Formaler Aufbau eines XML-Dokuments

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

Formaler Aufbau eines XML-Dokuments

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

Bemerkungen:

- Ein XML-Dokument besteht aus dem Prolog, einem Element, und verschiedenen Anhängen.
- Dieses Element heißt
  - *document element*, oder auch
  - *root element*
- Es besteht wiederum aus Unter-Elementen u.a.!
  - Typische Begriffsbildung: *parent elements, child elements*
- Definitionen von prolog, element, Misc erfolgen noch
- Vorab: Der Prolog kann auch leer sein / fehlen.

Beispiel:

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

ist bereits ein (wohlgeformtes) XML-Dokument.

Der Prolog

```
[22] prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
[27] Misc ::= Comment | PI | S
```

Beispiel

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

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.

Die XML-Deklaration

```
[23] XMLDecl ::=
  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq
       ("'" VersionNum "'" | '"' VersionNum '"')
[25] Eq ::= S? '=' S?
[26] VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
```

Beispiel

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

- Versionsangabe
  - Muß ggf. angegeben werden
  - Bisher nur „1.0“ veröffentlicht.

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, …)*
    - 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

Die XML-Deklaration

- *standalone* Dokumentdeklaration:

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

- Bemerkungen
  - Zulässige Werte sind nur “*yes*“ und “*no*“, *default*-Wert 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.
  - Diese Deklaration ist vergleichsweise unwichtig.

Die Text-Deklaration

```
[23] 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 einer jeden externen *entity* (außer der *document entity* selbst - dort ist die XML-Deklaration vorgeschrieben).
  - Zweck ist die Mitteilung des verwendeten Zeichensatzes einer jeden *entity* an den *Parser*.
  - Der *Parser* kann jede *entity* getrennt beim Lesen normieren, es ist also möglich, jeder *entity* ihren eigenen Zeichensatz zuzuordnen.

Die Dokumententyp-Deklaration

Formale Regel:

```
[28] doctypedecl ::= '<!DOCTYPE' S Name
       (S ExternalID)? S?
       ('[' (markupdecl | DeclSep)* ']' S?)? '>'
```

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>
```

Anmerkungen zur Dokumententyp-Deklaration

- 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 ([…]).
- 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 externe *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“.

Markup-Deklarationen

NOTATION, ENTITY,
ELEMENT, ATTLIST

Die *NOTATION*-Deklaration

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

Notations

- Sinn und Zweck von *notations*
  - *Notations* identifizieren über einen Namen
    - das Format einer *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 declaration 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.

Notations

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

    <!NOTATION ISODATE SYSTEM "http://www.iso.ch/date_specification">

    <!NOTATION 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

    <!NOTATION GIF SYSTEM "gifviewer.exe">

Notations

- Formale Regeln:

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

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


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

- Bemerkungen
  - Regel [11] definiert SystemLiteral, Regel [12] definiert PubidLiteral (s.o.)

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
    - a) *Fixed Public Identifier* (FPI)
      - Global einheitlich zu verwendender Name für das gemeinte Objekt
    - b) *System identifier*
      - Vom *Parser* zu verwenden, wenn er FPI nicht auflösen kann
      - In SGML nicht vorgesenen (optional?)
    - Formal wie ein SYSTEM *identifier* mit vorgelagertem FPI

FPI-Struktur, *catalog*

- Formal public identifiers (FPI): Aufbau
  - *prefix*
    - entweder ‚+' (registriert) oder ‚-' (nicht registriert)
    - Manchmal „ISOxxxx" (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...

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

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"
  "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 Number) - hier nicht behandelt
    - Ein aktuellerer, konzeptionell zu 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“*

ENTITY-Deklarationen

general - parameter

internal - external

parsed - unparsed

Entity-Klassifizierung

- XML kennt 5 verschiedene *entity*-Arten. Diese lassen sich durch 3 Begriffspaare abgrenzen.
- Von den dadurch „aufgespannten“ 2^3=8 Tripletts 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
	general	parameter	(nur *general*)
internal	(ja)	(ja)	(ex. nicht!)
external	SYSTEM PUBLIC	SYSTEM PUBLIC	SYSTEM NDATA 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-Entwicklern vergeben werden. Durch den separaten Namensraum besteht keine Kollisionsgefahr mit den Arbeiten der Dokument-Autoren.
- *internal / external*
  - *Internal entities* werden innerhalb der *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* (wie Bildmaterial) 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 eine *general entity* oder eine *parameter entity*.
    - Die Unterscheidung trifft allein das „%“-Zeichen!
  - Eine *parameter entity* wird entweder (intern) über ihren Wert oder über eine *ExternalID* definiert.
  - Bei einer *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 einen ‚%‘. Diese Referenzen können nur innerhalb der DTD verwendet werden, denn außerhalb des Prologs verliert ‚%‘ die *markup*-Eigenschaften.
- G*eneral 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;amp;).</p>" >
```

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

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

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

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


„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 eine externe entity - oder umgekehrt - verlagert werden.

Die zentralen Deklarationen

ELEMENT

ATTLIST

Vorab: Elemente und Attribute

start-tag, end-tag, empty-element tag

Attribute, Inhalt

Übung im Umgang mit der XML 1.0-Spezifikation: Dort nachschlagen!

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 entities* 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)
  - Dieser Absatz ist wichtig und sollte hervorgehoben werden. Wie schon <LitRef refId="123"/> <Kommentar>Zitat noch besorgen! </Kommentar> beschrieb, …

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 Komposits (*cp*), im einfachsten Fall nur ein *cp*.
- Eine Auswahl besteht aus mindestens zwei Komposits, 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 references* definiert wird.

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

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:
  - Mit Attributen vom Typ IDREF erstellt 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.
- Beispiel:

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


Die ATTLIST-Deklaration

- ENTITY:
  - Ein Attribut dieses Typs nimmt den Namen einer *unparsed entity* auf.
  - Es gelten die Vergaberegeln für *Name.*
  - In der DTD des Dokuments muß eine entsprechende *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>          <!-- usw. -->
<!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>          <!-- usw. -->
<!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 stellen 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) EnumerationType:
  - 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 deklarierter 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 einer 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 „sperren“ so Attribute für XML-Autoren - vgl. *Namespaces*.

Die ATTLIST-Deklaration

- 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">
```


Attribut-Konventionen und Normierungsregeln

xml:lang, xml:space

Zeilenenden, Attributwerte

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 - zu deklarieren, 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.

Attribut-Konventionen - **xml:space**

- 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: Zeilenenden

- Situation:
  - Verschiedene Betriebssysteme verwenden unterschiedliche Konventionen zur Kennzeichnung von Zeilenumbrüchen
    - (DOS/Win: (#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.

Normierungsregeln: Attributwerte

- 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 sein, 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.

Normierungsregeln: Attributwerte

- Der Normierungsalgorithmus:
  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 ein *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;&#xD;">
    ```
- Attributspezifikation 1: att="

  x y z"
  - Normiert, NMTOKENS: x y z
  - Normiert, CDATA: #x20 #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="

A

B
"
  - Normiert, NMTOKENS: #xD #xD A #xA #xA B #xD #xA
  - Normiert, CDATA: #xD #xD A #xA #xA B #xD #xA

Normierungsregeln: Attributwerte

- Bemerkungen zu den Beispielen
  - 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
  - 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.
  - Fall 3:
    - Die 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="LB"/>
</Dozent>
```

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

Einleitendes Beispiel-Dokument

```
<?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="LB"/>
</Dozent>
```


Nachwort zu XML 1.0

- Errata zu 1.0
  - Auch zur hier zugrundegelegten korrigierten Version von XML 1.0 gibt es inzwischen zahlreiche Änderungsmeldungen und Fehlerreports
  - Diese liegen als „Errata“ vor und sollten herangezogen werden, wenn man
    - einen Fehler in den Spezifikationen vermutet
    - sich ein Parserverhalten partout nicht erklären kann.
- XML 1.1 ist in Vorbereitung („*candidate recommendation*“)
  - Berücksichtigt Änderungen zwischen Unicode 2.0 und 3.1
  - 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