem.: Die Deklaration zu ELEMENT wird zwar erst später behandelt, vervollständigt aber nur das Beispiel zu einem gültigen XML-Dokument, ohne die *entity*-Problematik zu „stören“.

Entity-Deklarationen: Priorisierungsregeln

- Es ist zulässig, denselben *entity*-Namen mehrfach in einer Deklaration zu verwenden. In einem solchen Fall stellt sich die Frage, nach welchen Regeln der Namenskonflikt aufgelöst wird.
- Die Regeln dazu:
 - (1) „**The first instance binds**“ - die zuerst vom *Parser* angetroffene Deklaration ist die wirksame, nachfolgende werden ignoriert.
 - (2) Interne Deklarationen haben Vorrang vor Deklarationen in externen entities. - Regel (2) folgt aus Regel (1), wenn man unterstellt, dass *Parser* immer die internen Deklarationen vor den externen lesen.
- Nützliche Konsequenzen für die Praxis:
 - Autoren können aus externen DTDs „geerbte“ (*general*) *entities* lokal umdefinieren, indem sie sie lokal in der Dokument-Deklaration neu deklarieren.
 - Die externe DTD - selten unter Kontrolle des Autors - muss dazu nicht geändert werden!
 - Tests mit Varianten sind durch Einfügen am Anfang leicht möglich, ohne dass die Originale entfernt oder auskommentiert werden müssten.

entity- und char-Referenzen im Kontext

Referenz	Entity ref.				Character ref.
	parameter	internal general	external parsed general	unparsed	
im Inhalt	nicht erkannt	expandiert	expandiert ³⁾	verboten	expandiert
im Attributwert	nicht erkannt	expandiert ¹⁾	verboten	verboten	expandiert
erscheint als Attributwert	nicht erkannt	verboten	verboten	gemeldet	nicht erkannt
in entity-Wert	expandiert ¹⁾	unverändert	unverändert	verboten	expandiert
in DTD	expandiert ²⁾	verboten	verboten	verboten	verboten

entity- und char-Referenzen im Kontext

- Anmerkungen
 - Die Wirkung von *entity*- und *character*-Referenzen hängt vom Kontext ab und verwirrt leicht.
 - Die Tabelle stellt die Fallunterschiede zusammen.
 - Hier wird kein Versuch einer vollständigen Beschreibung unternommen. Die Details finden Sie bei Bedarf in Kapitel 4.4 der XML 1.0-Spezifikation, Anmerkungen 1) - 3) ebenfalls.
- Empfehlungen:
 - Nur bei unerwartetem Verhalten von *entity*-Referenzen sollten Sie die Tabelle und ggf. die Originalliteratur konsultieren.
 - Misstrauen Sie Ihrem Parser - erst die Übereinstimmung mehrerer XML Prozessoren spricht für ein Verständnisproblem.
 - Bevor Sie abgelehnte Konstrukte verwerfen: Testen Sie, ob sich das Parserverhalten ändert, wenn die Konstrukte vom internen Subset in ein externes *entity* - oder umgekehrt - verlagert werden.

Fachhochschule Wiesbaden - Fachbereich Informatik

Die zentralen Deklarationen

ELEMENT
ATTLIST

Die ELEMENT-Deklaration

- Eine Element-Deklaration verbindet den - eindeutig zu vergebenden - Namen des Elements mit einer Inhaltsbeschreibung:


```
[45] elementdecl ::=
            '<!ELEMENT' S Name S contentspec S? '>'
            [VC: Unique Element Type Declaration]
```
- Der Inhalt eines Elements kann leer sein, „beliebig“ (Sonderfall!), von einem gemischten Typ, oder aus Kind-Elementen bestehen:


```
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
```
- Den Typ „EMPTY“ kennen wir bereits. Derartige Elemente können nur Attribute enthalten.
- Mit „ANY“ können Parserprüfungen vorübergehend außer Kraft gesetzt werden. Während der DTD-Entwicklung nützlich, für Produktionszwecke zu vermeiden.

Die ELEMENT-Deklaration

[51] **Mixed** ::=

```
'(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' |  
'(' S? '#PCDATA' S? ')'
```

[VC: Proper Group/PE Nesting] [VC: No Duplicate Types]

- Der Inhaltstyp „Mixed“ beginnt **immer** mit „#PCDATA“!
- #PCDATA steht für *parsed character data* und meint Freitext, der durchaus auch *markup* wie *entities* oder *CDATA sections* enthalten darf, nur keine weiteren Elemente!
- „Mixed“ darf aus einer Folge von #PCDATA und Elementen bestehen.
- Mehrere Elemente dürfen direkt aufeinander folgen, aber zwischen zwei #PCDATA-Abschnitten muss immer ein Element sein - wie sonst sollten die Abschnitte auch getrennt werden?

Die ELEMENT-Deklaration

- Beispiele für ELEMENT-Deklarationen mit „Mixed“:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
```

Ein HTML-artiges Beispiel (*paragraph*)

```
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special;  
| %form;)* >
```

Auch *parameter entity*-Referenzen sind hier möglich.

```
<!ELEMENT b (#PCDATA)>
```

Im einfachsten Fall besteht Typ „Mixed“ nur aus #PCDATA.

Merke: Stets erscheint #PCDATA, und immer an erster Stelle!

- Anwendungsbeispiel (in den Nutzdaten)

```
<p>Dieser Absatz ist <em>wichtig</em> und sollte  
hervorgehoben werden.  
Wie schon <LitRef refId="123"/> <Kommentar>Zitat  
noch besorgen! </Kommentar> beschrieb, ... </p>
```

Die ELEMENT-Deklaration

[47] **children** ::= (choice | seq) ('?' | '*' | '+')?

[48] **cp** ::= (Name | choice | seq) ('?' | '*' | '+')?

[49] **choice** ::= '(' S? cp (S? '|' S? cp)+ S? ')'
[VC: Proper Group/PE Nesting]

[50] **seq** ::= '(' S? cp (S? ',' S? cp)* S? ')'
[VC: Proper Group/PE Nesting]

- Die *children* bestehen entweder aus einer Auswahl (*choice*) oder einer Sequenz (*sequence*), die jeweils eines der Wiederholzeichen tragen können.
- Eine Sequenz ist eine kommaseparierte Liste (Aufzählung mit bestimmter Reihenfolge) von Kompositen (*cp*), im einfachsten Fall nur ein *cp*.
- Eine Auswahl besteht aus mindestens zwei Kompositen, die alternativ zur Verfügung stehen.
- Ein Komposit ist eine beliebige Folge einzelner Elemente, Auswahl- und Sequenz-Listen, im einfachsten Fall ein optionales einzelnes Element.

Die ELEMENT-Deklaration

- Beispiele für ELEMENT-Deklarationen mit „children“:

```
<!ELEMENT spec (front, body, back?)>
```

Eine einfache Sequenz

```
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
```

Eine Sequenz einzelner Elemente und einer Auswahl, einschließlich Wiederholfaktoren

```
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

Eine Auswahl, die sich beliebig oft wiederholen darf, und die mittels *parameter entity*-Referenzen definiert wird.

- Anmerkungen zu den *parameter entity*-Referenzen:
 - Sie dürfen nicht vollständig zu *white space* expandieren
 - Der Expansionstext darf nicht mit den in der Deklaration verwendeten Verbindungszeichen (| oder ,) kollidieren.

Die ELEMENT-Deklaration

- Beispiel zu Übung 01:

```
<!ELEMENT Codetabelle (Eintrag+)>
<!ELEMENT Eintrag (Zeichen,
  Beschreibung, Unicode, ISO-Code?)>
<!ELEMENT Zeichen (#PCDATA)>
<!ELEMENT Beschreibung (#PCDATA)>
<!ELEMENT Unicode (#PCDATA)>
<!ELEMENT ISO-Code (#PCDATA)>
<!-- Vorgriff: -->
<!ATTLIST ISO-Code
  TabNr NMTOKEN #REQUIRED>
```

Element oder Attribut?

- Das Problem:
 - Bei der Datenmodellierung entsteht oft die Frage, ob ein bestimmtes Datenelement als XML-Element oder als Attribut implementiert werden soll.
 - **Es gibt keine formale Regel bzw. eindeutige Antwort!**
- Faustregeln zur Entscheidungsfindung:
 - Elemente gestatten spätere Verfeinerung
 - Elemente sind für die „eigentlichen Nutzdaten“ gedacht, Attribute für Ergänzungen (einfachen Typs), u. Metadaten
 - Attribute gestatten strengere Typisierung
 - Nur für Attribute gibt es Default-Regeln/-Belegungen
- Hinweis:
 - XML Schema wird diese Grenzen später verwischen!

Die ATTLIST-Deklaration

- Die Attribute eines Elements werden gemeinsam (als Liste) deklariert. Dem Namen des Elements wird eine Liste seiner Attribute und deren Beschreibungen, getrennt nur durch *white space*, zugeordnet:

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
```

```
[53] AttDef ::= S Name S AttType S DefaultDecl
```

- Jedes Attribut erhält einen Namen, einen Attributtyp und eine Regelung zur Befüllung (*default-Wert*, Auswahl, muß/kann)

Es gibt drei Attributtypen:

```
[54] AttType ::=
  StringType | TokenizedType | EnumeratedType
```

Der „StringType“:

- Er akzeptiert beliebige *character data* (Strings):

```
[55] StringType ::= 'CDATA'
```

Die ATTLIST-Deklaration

Der „Tokenized“-Typ:

```
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS' |
  'ENTITY' | 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'
```

- ID:
 - Elemente können dokumentweit eindeutig identifiziert werden über ein ID-Attribut. Ein solcher Attributwert unterliegt der Bildungsregel für Names und funktioniert ähnlich wie ein *unique key* bei Datenbankzugriffen:
 - Jedes Element darf höchstens ein Attribut vom Typ ID besitzen.
 - Jeder ID-Wert darf nur einmal im gesamten XML-Dokument vergeben werden (also nicht nur pro Elementtyp). Dies ist eine erhebliche Einschränkung!
 - Als Default-Deklarationen (s.u.) sind nur #IMPLIED und #REQUIRED zulässig.
- Beispiel:

```
<!ELEMENT Student (Name, Fachrichtung, Studiengang, ...)>
<!ATTLIST Student MatrNr ID #REQUIRED>      <!-- Vorsicht - Falle... -->
```

Die ATTLIST-Deklaration

- IDREF, IDREFS:
 - Mit Attributen vom Typ IDREF stellt man Verweise auf Elemente, die die referenzierten IDs tragen.
 - Diese Verweise dürfen nicht „ins Leere zeigen“, d.h. die referenzierten IDs müssen im Dokument existieren.
 - Der Wert eines derartigen Attributs muß der Regel für *Names* genügen. Jede ID dieser Liste muss im Dokument existieren.
 - IDREFS sind Listen von IDREF-Einträgen, separiert mit S
- Beispiel:

```
<!ELEMENT Kursteilnehmer (Student+)>
<!ATTLIST Kursteilnehmer MatrNrListe IDREFS #IMPLIED>
```

Die ATTLIST-Deklaration

- ENTITY:
 - Ein Attribut dieses Typs nimmt den Namen eines *unparsed entity* auf.
 - Es gelten die Vergaberegeln für *Name*.
 - In der DTD des Dokuments muß ein entsprechendes *entity* deklariert sein.
- Beispiel:

```
<!ENTITY Passbild-von-123456 SYSTEM
"file:///opt/bilder/123456.jpg" NDATA JPEG>
<!ENTITY Passbild-von-123457 SYSTEM
"file:///opt/bilder/123457.jpg" NDATA JPEG>
<!ELEMENT Student (Name, Fachrichtung, Studiengang, ...)>
<!ATTLIST Student MatrNr ID #IMPLIED
Passbild ENTITY #IMPLIED> ]> ...

<Student MatrNr="M123456"
Passbild="Passbild-von-123456"> ... </Student>
```

Die ATTLIST-Deklaration

- ENTITIES:
 - Ein Attribut dieses Typs nimmt eine Liste der Namen von *unparsed entities* auf. Es gelten die Vergaberegeln für *Names*.
 - Im DTD des Dokuments müssen entsprechende *entities* deklariert sein. **Beispiel:**

```
<!ENTITY Passbild-von-123456 SYSTEM
"file:///opt/bilder/123456.jpg" NDATA JPEG>
<!ENTITY Passbild-von-123457 SYSTEM
"file:///opt/bilder/123457.jpg" NDATA JPEG>
<!ELEMENT Teilnehmer (Student+)>
<!ATTLIST Teilnehmer Passbilder ENTITIES #IMPLIED
MatrNrListe IDREFS #REQUIRED>

<Teilnehmer MatrNr="M123456 M123457"
Passbild="Passbild-von-123456 Passbild-von-123457">
... </Teilnehmer>
```

Die ATTLIST-Deklaration

- NMTOKEN, NMTOKENS:
 - Attribute dieser beiden Typen werden oft verwendet. Ein NMTOKEN unterliegt nur geringen Einschränkungen, so dass dieser Attributtyp für viele Anwendungsfälle verwendet wird.
 - ACHTUNG - „Tücke im Detail“:
NMTOKEN(S) unterliegen anderen Normierungsregeln als CDATA (s.u.)
 - Für die Attributwerte gelten die Regeln für *Nmtoken* bzw. *Nmtokens*.
 - NMTOKENS entsprechen einfach einer Liste von NMTOKEN-Werten, mit *white space* separiert.
- Beispiel:

```
<!ELEMENT Teilnehmer (Student+)>
<!ELEMENT Student
(Name, Fachrichtung, Studiengang, ...)>
<!ATTLIST Student Belegte-Kurse NMTOKENS #IMPLIED>
```

Die ATTLIST-Deklaration

Der „Enumerated“-Typ:

```
[57] EnumeratedType ::= NotationType | Enumeration
```

```
[58] NotationType ::=
'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
```

```
[59] Enumeration ::=
'(' S? Nmtoken (S? '|' S? Nmtoken)* S? '),
```

„EnumeratedType“ gliedert sich in zwei Arten von Aufzählungstypen:

• 1) NotationType:

- Dem Schlüsselwort NOTATION folgt eine Auswahl von NOTATION-Referenzen, also Referenzen auf existierende NOTATION-Deklarationen.
- Unser Beispiel zur NOTATION-Deklaration verwendete diesen Attributtyp:

```
<!ELEMENT Today (#PCDATA)>
<!ATTLIST Today DATE-FORMAT NOTATION (ISODATE|EUDATE)
#REQUIRED>
```

Die ATTLIST-Deklaration

- Einschränkungen zu NotationType:

Ein Element darf höchstens ein NOTATION Attribut erhalten,

EMPTY Elemente dürfen kein NOTATION Attribut erhalten

• 2) Enumeration-Typ:

- Dieser Attributtyp besteht einfach aus einer Auswahlliste von *name tokens*.
- Diese müssen nur der Bildungsregel zu *Nmtoken* genügen und können ansonsten in der DTD frei vergeben werden.
- Die *name tokens* werden ohne *quotation* aufgelistet.
- Bei der Validierung prüft der XML-Prozessor, ob Elementinstanzen nur Attribute mit Werten aus der hiermit hinterlegten Liste von *name tokens* annehmen.
- Beispiel: Attribut „Wochentag“ des Elements „Vorlesung“ beschreibe den Tag der Veranstaltung im laufenden Semester:

```
<ELEMENT Vorlesung (Titel, Beschreibung, ...)>
<ATTLIST Vorlesung Wochentag (Montag|Dienstag|...|Sonntag) #IMPLIED>
```

Die ATTLIST-Deklaration

Attribut-Defaults:

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' |
(( '#FIXED' S)? AttValue)
```

- Gemäß Regel [53] wird jedem Attributnamen ein Typ und eine Deklaration über seine *default*-Befüllung zugeordnet.
- Die Spezifikationen unterscheiden hier drei Fälle:

#REQUIRED

- So deklarierte Attribute müssen in Elementinstanzen stets gefüllt werden, und zwar innerhalb des Dokuments selbst. Beispiel: Typ „ID“
- Hinweis: Der Begriff Attribut-„*default*“ ist hier irreführend.

#IMPLIED

- Auch ein so deklariertes Attributtyp wird nur innerhalb des Dokuments befüllt - auch #IMPLIED stellt keine *default*-Befüllung zur Verfügung.
- Im Unterschied zu #REQUIRED darf das Attribut aber auch fehlen.

Die ATTLIST-Deklaration

AttValue

- Durch einfache Angabe eines Attributwerts (diesmal aber *quoted!*) wird dieser zum *default*-Befüllungswert deklariert.
- Derartige Attribute werden von validierenden Parsern also stets gefüllt an die Anwendung durchgereicht, wobei die Befüllung innerhalb des Dokuments stets Vorrang vor der *default*-Befüllung über die DTD genießt.
- VORSICHT: Nicht validierende Parser führen derartige *default*-Befüllungen nicht immer aus, insbesondere wenn die DTD sich in einem externen *entity* befindet. Vergleiche dazu auch die *standalone document declaration*.

#FIXED AttValue

- Dies ist eine Variante der *AttValue*-Befüllung. Hiermit wird der angegebene *default*-Attributwert zum einzig erlaubten Wert erklärt!
- Sinnvoll ist dies insbesondere für die flexible Verwaltung von Eigenschaften, die - für eine Übergangszeit - nur einen gültigen Wert besitzen.
- DTD-Designer „sperrern“ so Attribute für XML-Autoren - vgl. *Namespace*s.



- Beispiele (aus XML 1.0):

```
<!ATTLIST termdef
  id      ID      #REQUIRED
  name    CDATA   #IMPLIED>

<!ATTLIST list
  type    (bullets|ordered|glossary)
          "ordered">

<!ATTLIST form
  method  CDATA   #FIXED "POST">
```