

# Chapter 4

## XML (Extensible Markup Language)

### Introduction

- SGML *very* expressive and flexible  
HTML very specialized.
  - Summer 1996: John Bosak (Sun Microsystems) initiates the XML Working Group (SGML experts), cooperation with the W3C.  
Development of a subset of SGML that is simpler to implement and to understand  
<http://www.w3.org/XML/>: the homepage for XML at the W3C
- ⇒ XML is a “stripped-down version of SGML”.
- for understanding XML, it is not necessary to understand everything about SGML ...

133

## HTML

let's start the other way round: HTML ... well known, isn't it?

- tags: pairwise opening and closing: `<TABLE> ... </TABLE>`
  - “empty” tags: without closing tag `<BR>`, `<HR>`
  - `<P>` is *in fact* not an empty tag (it should be closed at the end of the paragraph)!
  - attributes: `<TD colspan = “2”> ... </TD>`
  - empty tags with attributes:  
`<IMG SRC=“http://www.informatik.uni-goettingen.de/photo.jpg” ALIGN=“LEFT”>`
  - content of tag structures: `<TD>123456</TD>`
  - nested tag structures: `<TH><B>Name</B></TH>`  
`<A href=“http://www.ifi.informatik.uni-goettingen.de”>`  
`<B>Homepage of the IFI</B></A>`
- ⇒ hierarchical structure
- Entities: `ä = &auml;` `ß = &szlig;`

134

## HTML

- browser must be able to interpret tags  
→ semantics of each tag is fixed for all (?) browsers.
- fixed specifications how tags can be nested  
(described by a DTD (Document Type Definition))

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

- analogously for tables and lists ...
- reality: people do in general not adhere to this structure
  - closing tags are omitted
  - structuring levels are omitted
- parser has to be fault-tolerant and auto-completing

135

## KNOWLEDGE OF HTML FOR XML?

- intuitive idea – but only of the *textual/unicode representation*
- this is *not a data model*
- no query language
- only a very restricted viewpoint:  
HTML is a markup language for browsers  
(note: we don't "see" HTML in the browser, but only what the browser makes out of the HTML).

**Not any more.**

136

## GOALS OF THE DEVELOPMENT OF XML

- XML must be directly usable and transmitted in the internet (unicode-files/streams),
- XML must support a wide range of applications,
- XML must be compatible with SGML,
- XML documents must be human-readable and understandable,
- XML documents must be easy to create,
- it must be easy to write programs that evaluate/process/parse XML documents.

137

## DIFFERENCES BETWEEN XML AND HTML?

- Goal: *not browsing*, but representation/storage of (semistructured) data (cf. SGML)
- SGML allows the definition of new tags according to the application semantics; each SGML application uses its own *semantic tags*.  
These are defined in a DTD (Document Type Definition).
- HTML is *an* SGML application (cf. <HTML> at the beginning of each document </HTML>), that uses the DTD “HTML.dtd”.
- In XML, (nearly) arbitrary tags can be defined and used:  

```
<country> ... </country>  
<city> ... </city>  
<province> ... </province>  
<name> ... </name>
```
- These *elements* represent objects of the application.

138

## XML AS A META-LANGUAGE FOR SPECIALIZED LANGUAGES

- For each application, it can be chosen which “notions” are used as element names etc.:  
⇒ document type definition (DTD)
- the set of allowed element names and their allowed nesting and attributes are defined in the DTD of the document (type).
- the DTD describes the *schema*
- XML is a *meta-language*, each DTD defines an own language
- for an application, either a new DTD can be defined, or an existing DTD can be used  
→ standard-DTDs
- HTML has (as an SGML application) a DTD

139

## EXAMPLE: MONDIAL

```
<mondial>
:
<country code="D" capital="city-D-Berlin" memberships="EU NATO UN ...">
  <name>Germany</name>
  <encompassed continent="europe">100</encompassed>
  <population year="1997">82501000</population>
  <population year="2011">80219695</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"/>
:
```

140

## Example: Mondial (Cont'd)

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

141

## CHARACTERISTICS:

- hierarchical “data model”
- subelements, attributes
- references
- ordering? documents – yes, databases – no

Examples can be found at

<http://dbis.informatik.uni-goettingen.de/Mondial/#XML>

142

## XML AS A DATA MODEL

XML is much more than only the character/unicode representation shown above as known from HTML

(see also introductory talk)

- abstract data model (comparable to the relational DM)
- abstract datatype: DOM (Document Object Model) – see later
- many concepts around XML  
(XML is *not* a programming language!)
  - higher-level declarative query/manipulation language(s)
  - notions of “schema”

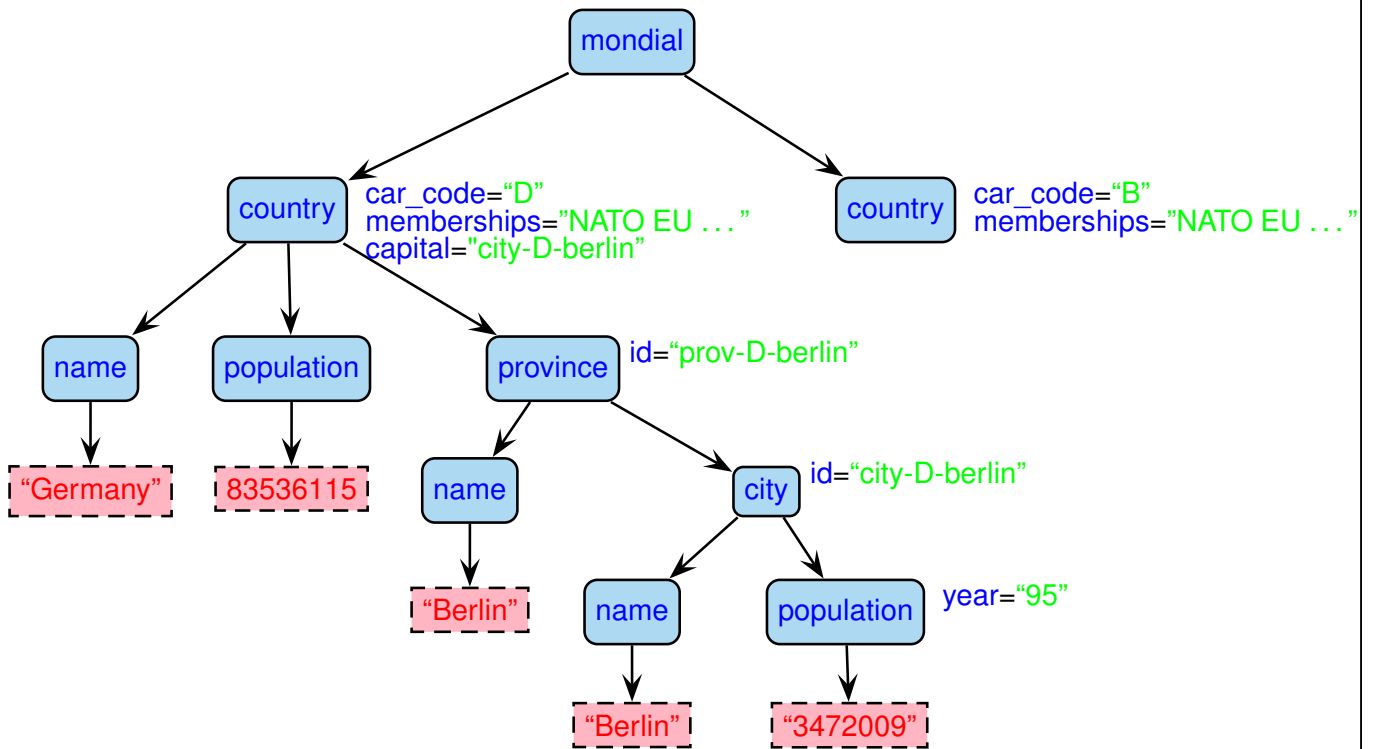
143

### 4.1 Structure of the Abstract XML Data Model (Overview)

- for each document there is a document node which “is” the document, and which contains information about the document (reference to DTD, doctype, encoding etc).
- the document itself consists of nested *elements* (tree structure),
- among these, exactly one *root element* that contains all other elements and which is the only child of the document node.
- elements have an element type (e.g. *Mondial*, *Country*, *City*)
- element content (if not empty) consists of text and/or *subelements*.  
These *child nodes* are ordered.
- elements may have *attributes*.  
Each attribute node has a name and a value (e.g. (*car\_code*, “D”)).  
The attribute nodes are unordered.
- *empty elements* have no content, but can have attributes.
- a *node* in an XML document is a logical unit, i.e., an element, an attribute, or a text node.
- the allowed structure can be restricted by a schema definition.

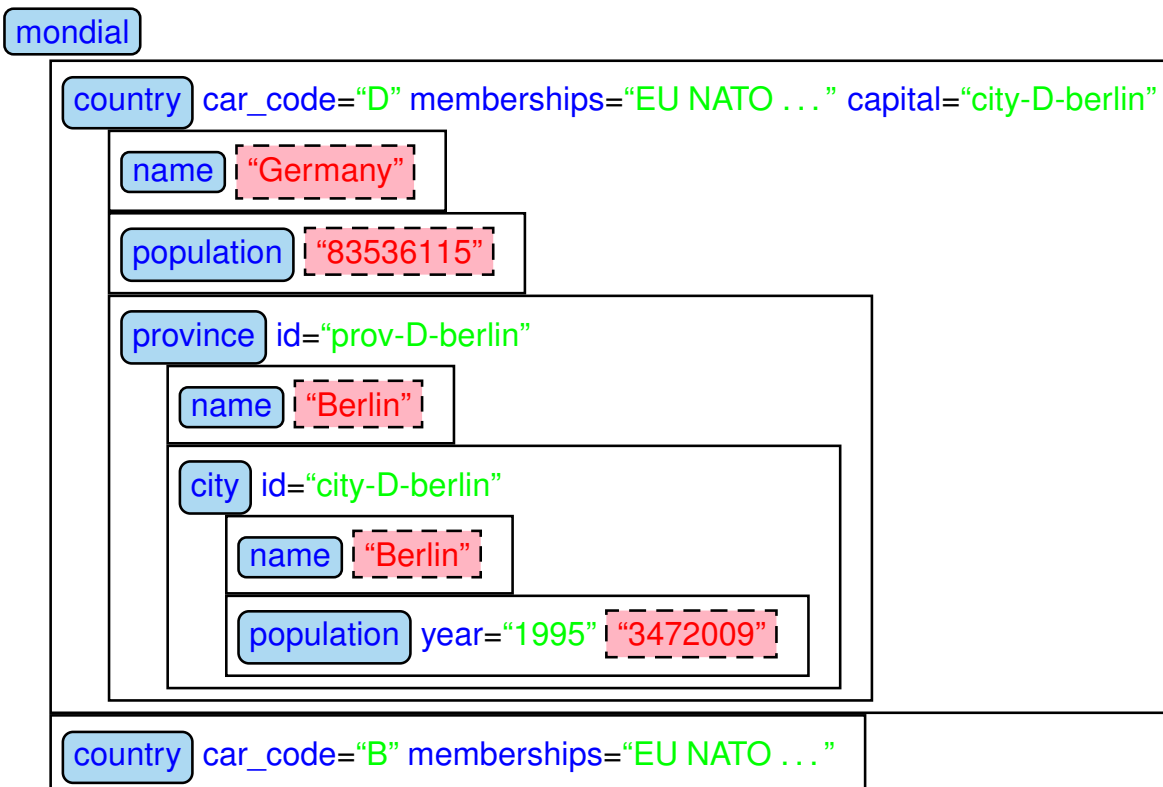
144

## EXAMPLE: MONDIAL AS A TREE



145

## EXAMPLE: MONDIAL AS A NESTED STRUCTURE



146

## OBSERVATIONS

- there is a global order (preorder-depth-first-traversing) of all element- and text nodes, called *document order*.
- actual text is only present in the **text-nodes**  
Documents: if all text is concatenated in document order, a pure text version is obtained.  
Exercise: consider an HTML document.
- element nodes serve for structuring (but do not have a “value” for themselves)
- attribute nodes contain values whose semantics will be described in more detail later
  - attributes that describe the elements in more detail (e.g. `td/@colspan` or `population/@year`)
  - IDs and references to IDs
  - can be used for application-specific needs

147

## 4.2 XML Character Representation

- Tree model and nested model serve as abstract datatypes (see later: DOM)

data exchange? how can an XML document be represented?

- a relational DB can be output as a finite set of tuples (cf. relational calculus)  
positional (cf. Datalog): `country("Germany", "D", 83536115, 356910, "Berlin", "Berlin")`  
or slotted (cf. formal definition of tuples in the relational algebra):  
`country(Name: "Germany", Code: "D", Population: 83536115, Area: 356910, Capital: "Berlin", CapitalProvince: "Berlin")`
- object-oriented databases: OIF (Object Interchange Format)
- OEM-tripels, F-Logic-frames
- XML?  
Exporting the tree in a *preorder-depth-first-traversing*.  
The node types are represented in a specified syntax:  
⇒ XML as a representation language

148



## XML AS A REPRESENTATION LANGUAGE

- elements are limited by
  - opening `<country>` and
  - closing *tags* `</country>`,
  - in-between, the *element content* is output recursively.

- Element content consists of text  
`<name>United Nations</name>`

- and *subelements*:  
`<country> <city> ... </city>`  
`<city> ... </city>`  
`</country>`

- *attributes* are given in the opening tag:  
`<country car_code="D"> ... </country>`

where attribute values are always given as strings, they do not have further structure. The difference between value- and reference attributes is not visible, but is only given by the DTD.

- *empty elements* have only attributes: `<border country="F" length="451"/>`

149

## XML AS A REPRESENTATION LANGUAGE: GRAMMAR

The language “XML” defined as above can be given as an BNF grammar:

```
Document ::= Element
Element  ::= "<" ElementName Attributes ">" Content "</" ElementName ">"
           | "<" ElementName Attributes "/>"
Content  ::= ε | Element Content | Chars Content
Chars    ::= characters including whitespace
Attributes ::= ε | AttributeName "=" Chars "'" Attributes
ElementName, AttributeName ::= character string with some restrictions
```

- note that this grammar does not guarantee that the opening and closing tags match!
- instead of `'`, also the usual `"` are allowed
- strict adherence to these rules (closing and empty elements) is required.
- an XML instance (as a sequence of Unicode/UTF-8/UTF-16 characters) is *well-formed*, if it satisfies these rules.
- “XML parsers” process this input.

150

## XML PARSER

- an *XML parser* is a program that processes an XML document given in Unicode representation according to the XML grammar, and generates a result:
  - **correctness**: check for well-formedness (and adherence to a given DTD)
  - **DOM-parser**: transformation of the XML instance into a DOM model (implementation of the **abstract datatype**; see later).
  - **SAX-parser**: traversing the XML tree and generation of a sequence of “events” that **serialize** the document (see later).
- XML parsers are required to accept only well-formed instances.
  - simple grammar, simple (non-fault-tolerant) parser
  - HTML: fault-tolerant parsers are much more complex (fault tolerance wrt. omitted tags is only possible when the DTD is known)
- each XML application must contain a parser for processing XML instances in Unicode representation as input.

151

## XML PARSING IN THE GENERAL CASE

- ElementName is a separate production and  
Element ::= “<” ElementName Attributes “>” Content “</” ElementName “>”  
| “<” ElementName Attributes “/>”  
does not guarantee matching tags

⇒ not context-free!

- Nevertheless, context-free-style parsing with push-down-automaton *without fixed stack alphabet* possible:
  - for every opening tag, put ElementName on the stack
  - for every closing tag, compare with top of stack, pop stack.

⇒ linear-time parsing

- Exercise: give an automaton for parsing XML and describe the handling of the stack (solution see Slide 179).

152

## VIEWING XML DOCUMENTS?

- as a file in the editor
  - emacs with xml-mode
  - Linux/KDE: kxmleditor
- browser cannot “interpret” XML  
(in contrast to HTML)
- with “show source” in a browser:  
current versions of most browsers show XML in its Unicode representation with indentation and allow to open/close elements/subtrees.
- but, in general, XML is not intended for viewing:  
→ transformation to HTML by XSLT stylesheets  
(see later)

153

## 4.3 Datatypes and Description of Structure for XML

- relational model: atomic data types and tuple types
- object-oriented model: literal types and object types, reference types

### Data Types in XML

- data types for text content
- data types for attribute values
- element types (as “complex objects”)
- somewhat different approaches in DTD (document-oriented, coarse) and XML Schema (database-oriented, fine)

154

## DOCUMENT TYPE DEFINITION – DTD

- the set of allowed tags and their nestings and attributes are specified in the **DTD** of the document (type).
- the idea of the DTD comes from the SGML area
  - meets the requirements for describing document structure
  - does not completely meet the requirements of the database area
    - **XML Schema** (later)
  - simple, and easy to understand.
- the DTD for a document type *doctype* is given by a grammar (context-free; regular expression style) that characterizes a class of documents:
  - **what elements** are allowed in a document of the type *doctype*,
  - **what subelements** they have (element types, order, cardinality)
  - **what attributes** they have (attribute name, type and cardinality)
  - additionally, “entities” can be defined (they serve as constants or macros)

155

## DATA TYPES OF XML AND DTDs

- text content of elements: PCDATA – “parsed character data”; (nearly) arbitrary strings; it is up to the application to distinguish between string data and numerical data; for having “<” in element contents, see Slide 181
- data types for attribute values:
  - CDATA: (Character data) arbitrary strings
  - NMTOKEN: string without blanks; some special chars not allowed
  - NMTOKENS: a list of NMTOKENs, separated by blanks
  - ID: restriction of NMTOKEN, start with [a-zA-Z:\_],  
**each value must be unique in the document**,
  - IDREF: like ID, each value must occur in the same document as an ID value
  - IDREFS: the same, multivalued
  - for the ugly details which characters are (dis)allowed, see  
<https://www.w3.org/TR/2008/REC-xml-20081126/#sec-attribute-types>
- element types: definition of structure in the style of regular expressions.

156

## DTD: ELEMENT TYPE DEFINITION – STRUCTURE OF THE ELEMENT CONTENTS

<!ELEMENT *elem\_name struct\_spec*>

- EMPTY: empty element type,
- (#PCDATA): text-only content
- (*expression*): expression over element names and combinators (same as for regular expressions). Note that the expression must be deterministic.
  - “,”: sequence,
  - “|”: (exclusive-)or (choice),
  - “\*”: arbitrarily often,
  - “+”: at least once,
  - “?”: optional
- (#PCDATA|*elem\_name*<sub>1</sub>|...|*elem\_name*<sub>*n*</sub>)\*  
mixed content, here, only the types of the subelements that are allowed to occur together with #PCDATA can be specified; no statement about order or cardinality.
- ANY: arbitrary content

157

### Element Type Definition: Examples

- from HTML: images have only attributes and no content  
<!ELEMENT img EMPTY >
- from Mondial:  
<!ELEMENT country (name, encompassed+, population\*,  
ethnicgroup\*, religion\*, border\*,  
(province+ | city+))>  
<!ELEMENT name (#PCDATA)>
- for text documents:  
<!ELEMENT Section (Header,  
(Paragraph|Image|Figure|Subsection)+,  
Bibliography?)>
- Element type definitions by **regular expressions**  
⇒ can be checked by **finite state automata**

158

## DTD: ATTRIBUTE DEFINITIONS

- General: an element contains at most one attribute of every attribute name.
- details about allowed attribute names and their types are specified in the DTD.

```
<!ATTLIST elem_name
    attr_name1 attr_type1 attr_constr1
    :
    :
    attr_namen attr_typen attr_constrn>
```

- *attr\_type<sub>i</sub>*: value/reference attribute and scalar/multi-valued
  - CDATA, NMTOKEN, NMTOKENS, ID, IDREF, IDREFS: see Slide 156.
  - (*const<sub>1</sub>* | ... | *const<sub>k</sub>*): scalar, from a given domain.
- *attr\_constr<sub>i</sub>*: minimal cardinality
  - #REQUIRED: attribute must be present for each element of this type.
  - #IMPLIED: attribute is optional.
  - *default*: Default-value (non-monotonic value inheritance).
  - #FIXED *value*: attribute has the same (given) value for each element of this type (monotonic value inheritance).

159

## DTD: ATTRIBUTE-DEFINITIONS (EXAMPLES)

```
<!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' >
```

- when an XML parser reads an XML instance and its DTD, it fills in default and fixed values.

160

## DTD AND XML INSTANCES

- Each DTD defines an own markup language (i.e., an XML application – HTML is one, Mondial is another).
- an XML instance has a *document node* (which is not the root node, but even “superior”) that contains among other things information about the DTD.  
(see next slides ...)
- the root element of the document must be of an element type that is defined in the DTD.
- an XML instance is *valid* wrt. a DTD if it satisfies the structural constraints specified in the DTD.  
Validity can be checked by an extended finite state automaton in linear time.
- XML-instances can exist without a DTD (but then, it is not explicitly specified what their tags “mean”).

161

## XML DOCUMENT STRUCTURE: THE PROLOG

The *prolog* of an XML document in Unicode representation contains additional information about the document (associated with the document node):

- XML declaration (with optional attributes):  
`<?xml version="1.0" encoding="utf-8"?>` allows all characters (chinese, cyrillic, arabian, ...)  
e.g. `encoding="ISO-8859-1"` allows German “Umlauts”.
- document type *declaration*: indication of the document type, and where the document type *definition (DTD)* can be found.
  - `<!DOCTYPE name {SYSTEM own-url | PUBLIC public-id public-url}>`  
*name*: one of the element names given in the DTD  
SYSTEM *own-url*: own document type,  
`<!DOCTYPE mondial SYSTEM “mondial.dtd”>`  
PUBLIC *public-id public-url*: standard document type (e.g. XHTML), or
  - `<!DOCTYPE name [ dtd ]>`  
with DTD directly included *in* the document.
- then follows the document content (i.e., the root node with the document body as its content).

162

## NOTE: DOCUMENT TYPE DECLARATION WITH PUBLIC ID, PUBLIC URL

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- id is a globally agreed string
- url looks like a URL for being accessed through the Web
  - ... maybe this was intended at the beginning.
    - any software that processes a document accesses the DTD at the URL.
  - ⇒ turned out to be a bad idea: billions of accesses to this URL  
([http://www.w3.org/blog/system/2008/02/08/w3c\\_s\\_excessive\\_dtd\\_traffic](http://www.w3.org/blog/system/2008/02/08/w3c_s_excessive_dtd_traffic))
  - ⇒ W3C blocked access to this URL!
  - ⇒ problem for the users who now get unintelligible error messages when using any tools (e.g., creating the DBIS Web pages with XSLT).
- W3C: this URL is to be understood as a URI (Uniform Resource Identifier; in a sense that rather belongs to the Semantic Web area) that only tells the tool that the document “is” XHTML 1.0; *not* that the XHTML DTD should/can be accessed there.
- **technically to be solved by using “XML Catalogs”, cf. Slide 235**

163

## EXAMPLE: MONDIAL

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mondial SYSTEM "mondial.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"/>
    <ethnicgroup percentage="3">Greeks</ethnicgroup>
    <ethnicgroup percentage="95">Albanian</ethnicgroup>
    <border country="GR" length="282"/> <border country="MK" length="151"/>
    <border country="YU" length="287"/>
    <city id="cty-cid-cia-Albania-Tirane" country="AL">
      <name>Tirane</name>
      <latitude>46.2</latitude> <longitude>10.7</longitude>
      <population year="87">192000</population>
    </city>
  :
</country>
:
</mondial>
```

164



## TOOL: XMLLINT

`xmllint` is a simple tool that allows (among other things – see later) to validate a document (belongs to libxml2):

- `man xmllint`: lists all available commands
- currently, we are mainly interested in the following:  
`xmllint -loadtd --noblanks -valid -noout mondial-europe.xml`  
validates an XML document wrt. the DTD given in the prolog  
(`--noblanks` ignores (indentation) whitespaces that would otherwise be seen as mixed content)

165

### XMLLINT: Further Functionality (see later)

XMLLINT can be used to “visit” the document, and to walk through it:

- call `xmllint -loadtd -shell mondial-europe.xml`.

Then, one gets a “navigating shell” “inside” the XML document tree (very similar to navigating in a UNIX directory tree):

- `validate`: validates the document
- `cd xpath-expression`: navigates into a node  
(the XPath expression must uniquely select a single node)  
relativ: `cd country[1]`  
absolut: `cd //country[@car_code="D"]`
- `pwd`: gives the path from the root to the current position
- `cat`: prints the current node
- `cat xpath-expression`  
`cat ../city/name`
- `du xpath-expression` lists the content of the node that is selected by `xpath-expression` (starting from the current node)
- `dir xpath-expression` prints the node type and attributes of the selected node

166

### Example: "Books" from W3C XML Use Cases

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE bib SYSTEM "books.dtd">
<!-- from W3C XML Query Use Cases -->
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
  <book year="1999">
    <title>Economics of ... for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
[see XML-DTD/books.xml]
```

167

### Exercise: Generate a DTD for the above XML

... do it step-by-step, using a validator:

- for all element types:  
  <!ELEMENT *name* ANY>
- declare <!ATTLIST *name* ...> where needed
- validate
- stepwise refinement of content models ...
- ... blackboard demonstration ...
- solution see Slide 175

## DATA-CENTERED VS. DOCUMENT-CENTERED XML DOCUMENTS

### Data-Centered XML Documents

- very regular structure with “data fields”
- only some text
- no naturally induced tree structure

### Document-Centered XML Documents

- tree structure with much text (text content is the text of the document)
- non-regular structure of elements
- logical markup of the documents
- annotations of the text by additional elements/attributes

### Semistructured XML Documents

- combine both (e.g. medical information systems)

169

## SUBELEMENTS VS. ATTRIBUTES

When designing an XML structure, often the choice of representing something as subelement or as attribute is up to the designer.

### Document-Centered XML

- the concatenation of the whole text content should be the “text” of the document
- element structures for logical markup and annotations
- attributes contain additional information *about* the structuring elements.

### Data-Centered XML

- more freedom
- attributes are unstructured and cannot have further attributes
- elements allow for structure and refinement with subelements and attributes
- using DTDs as schema language allows the following functionality only for attributes:
  - usage as identifiers (ID)
  - restrictions of the domain
  - default values(XML Schema allows many more things)

170

## EXAMPLES AND EXERCISES

- The MONDIAL database is used as an example for practical experiments.  
See <http://dbis.informatik.uni-goettingen.de/Mondial#XML>.
- many W3C documents base on examples about a literature database (book, title, authors, etc.).
- each participant (possibly in groups) should choose an *own* application area to set up an own example and to experiment with it.
  - from the chosen branch of study?
  - database of music CDs
  - lectures and persons at the university
  - exams (better than FlexNever?)
  - calendar and diary
  - other ideas ...

Exercise: Define a DTD and generate a small XML document for your chosen application.

171

## EXERCISES

- Validate your example document with a suitable prolog and internal DTD.
- put your DTD publicly in your public-directory and validate a document that references this DTD as an external DTD.
- take a DTD+url from a colleague and write a small instance for the DTD and validate it.
- note: if you do this with an XHTML document and W3Cs XHTML DTD, care for the XML Catalog issue, cf. Slides 163 and 235.

172

## DATA EXCHANGE WITH XML

For *Electronic Data Interchange (EDI)*, a commonly known+used DTD is required

- producers and suppliers in the automobile industry
- health system, medical area
- finance/banking

## PROCEEDING

Usually, XML data is exchanged in its Unicode representation.

- XML-Server make documents in the Unicode representation accessible (i.e., as a stream or as a textfile)
- applications *parse* this input (linear) and store it internally (DOM or anything else).

173

### 4.3.1 Aside: XML Parsing

... side objective of this lecture: show applications and connections of basic concepts of CS:

- XML/DTD: content models are regular expressions
  - ⇒ can be checked by finite state automata
    - design one automaton for each `<!ELEMENT ...>` declaration
    - design a combined automaton for validating documents against a given DTD (recursion requires usage of a return-stack, still linear time)
    - extension to attributes: straightforward (when processing opening tags, dictionary-based)
    - checking for well-formedness and validity in linear time
      - \* with a DOM parser: during generation of the DOM
      - \* with a SAX parser: streaming, on the fly
      - \* using a DOM instance: depth-first traversal
- without a DTD: requires a push-down automaton (remembering opening tags); still linear time
  - checking well-formedness
  - generating a DOM instance, or on-the-fly (SAX)

174

## FINITE STATE AUTOMATA FOR VALIDATION EXAMPLE: BOOKS.DTD

Consider the “books” example:

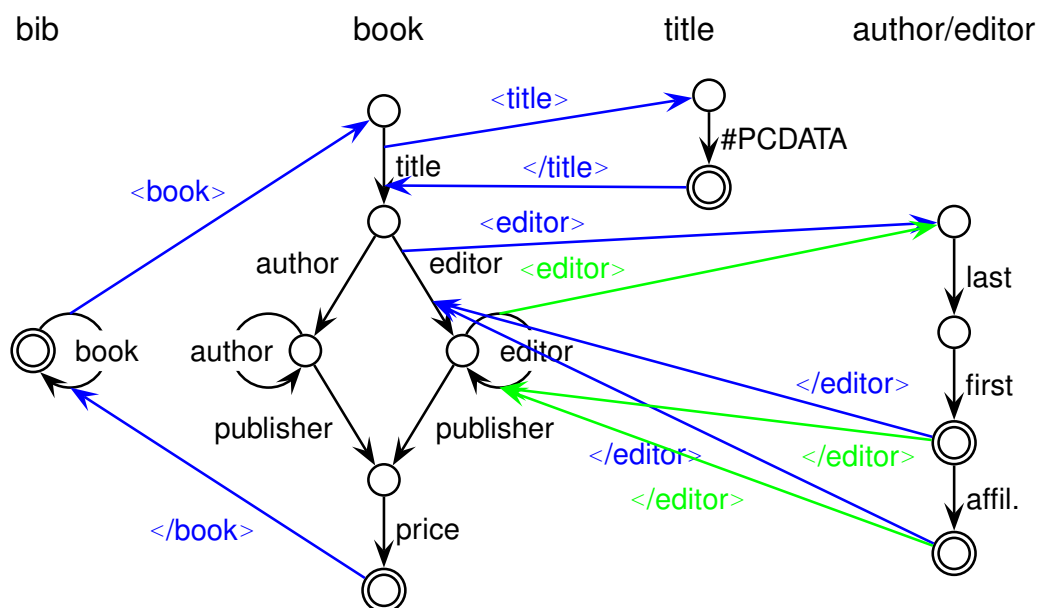
```

<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ | editor+), publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (last, first, affiliation?)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT editor (last, first, affiliation?)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
  
```

175

### Finite State Automata

- individual automata for element content models (recall that the content model must be deterministic)
- combined by nesting (jumping and returning on opening/closing tags)



- author edges use the same author/editor subautomaton → use return-stack

176

## XML Grammar in presence of a DTD

Consider the grammar from Slide 150:

- Element names known from a DTD: context-free grammar (nonterminals in BLUE) (translate regexps in BNF as in the CS I course)

<b>DOCUMENT</b> ::= <b>BIB</b>	
<b>BIB</b> ::= "<bib>" <b>BOOKS</b> "</bib>"	
<b>BOOKS</b> ::= $\epsilon$   "<book year='CHARS'>" <b>TITLE</b> <b>AUTHORS</b> <b>PUBLISHER PRICE</b> "</book>" <b>BOOKS</b>   "<book year='CHARS'>" <b>TITLE</b> <b>EDITORS</b> <b>PUBLISHER PRICE</b> "</book>" <b>BOOKS</b>	regexp: book*  regexp: ...(auth+ edi+)...
<b>TITLE</b> ::= "<title>" <b>CHARS</b> "</title>"	
<b>AUTHORS</b> ::= <b>AUTHOR</b>   <b>AUTHOR</b> <b>AUTHORS</b>	regexp: author+
<b>AUTHOR</b> ::= "<author>" <b>LAST FIRST AFFILIATION</b> "</author>"	
<b>AFFILIATION</b> ::= $\epsilon$   "<affiliation>" <b>CHARS</b> "</affiliation>"	regexp: affiliation?
⋮	
<b>CHARS</b> ::= <i>characters</i>	

177

## XML GRAMMAR IN GENERAL

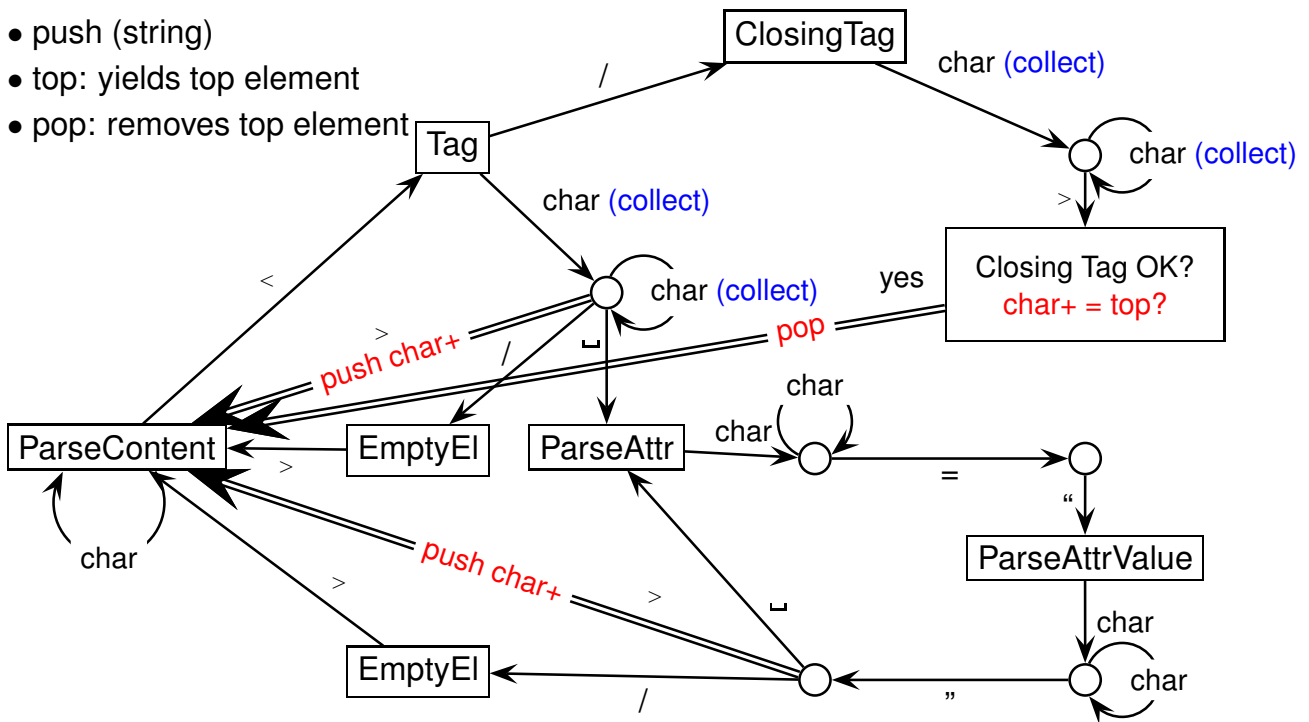
- no DTD present/element names not known:  
Consider the grammar from Slide 150:
- ElementName is a separate production and  
Element ::= "<" ElementName Attributes ">" Content "</" ElementName ">"  
| "<" ElementName Attributes "/>"  
does not guarantee matching tags.
- Nevertheless, context-free-style parsing with push-down-automaton *without fixed stack alphabet* possible:
  - for every opening tag, put ElementName on the stack
  - for every closing tag, compare with top of stack, pop stack.
- Automaton: see next slide.

178

## XML GRAMMAR IN GENERAL

Stack Commands:

- push (string)
- top: yields top element
- pop: removes top element



179

## 4.4 Example: XHTML

- XML documents that adhere to a strict version of the HTML DTD
- Goal: browsing, publishing
- DTD at <http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd> (note that the DTD requires also some entity files)
- Validator at <http://validator.w3.org/>
- Example at ... DBIS Web Pages
- only the text content is shown in the browser, all other content describes *how* the text is presented.
- no logical markup of the documents (sectioning etc), but
- only optical markup ("how is it presented").

### Exercise

Design (and validate) a simple homepage in XHTML, and put it as `index.html` in your public-directory.

180



## 4.5 Miscellaneous about XML

### 4.5.1 Remarks

- all letters are allowed in element names and attribute names
- text (attribute values and element content) can contain nearly all characters. Western european umlauts are allowed if the XML identification contains `encoding="UTF-8"` or `encoding="ISO-8859-1"` etc.

- comments are enclosed in `<!-- ... -->`

- inside XML content,

`<![CDATA[ ... ]]>`

(*character data sequences*) can be included that are not parsed by XML parsers, but which are copied character-by-character.

E.g. in HTML:

```
<li>coloring: <font color="red"> <![CDATA[<font color="blue">XXX</font>]]></font>
prints <font color="blue">XXX</font></li>
```

yields

- coloring: `<font color="blue">XXX</font>` prints `XXX`

181

### 4.5.2 Entities

Entities serve as macros or as constants and are defined in the DTD. They are then accessible as `"&entityname;"` in the XML instance and in the DTD:

```
<!ENTITY entity_name replacement_text>
```

- additional special characters, e.g. ç:

DTD: `<!ENTITY ccedilla "&#231">`

XML: `president="Fran&ccedilla;ois Mitterand"`

- reserved characters can be included as *references to predefined entities*:

`< = &lt;`; (less than), `> = &gt;`; (greater than)

`& = &amp;`; (ampersand), `space = &nbsp;`;, `apostroph = &apos;`;, `quote = &quot;`;

`ä = &auml;`;, ..., `Ü = &Uuml;`;

```
<name>D&uuml;sseldorf </name>
```

- characters can also be given directly as character references, e.g. `&#x20` (space), `&#xD` (CR).

182

## Entities (cont'd)

- global definitions that may change can be defined as constants:

DTD: `<!ENTITY server "http://dbis.informatik.uni-goettingen.de">`

XML(HTML): `<a href="\&server;/Teaching">Teaching</a>`

- macros that are needed frequently:

DTD: `<!ENTITY european`

`"<encompassed continent='europe'>100</encompassed">`

XML: `<country car_code="D">`

`<name>Germany</name>`

`&european; ...`

`</country>`

- note: single and double quotes can be nested.

## PARAMETER ENTITIES

Entities that should be usable only in the DTD are defined as *parameter entities*:

- macros that are needed frequently:

`<!ENTITY % namedecl "name CDATA #REQUIRED">`

`<!ATTLIST city`

`%namedecl;`

`zipcode ID #REQUIRED>`

- define enumeration types:

`<!ENTITY % waters "(river|lake|sea)">`

`<!ATTLIST city_located_at`

`type %waters; #REQUIRED`

`at IDREF #REQUIRED>`

## ENTITIES FROM EXTERNAL SOURCES

Entity “collections” can also be used from external sources as *external entities*:

```
<!ENTITY entity_name SYSTEM "url">
```

is an entity that stands for a remote resource which itself defines a set of entities by

```
<!ENTITY entity_name ' replacement_text'>
```

e.g. a set of technical symbols:

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

the reference %isotech; makes then all symbols accessible that are defined in the external resource.

This can be iterated for defining “style files” that collect a set of external resources that are used by an author.

185

### 4.5.3 Processing Instructions

```
<?pi-target something-without-“?” ?>
```

- Embedded PHP calls are an example for processing instructions

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head><title>PHP test</title></head>  
  <body>  
    <p>This should print hello world (if PHP is activated):  
    <?php echo 'Hello World '; echo date('D, d M Y H:i:s'); ?></p>  
  </body>  
</html>
```

[Filename: XML-DTD/html-php.xml]

- the document validates: `xmllint -noout -valid html-php.xml`
- Browsing: PHP must be activated on the server (cf. Slide 588), files must be named *filename.php*
- Querying: see Slide 303.

186

## 4.5.4 Integration of Multimedia

- for (external) non-text resources, it must be declared which program should be called for showing/processing them. This is done by **NOTATION** declarations:
  - <!NOTATION *notation\_name* SYSTEM “*program\_url*”>
  - <!NOTATION **postscript** SYSTEM “file:/usr/bin/ghostview”>
- the entity definition is then extended by a declaration which notation should be applied on the entity:
  - <!ENTITY *entity\_name* SYSTEM “*url*”  
    NDATA *notation\_name*>
  - <!ENTITY **manual** SYSTEM “file:/../name.ps”  
    NDATA **postscript**>
- the *application program* is then responsible for evaluating the entity and the NDATA definition.
- [XLink provides another mechanism for referencing resources – rarely used].

187

## 4.6 Summary and Outlook

XML: “basic version” consists of DTD and XML documents

- tree with additional cross references
- hierarchy of nested elements
- order of the subelements
  - documents: 1st, 2nd, . . . section etc.
  - databases: order in general not relevant
- attributes
- references via IDREF/IDREFS
  - documents: mainly cross references
  - databases: part of the data (relationships)
- XML model similar to the network data model:  
relationships are mapped into the structure of the data model
  - the basic explicit, stepwise navigation commands of the network data model have an equivalent for XML in the **DOM**-API (see later), but
  - XML also provides a declarative, high-level, set-oriented language.

188

## REQUIREMENTS

- Documents: logical markup (Sectioning etc.)  
presentation on Web pages in (X)HTML? – transformation languages
- databases: structuring of data;  
several equivalent alternatives  
query languages?  
presentation on Web pages in (X)HTML? – transformation languages
- application-specific formats:  
DTDs are induced by the application-programs  
XHTML: browsing  
ant: configuration of automated software build process  
Web-Services: WSDL, UDDI; CAD; ontology languages; ...  
transformation between different XML languages  
application-programs must “understand” XML internally

189

## FURTHER CONCEPTS OF THE XML WORLD

Extensions:

- namespaces: use of different DTDs in a database  
(see Slide 226)
- APIs: DOM, SAX
- theoretical foundations
- query languages: XPath, XML-QL, Quilt, XQuery
- stylesheets/transformation languages: CSS, DSSSL, XSL
- better schema language: XML Schema
- [XML with inter-document handling: XPointer, XLink – rarely used]

190