Kapitel 4
XML (Extensible Markup Language)

Einführung

- SGML *sehr* ausdruckskräftig, flexibel
  HTML sehr spezialisiert.
- Sommer 1996: John Bosak (Sun Microsystems) begründet die XML Working Group (SGML-Experten), Zusammenarbeit mit dem W3C.
  Entwicklung einer einfacher zu implementierenden Teilmenge von SGML
  `http://www.w3.org/XML/`: Die Homepage für XML beim W3C

⇒ XML ist eine "stripped-down version of SGML".

- Um XML zu verstehen, muß man aber nicht alles über SGML wissen ...

HTML

... kennt doch jeder, oder?

- Tags: paarweise öffnen und schließen: <TABLE> ... </TABLE>
- "leere" Tags: ohne schließenden Tag
, <HR>
- <P> ist *eigentlich* kein leerer Tag (sollte am Ende des Absatzes auch wieder geschlossen werden)!
- Attribute: <TD colspan = 2> ... </TD>
- leere Tags mit Attributen:
  <IMG SRC="http://www.informatik.uni-freiburg.de/photo.jpg" ALIGN=LEFT>
- Inhalt von Tags: <TD>123456</TD>
- Geschachtelte Tags: <TH><B>Name</B></TH>

  <A href="http:www.informatik.uni-freiburg.de">
  <B>Homepage des IIF</B></A>

⇒ Hierarchische Struktur

- Entities: ä = ä ß = ß

HTML

- Browser muß Tags interpretieren können
  → Darstellung jedes Tags für alle (?) Browser fest definiert.
- feste Vorstellungen, wie Tags geschachtelt sein sollen
  (sind in einer DTD (document type description) festgelegt)

```
<body><H1><H2>
            <P> ... </P>
            <H2>
            <P> ... </P>
       <H1><H2>
            <P> ... </P>
</body>
```

- analog für Tabellen und Listen ...
- Realität: Struktur wird nur selten eingehalten
  - schließende Tags werden weggelassen
  - Struktur wird nicht eingehalten (Gliederungsebenen)

  → Parser muß vervollständigen

WAS BRINGT DIE KENNTNIS VON HTML FÜR XML?

- intuitive Vorstellung – aber nur die *ASCII-Repräsentation*
- Das ist noch *kein Datenmodell*
- keine Anfragesprache
- nur eine sehr eingeschränkte Sichtweise:
  HTML ist eine Markup-Sprache für Browser
  (Hinweis: man "sieht" kein HTML im Browser, sondern nur das was der Browser daraus macht).

  Mehr nicht.

ZIELE BEI DER ENTWICKLUNG VON XML

- XML soll über das Internet direkt und auf einfache Weise nutzbar sein (Unicode-Files)
- XML soll eine große Bandbreite von Anwendungen unterstützen
- XML soll zu SGML kompatibel sein
- XML-Dokumente sollen für Menschen lesbar und nachvollziehbar sein
- XML-Dokumente sollen einfach zu erstellen sein
- Es soll einfach sein, Programme zu schreiben, die XML-Dokumente auswerten

WAS UNTERSCHEIDET XML VON HTML?

- Ziel: *Nicht Browsing*, sondern Speicherung von (semistrukturierten) Daten (vgl. SGML)
- SGML erlaubt die Definition von neuen Tags, entsprechend der Anwendungssemantik, jede SGML-Applikation besitzt ihre eigenen *semantischen Tags*.
  Diese werden in einer DTD (Document Type Description) definiert.
- HTML ist *eine* SGML-Applikation (vgl. `<HTML> am Anfang jedes Dokuments </HTML>`), die die DTD "HTML.dtd" verwendet.
- In XML kann man ebenfalls (nahezu) beliebige Tags definieren und verwenden:

```
<country> ... </country>
<city> ... </city>
<province> ... </province>
<name> ... </name>
```

- Diese *Elemente* beschreiben dann Objekte der Anwendungswelt.

XML als Metasprache für spezielle Sprachen

- Für jede Anwendung kann man festlegen, welche “Begriffe” als Elementnamen etc. verwendet werden sollen:
  ⇒ Dokumenttyp-Definition (DTD)
- Die Menge der erlaubten Elementnamen sowie die erlaubten Schachtelungen und Attribute werden in der DTD des Dokument(typ)s festgelegt.
- Man kann für jede Anwendung eine DTD definieren, oder eine bereits verfügbare verwenden → Standard-DTDs
- DTD beschreibt das *Schema*
- XML ist eine *Metasprache*, jede DTD definiert eine eigene Sprache
- HTML besitzt (als SGML-Applikation) eine DTD

Beispiel: Mondial

```
<mondial>
   :
 <country code="D" capital="cty-Germany-Berlin"
          memberships="EU NATO UN ...">
   <name>Germany</name>
   <encompassed continent="europe">100</encompassed>
   <population year="1995">83536115</population>
   <ethnicgroup name="German">95.1</ethnicgroup>
   <ethnicgroup name="Italians">0.7</ethnicgroup>
   <religion name="Roman Catholic">37</religion>
   <religion name="Protestant">45</religion>
   <language name="German">100</language>
   <border country="F" length="451"/>
   <border country="A" length="784"/>
   <border country="CZ" length="646"/>
     :
```

Beispiel: Mondial (Forts.

```
      :
    <province id="prov-D-berlin" capital="city-D-berlin">
      <name>Berlin</name>
      <population year="1995">3472009</population>
      <city id="city-D-berlin">
        <name>Berlin</name> <population year="1995">3472009</population>
      </city>
    </province>
    <province id="prov-D-baden-wuerttemberg" capital="cty-D-stuttgart">
      <population year="1995">10272069</population>
      <name>Baden Wuerttemberg</name>
      <city id="cty-D-stuttgart">
        <name>Stuttgart</name> <population year="95">588482</population>
      </city>
      <city id="cty-D-mannheim"> ... </city>
      :
    </province>
    :
  </country>
  :
<mondial>
```


MAN SIEHT:

- Hierarchisches “Datenmodell”
- Subelemente, Attribute
- Referenzen
- Reihenfolge? Dokumente – ja, Datenbanken – nein

Beispiele auf

`http://user.informatik.uni-goettingen.de/~may/Mondial/#XML`

XML ALS DATENMODELL

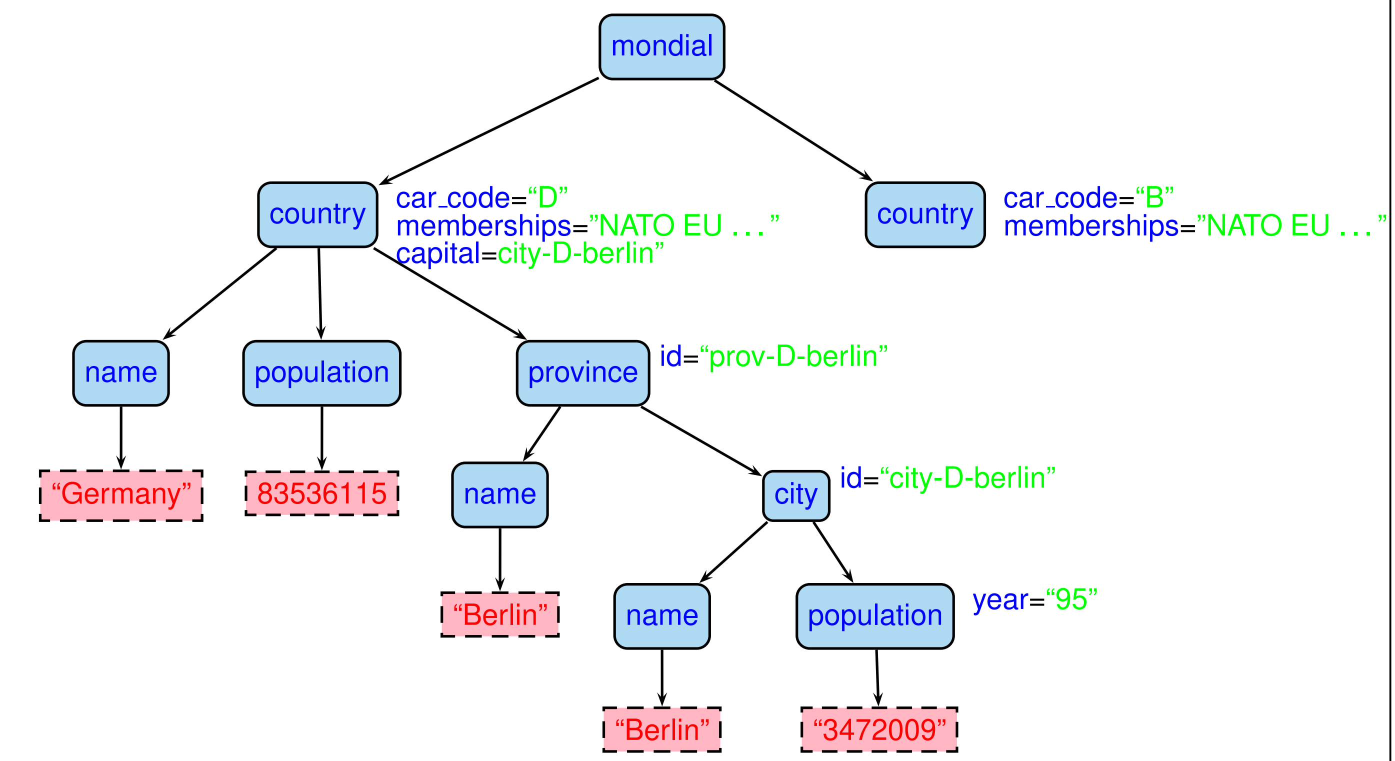
XML ist weit mehr als nur die oben gezeigte ASCII-Repräsentation, die man von HTML kennt (vergleiche einführender Vortrag)

- abstraktes Datenmodell (vergleichbar dem relationalen DM)
- viele Konzepte um XML herum
  (XML ist *keine* Programmiersprache!)

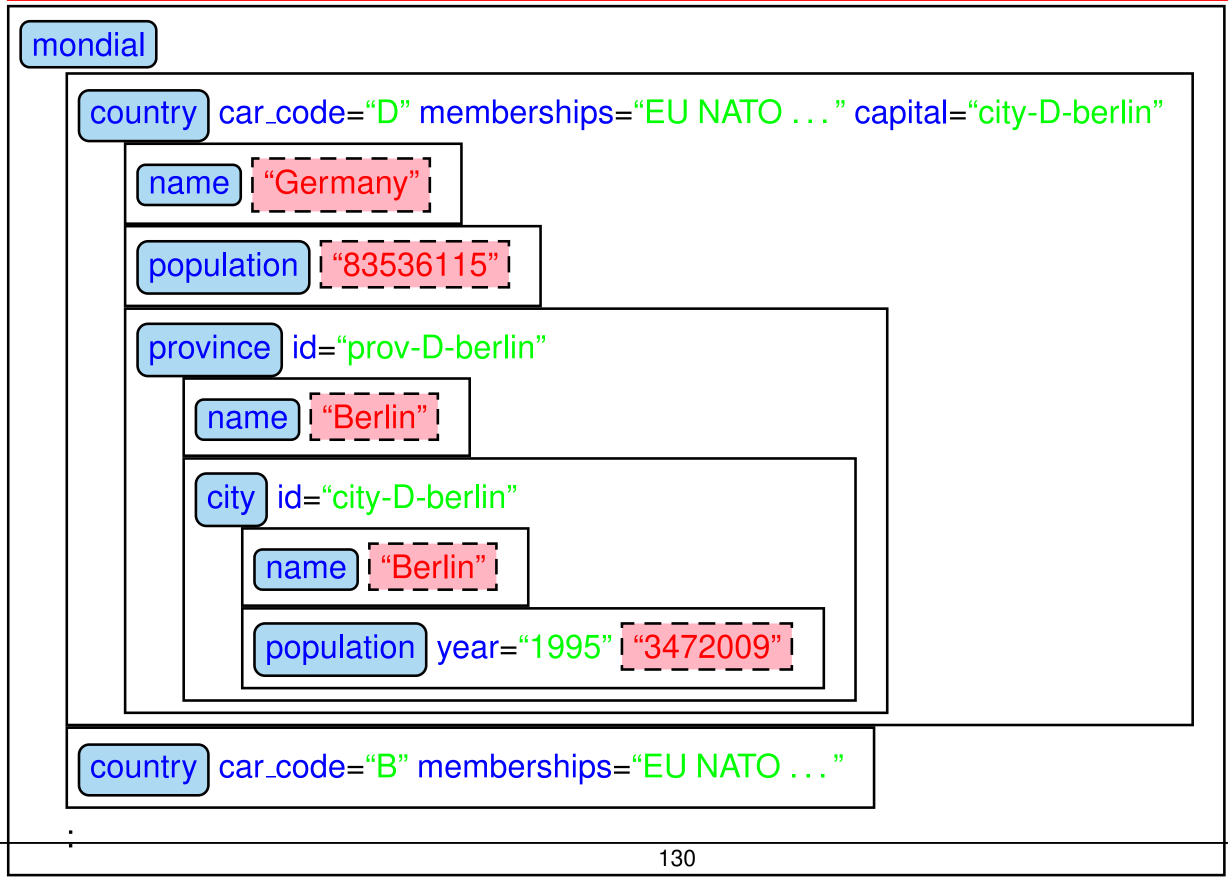
4.1 Grobstruktur des abstrakten XML-Datenmodells

- Für jedes Dokument gibt es einen Dokumentknoten, der das Dokument "ist" und Informationen über das Dokument enthält.
- Das Dokument an sich besteht aus geschachtelten *Elementen* (Baumstruktur),
- davon genau ein *root-Element*, das alle anderen Elemente enthält und einziges Kind des Dokumentknotens ist.
- Elemente besitzen einen Elementtyp (z.B. Mondial, Country, City)
- Element-Inhalt kann aus Text und/oder *Subelementen* bestehen.
  Diese Kindknoten sind geordnet.
- Elemente haben *Attribute*.
  Jeder Attributknoten hat einen Namen und einen Wert (z.B. (car_code, "D")).
  Die Attributknoten sind ungeordnet.
- Leere Elemente haben keinen Inhalt, können aber Attribute haben.
- Ein *Knoten* in einem XML-Dokument ist eine logische Einheit, d.h., ein Element, ein Attribut, oder ein Textknoten.
- Die Struktur kann durch Schemavorgaben eingeschränkt werden.

BEISPIEL: MONDIAL ALS BAUM



BEISPIEL: MONDIAL ALS SCHACHTELSTRUKTUR



Feststellungen

- es existiert eine globale Ordnung (präorder-Tiefensuche-Durchgang) aller Element- und Textknoten, als *Dokumentordnung* bezeichnet.
- echter Text sind nur die Text-Knoten
  Dokumente: wenn man alle Textknoten in Dokumentordnung aneinanderreiht, erhält man eine reine Textversion.
  Aufgabe: Machen Sie sich dies an einem HTML-Dokument klar.
- Elementknoten dienen der Strukturierung (enthalten selbst aber keinen "Wert")
- Attributknoten enthalten Werte, deren Semantik im Detail noch beschrieben wird
  - Attribute, die die Elemente näher beschreiben (z.B. population/@year)
  - IDs und Referenzen darauf
  - können je nach Anwendung unterschiedlich genutzt werden

4.2 XML ASCII-Repräsentation

- Baum-Modell und Schachtelungsmodell dienen als abstrakte Datentypen (siehe später: DOM)

Datenaustausch? Wie "schreibt" man ein XML-Dokument?

- eine relationale DB kann als endliche Menge von Tupeln ausgegeben werden (vgl. relationaler Kalkül)
  country("Germany","D",83536115,356910,"Berlin","Berlin")
  oder
  country(Name: "Germany", Code: "D", Population: 83536115, Area: 356910,
  Capital: "Berlin", CapitalProvince: "Berlin")
- Objektorientierte Datenbanken: OIF (Object Interchange Format)
- OEM-Tripel, F-Logic Frames
- XML?
  Exportieren des Baumes in einem *präorder-Tiefensuche-Durchlauf*.
  Die Knotentypen werden dabei gemäß einer gegebenen Syntax dargestellt:
  ⇒ XML als Repräsentationssprache

XML ALS REPRÄSENTATIONSSPRACHE

- Elemente werden durch
  - öffnende <Country> und
  - schließende *Tags* </Country> begrenzt,
  - dazwischen wird (rekursiv) der *Element-Inhalt* angegeben.
- Element-Inhalt kann aus Text

  <Name> United Nations </Name>
- und *Subelementen* bestehen:
  ```
  <Country>   <City> ... </City>
              <City> ... </City>
  </Country>
  ```
- *Attribute* werden im öffnenden Tag angegeben:

  <Country car_code="D"> . . . </Country>

  Attribute sind dabei immer String-wertig, haben also keine weitere Struktur. Der Unterschied zwischen Wert- und Referenzattributen ist dabei nicht sichtbar, sondern erst durch die DTD gegeben.
- *leere Elemente* haben nur Attribute: <border country="F" length="451"/>

XML ALS REPRÄSENTATIONSSPRACHE: GRAMMATIK

Die so definierte Sprache "XML" kann als EBNF-Grammatik angegeben werden:

Dokument	::=	Element
Element	::=	"<" Elementname ["["Attribut+"]"] ">" Content "</" Elementname ">"
		| "<" Elementname ["["Attribut+"]"] "/>"
Content	::=	(Element | Text)+
Text	::=	*Zeichenfolge*
Attribut	::=	Attributname "='" Attributwert "'"

- statt ' sind auch normale " erlaubt
- Kommentare können mit <!-- ... --> eingefügt werden (wie in HTML).
- Strenge Einhaltung (schliessende und leere Elemente) gefordert.
- Eine als ASCII gegebene XML-Instanz ist *well-formed (wohlgeformt)*, wenn sie diese Regeln erfüllt.

XML-Parser

- Ein *XML-Parser* ist ein Programm, das ein in ASCII-Repräsentation gegebenes Dokument entsprechend der Grammatik parst und ein Ergebnis liefert:
  - Korrektheit: Überprüfen auf Wohlgeformtheit (und ggf. Einhaltung einer DTD)
  - DOM-Parser: Transformieren und Abspeichern der XML-Instanz in ein DOM-Modell (Implementierung des abstrakten Datentyps; siehe später).
  - SAX-Parser: Traversieren des XML-Baumes und Ausgabe einer Folge von "Events", die das Dokument *serialisieren* (siehe später).
- XML-Parser müssen nur wohlgeformte Instanzen akzeptieren.
  - einfache Grammatik, einfacher Parser
  - HTML: fehlertolerante Parser → viel komplizierter
- Jede XML-Anwendung muss einen XML-Parser beinhalten um dieses Eingabeformat verarbeiten zu können.

Anschauen von XML-Dokumenten

- als File im Editor
  - emacs mit xml-mode
  - unter Linux/KDE: kxmleditor
- Browser können XML nicht "interpretieren" (im Gegensatz zu HTML)
- mit "show source" im Browser
- neue Versionen der meisten Browser stellen XML in ASCII-Repräsentation mit Einrückung und auf-/zuklappbaren Elementen dar.
  Z.B. mozilla
- Aber eigentlich ist XML nicht zum Anschauen gedacht:
  → Transformation nach HTML durch XSLT-Stylesheets
  (siehe später)

4.3 Datentypen und Strukturbeschreibung in XML

- Relationales Modell: atomare Datentypen und Tupeltypen
- Objektorientiertes Modell: Literaltypen und Objekttypen, Referenztypen

Datentypen in XML

- Datentypen für Textinhalt
- Datentypen für Attributwerte
- Elementtypen
- etwas unterschiedliche Herangehensweisen in DTD (grob) und XML Schema (fein)

Document Type Description – DTD

- Die Menge der erlaubten Tags sowie die erlaubten Schachtelungen und Attribute werden in der DTD des Dokument(typ)s festgelegt.
- die Idee der DTD kommt aus dem SGML-Bereich
  - erfüllt die Anforderungen an Dokumentstrukturierung
  - genügt nicht so ganz den Anforderungen der Datenbankwelt
    → XML Schema (später)
  - kurz und einfach zu verstehen.
- Die DTD zu einem Dokumenttyp enthält eine Grammatik, die eine Klasse von Dokumenten beschreibt:
  - welche Elemente es in einem Dokument des Type $doctype$ gibt,
  - welche Subelemente sie besitzen (Elementtyp, Reihenfolge, Kardinalität)
  - welche Attribute sie besitzen (Attributname, Art und Kardinalität)
  - außerdem können Entities definiert werden (dienen als Konstanten bzw. Macros)

DATENTYPEN DER DTDS

- Textinhalt: PCDATA – parsed character data
  es obliegt der Anwendung, zwischen Stringdaten und numerischen Daten zu unterscheiden
- Datentypen für Attributwerte:
  - CDATA: einfache Strings
  - NMTOKEN: String ohne Leerzeichen
  - NMTOKENS: mehrere Tokens, durch Leerzeichen getrennt
  - ID: wie NMTOKEN, jeder Wert muss einmalig innerhalb eines Dokuments sein
  - IDREF: wie NMTOKEN, jeder Wert muss in dem selben Dokument als ID vorkommen.
  - IDREFS: dasselbe mehrwertig
- Elementtypen: Strukturdefinition im Stil von regulären Ausdrücken.

DTD: ELEMENTTYP-DEFINITION – STRUKTUR DES ELEMENTINHALTS

<!ELEMENT *elem_name struct_spec*>

- EMPTY: leerer Elementtyp,
- (#PCDATA): Text-only Inhalt
- (*expression*): Ausdruck über Element-Namen, und Kombinatoren (wie bei regulären Ausdrücken)
  - ",": Sequenz,
  - "|": (Exklusiv-)Oder (choice),
  - "*": beliebig oft,
  - "+": mindestens einmal,
  - "?": optional
- (#PCDATA|*elem_name*$_1$|...|*elem_name*$_n$)*
  Mixed Contents, hier kann nur die Art der außer #PCDATA vorkommenden Subelemente eingeschränkt werden; keine Aussage über erlaubte Reihenfolge.
- (ANY): beliebiger Inhalt

Elementtyp-Definitionen: Beispiele

- aus HTML: Images haben nur Attribute und keinen Inhalt
  <!ELEMENT Img (EMPTY) >
- aus Mondial:

  <!ELEMENT Country (Name, Encompassed+, Population*,
  Ethnicgroup*, Religion*, border*,
  (Province+ | City+))>
  <!ELEMENT Name (#PCDATA)>
- für Textdokumente:

  <!ELEMENT Section (Header,
  (Paragraph|Image|Figure|Subsection)+,
  Bibliography?)>

DTD: ATTRIBUT-DEFINITIONEN

- Allgemein: Ein Element darf zu jedem Attributnamen nur einen Attributknoten enthalten.
- Genaueres ist in der DTD festgelegt:

<!ATTLIST *elem_name*
attr_name$_1$ *attr_type*$_1$ *attr_constr*$_1$
: : :
attr_name$_n$ *attr_type*$_n$ *attr_constr*$_n$>

- *attr_type*$_i$: Wert/Referenzattribute und skalar/mehrwertig
  - CDATA: beliebiger Text.
  - NMTOKEN: skalar, Token-Inhalt (Text ohne Spaces).
  - NMTOKENS: mehrwertig, Token-Inhalt.
  - (*const*$_1$| ... |*const*$_k$): skalar, aus dem gegebenen Wertebereich.
  - ID: ausgezeichnetes skalares Attribut, Token-Inhalt, eindeutig im gesamten Dokument.
  - IDREF: skalar, enthält ein Token, das im selben Dokument als Wert eines ID-Attributs auftritt (Referenz).
  - IDREFS: mehrwertiges Referenzattribut.

DTD: Attribut-Definitionen (Forts.)

<!ATTLIST *elem_name*

	$attr\_name_1$	$attr\_type_1$	$attr\_constr_1$
	:	:	:
	$attr\_name_n$	$attr\_type_n$	$attr\_constr_n$>

- $attr\_constr_i$: minimale Kardinalität
  - REQUIRED: Attributwert muß für jedes Element dieses Typs angegeben werden.
  - IMPLIED: Attribut ist optional.
  - *default*: Default-Wert (nichtmonotone Wertvererbung).
  - FIXED *value*: Attribut hat denselben Wert für jedes Element dieses Typs (monotone Wertvererbung).

DTD: Attribut-Definitionen (Beispiele)

```
<!ATTLIST Country
    Code         ID          #REQUIRED
    Capital      IDREF       #REQUIRED
    Memberships  IDREFS      #IMPLIED
    Products     NMTOKENS    #IMPLIED >
<!ATTLIST desert
    id           ID          #REQUIRED
    Type         (sand,rocks,ice) 'sand'
    Climate      NMTOKENS    #FIXED 'dry' >
```

DTD UND XML-INSTANZEN

- Jede DTD definiert eine eigene Markup-Sprache (d.h., eine XML-Applikation (HTML ist z.B. eine solche, Mondial ebenfalls).
- eine XML-Instanz besitzt einen (allem übergeordneten) *Dokumentknoten*, der u.a. Angaben über ihre DTD enthält.
- Das root-Element des Dokuments muß einen im Dokumenttyp definierten Elementtyp haben.
- Eine XML-Instanz ist *valid (gültig)* bzgl. einer DTD, wenn sie die in der DTD angegebenen Bedingungen erfüllt.
- XML-Instanzen können auch ohne DTD existieren (man weiß dann aber eigentlich nicht, was sie bedeuten).

XML-DOKUMENTSTRUKTUR: PROLOG

Der *Prolog* eines XML-Dokumentes in ASCII-Repräsentation enthält zusätzliche Informationen:

- XML-Identifikation (mit optionalen Attributen)

  <? xml version="1.0" encoding="utf-8"?>

  encoding="ISO-8859-1" erlaubt zusätzlich Umlaute.
- Dokumenttyp*deklaration*: Angabe des Dokumenttyps, und wo die Dokumenttyp*definition* zu finden ist
  - <!DOCTYPE *name* {SYSTEM|PUBLIC} *url*>
    SYSTEM *url*: eigener Dokumenttyp,

    <!DOCTYPE Mondial SYSTEM "mondial-2.0.dtd">

    PUBLIC *url*: Standarddokumenttyp (z.B. HTML), oder
  - <!DOCTYPE *name* [*dtd*]>
    mit direkter Angabe einer DTD *im* Dokument.
- dann folgt der eigentliche Dokumentinhalt (d.h., der Wurzelknoten mit dem Rest als Inhalt).

BEISPIEL: MONDIAL

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mondial SYSTEM "mondial-2.0.dtd">
<mondial>
 <country car_code="AL" area="28750" capital="cty-cid-cia-Albania-Tirane"
          memberships="org-BSEC org-CE org-CCC ...">
  <name>Albania</name> <population>3249136</population>
  <encompassed continent="europe" percentage="100"/>
  <ethnicgroups percentage="3">Greeks</ethnicgroups>
  <ethnicgroups percentage="95">Albanian</ethnicgroups>
  <border country="GR" length="282"/> <border country="MK" length="151"/>
  <border country="YU" length="287"/>
  <city id="cty-cid-cia-Albania-Tirane" is_country_cap="yes" country="AL">
   <name>Tirane</name>
   <longitude>10.7</longitude> <latitude>46.2</latitude>
   <population year="87">192000</population>
  </city>
  :
 </country>
 :
</mondial>
```


TOOL: XMLLINT

`xmllint` ist ein einfaches Tool, mit dem man ein Dokument validieren kann (gehört zur libxml2):

- `man xmllint`: listet alle möglichen Befehle
- momentan ist vor allem der folgende Aufruf von Interesse:
  `xmllint -loaddtd -valid -noout mondial-europe-2.0.xml`
  validiert ein XML-Dokument bezüglich der im Prolog angegebenen DTD.

XMLLINT: Ausblick

Viele weitere Dinge kann man ebenfalls ausprobieren:

- zuerst `xmllint -loaddtd -shell mondial-europe-2.0.xml` aufrufen.

Damit befindet man sich mit einer “Navigating Shell” “im” XML-Dokument-Baum:

- `validate`: validiert das Dokument
- `cd xpath-ausdruck`: wechselt in einen Knoten
  (dazu muss der XPath-Ausdruck einen Knoten eindeutig bestimmen)
  `cd //country[@car_code=''D'']`
- `pwd`: gibt den Pfad an, wo man ist
- `cat`: gibt den aktuellen Knoten aus
- `cat xpath-ausdruck`
  `cat .//city/name`
- `du xpath-ausdruck` gibt an, welchen contents der angesprochene Knoten hat
- `dir xpath-ausdruck` gibt Knotentyp und Attribute an

Datenzentrierte vs. Dokumentzentrierte XML-Dokumente

Datenzentrierte XML-Dokumente

- sehr regelmäßige Struktur mit “Datenfeldern”
- wenig Text
- keine natürlich vorgegebene Baumstruktur

Dokumentzentrierte XML-Dokumente

- Baumstruktur mit viel Text (Textinhalte sollen den Dokumenttext ausmachen)
- unregelmäßige Elementstrukturen
- logisches Markup des Dokuments
- Annotationen des Textes durch Elemente/Attribute

Semistrukturierte XML-Dokumente

- kombinieren beides (z.B. Klinik-Informationssysteme)

SUBELEMENTE VS. ATTRIBUTE

Man kann oft zwischen der Repräsentation als Subelement und Attribut wählen.

Dokumentzentriertes XML

- Hängt man den gesamten Textinhalt aneinander, soll der “Dokumenttext” herauskommen
- Elementstrukturen für logisches Markup und Annotationen
- Attribute beinhalten Zusatzinformationen

Datenzentriertes XML

- mehr Freiheiten
- Attribute sind unstrukturiert und können nicht mit weiteren Attribute versehen werden
- Elemente erlauben Struktur und Verfeinerung mit Subelementen und Attributen
- das durch DTDs beschriebene Modell erlaubt die folgende Funktionalität nur für Attribute
  - Verwendung zur Identifizierung
  - Wertebereichseinschränkung
  - Default-Werte

  (Mit XLink und XML Schema kommt man auch hier weiter)

BEISPIELE UND AUFGABEN

- In der Vorlesung dient die MONDIAL-Datenbasis als allgegenwärtiges Beispiel, an der Routineaufgaben durchgeführt werden.
  Siehe `http://user.informatik.uni-goettingen.de/~may/Mondial#XML`.
- Viele W3C-Dokumente basieren auf Beispielen zu einer Literaturdatenbank (Buch, Titel, Autoren, etc).
- jeder Teilnehmer (evtl. gruppenweise) sollte einen *eigenen* Anwendungsbereich auswählen, in dem ein eigenes Beispiel entwickelt und durchgeführt wird.
  - aus dem gewählten Anwendungsfach?
  - Datenbank über Musik-CDs (Titel, CD, Interpret ...)
  - Vorlesungen und Personen der Universität
  - Kalender- und Tagebucheinträge
  - sonstiges

Aufgabe: Definieren Sie eine DTD und erstellen Sie ein kleines XML-Dokument für Ihren gewählten Anwendungsbereich.

AUFGABEN

- Validieren Sie Ihr Beispieldokument mit geeignetem Prolog und interner DTD.
- Legen Sie Ihre DTD öffentlich in Ihr public-Verzeichnis und validieren Sie ein Dokument, das diese DTD als externe DTD referenziert.
- Lassen Sie sich von einem Kollegen dessen DTD+url geben und schreiben Sie selber eine kleine Instanz und validieren diese.

DATENAUSTAUSCH MIT XML

Um Daten auszutauschen (EDI; Electronic Data Interchange) ist eine von allen Beteiligten verwendete gemeinsame DTD erforderlich.

- Automobil-Zulieferer
- Gesundheitswesen
- Finanzen

VORGEHENSWEISE

Der Austausch geschieht üblicherweise in der ASCII-Repräsentation.

- XML-Server machen Dokumente in der ASCII-Repräsentation zugänglich (d.h. als Stream oder als Textfile)
- Anwendungen *parsen* diese Eingabe (linear) und speichern sie intern (DOM oder sonstwie) ab.

4.4 Beispiel: XHTML

- XML-Dokumente, die einer strengen Version der HTML-DTD genügen
- Ziel: Browsing, Publizierung
- DTD auf `http://...`
- Validator auf `http://...`
- Beispiel auf ...

- Nur der Textinhalt wird dargestellt, der Rest beschreibt, *wie* die Darstellung aussieht.
- kein logisches Markup des Dokuments (Sectioning etc), sondern
- nur optisches Markup ("wie wird es dargestellt").

AUFGABE

Gestalten (und validieren) Sie eine einfache Homepage in XHTML, die Sie als `index.html` öffentlich in Ihr public-Verzeichnis legen.

4.5 Sonstiges

Web-Seiten

- Allgemeine Seiten: `www.w3.org/XML`, `www.oasis-open.org/cover`, `www.heise.de/ix/raven/Web/xml`
- Übersicht über Tools: `www.oasis-open.org/cover/publicSW.html`, `www.rpbourret.com/xml/XMLDBLinks.html`, `www.xmlsoftware.com`

4.5.1 Diverses

- Man kann alle Buchstaben in Tag-Namen verwenden
- Text (Attributwerte und Element-Inhalt) kann fast alle Characters enthalten. Westeuropäische Umlaute sind zugelassen, wenn man in der XML-Identifikation encoding=“ISO-8859-1” angibt.
- Kommentare werden mit <!-- ... --> eingeschlossen.
- Innerhalb von XML-Contents kann man mit

  <![CDATA[...]]>

  character-data-Sequenzen (d.h., sie werden nicht geparst, sondern zeichenweise übernommen) schreiben.
  (Wenn man z.B. ein “<”-Zeichen innerhalb des Textinhalts benötigt)

4.5.2 Entities

Entities dienen als Macros oder als Konstanten und werden in der DTD mitdefiniert und sind dann als “&*entityname*;” in der XML-Instanz verwendbar:

<!ENTITY *entity_name replacement_text*>

- Zusätzliche Sonderzeichen, z.B. ç:

  DTD: <!ENTITY ccedilla “&#231”>
  XML: president=“Fran&ccedilla;ois Mitterand”

- Unzulässige Characters können als *Referenzen auf vordefinierte Entities* eingebaut werden:
  < = &lt (less than), > = &gt (greater than)
  & = &amp (ampersand), *space* = , *apostroph* = &apos, *quote* = &quot
  ä = &auml, ..., Ü = &Uuml

  <name>Düsseldorf </name>

- Characters können ebenfalls als Character-References, z.B. &#x20 (Space), &#xD (CR) angegeben werden.

Entities (Forts.)

- für Dinge, die sich häufig ändern einmal eine Konstante definieren:

  DTD: <!ENTITY server “http://www.informatik.uni-goettingen.de”>
  XML: <url> &server;/dbis <url>

- Tipparbeit sparen (Makro):

  DTD: <!ENTITY european
  “<encompassed continent=’europe’>100</encompassed>”>
  XML: <country car_code=“D”>
  <name >Germany</name>
  &european; ...
  </country>

- Anmerkung: man kann einzelne und doppelte Quotes schachteln.

PARAMETER-ENTITIES

Entities, die bereits in der DTD verwendbar sein sollen, muß man als *Parameter-Entities* definieren:

- Tipparbeit sparen (Makro):
  <!ENTITY % namedecl “name CDATA #REQUIRED”>
  <!ATTLIST City
  %namedecl;
  zipcode ID #REQUIRED>

- Enumerate-Typen definieren:
  <!ENTITY % waters “(river|lake|sea)”>
  <!ATTLIST City_located_at
  type %waters #REQUIRED;
  at IDREF #REQUIRED>

Entities aus externen Quellen

Man kann Entities auch aus externen Quellen definieren:

<!ENTITY *entity_name* SYSTEM "url">

um allgemeine Resourcen einbinden, die ihrerseits Entities definieren:

<!ENTITY *entity_name' replacement_text*>

z.B. eine Menge technischer Zeichen, die es sonst nur in LaTeX gibt:

<!ENTITY % isotech SYSTEM
"http://www.schema.net/public-text/ISOtech.pen">
%isotech;

die Referenz %isotech; macht dann alle Symbole, die in der externen Resource definiert sind, verfügbar.

Man kann das auch iterieren, d.h., sich eine Art "style-File" definieren, das alle üblicherweise benötigten externen Resourcen aufruft.

4.5.3 Einbindung von Multimedia

- Für (externe) non-Text-Resourcen muß angegeben werden, welches Programm sie darstellen soll. Dies geschieht über NOTATION-Definitionen:

  <!NOTATION *notation_name* SYSTEM "$programm\_url$">
  <!NOTATION postscript SYSTEM "file:/usr/bin/ghostview">

- Bei der Entity-Definition wird dann definiert, welche Notation für die Entity verwendet werden soll.

  <!ENTITY *entity_name* SYSTEM "url"
  NDATA *notation_name*>
  <!ENTITY manual SYSTEM "file:/.../name.ps"
  NDATA postscript>

- Das *Anwendungsprogramm* muß dann aber die NDATA-Definition und das Entity noch auflösen.

- mit XLink wird später noch ein Mechanismus zur Referenzierung von Resourcen vorgestellt.

4.6 Zusammenfassung und Ausblick

XML: “Grundversion” besteht aus DTD und XML-Dokumenten

- Baum mit zusätzlichen Kreuzreferenzen
- Element-Schachtelungs-Hierarchie
- Ordnung der Subelemente
  - Dokumente: 1., 2., ... Section etc.
  - Datenbanken: meistens nicht relevant
- Attribute
- Referenzen über IDREF/IDREFS
  - Dokumente: im wesentlichen Kreuzreferenzen
  - Datenbanken: unmittelbar Teil der Daten
- XML-Modell ähnlich zum Netzwerkmodell:
  Beziehungen werden in die Struktur des Datenmodells abgebildet
  - die dortigen in eine Host-Sprache eingebetteten expliziten, schrittweisen Navigationsbefehle bietet hier das DOM-API (siehe später)

ANFORDERUNGEN

- Dokumente: logisches Markup (Sectioning etc)
  Darstellung auf Web-Seiten in (X)HTML? – Transformationssprachen
- Datenbanken: Strukturierung eines Datenbestandes;
  viele äquivalente Möglichkeiten
  Anfragesprachen?
  Darstellung auf Web-Seiten in (X)HTML? – Transformationssprachen
- Anwendungsspezifische Formate
  z.B. XHTML: Browsing
  von den Anwendungsprogrammen vorgegebene DTDs
  Web-Services: WSDL, UDDI; CAD; Ontologie-Sprachen; ...
  Transformation zwischen verschiedenen XML-Sprachen
  Anwendungsprogramme müssen XML intern “verstehen”

WEITERE KONZEPTE DER XML-WELT

Erweiterungen:

- Namensräume: Nutzung verschiedener DTDs in einem Datenbestand (siehe Folie **??**)
- APIs: DOM, SAX
- theoretischer Unterbau
- Anfragesprachen: XPath, XML-QL, Quilt, XQuery
- Stylesheets/Transformationssprachen: CSS, DSSSL, XSL
- bessere Schema-Definition: XML Schema
- XML-Intra-Dokument-Handling: XPointer, XLink

4.7 Recall

... nach der Pfingstpause

- XML als abstraktes *Datenmodell*
  - vgl. relationales DM
  - XML jetzt schon weniger abstrakt: Instanzen im Editor erzeugen ("schreiben"), validieren, anzeigen ...
- zu einem DM gehört ... Implementierung? Theorie?
- ... erst noch etwas anderes: *abstrakter Datentyp, Schnittstelle(n)*
  - Konstruktoren, Modifikatoren, Selektoren (vgl. Info I)
- Hier: "zweistufiges Modell"
  - als ADT (Programmierschnittstelle): Document Object Model (DOM): detaillierte Operationen wie in Programmiersprachen (Java, C++) üblich.
  - als Datenbank-Datenmodell (Benutzerschnittstelle; deklarativ): Einlesen (Parser), [Ändern], *Anfragen*.
- Theorie: formale Spezifikation der Semantik der Sprachen, ansonsten zieht man sich auf klassische DB-Theorie zurück.