Valid XML Documents

• an XML document is called a *valid* one, if
  – it is well-formed and
  
  – the prologue of the document contains a document type declaration that again contains or refers to a document type definition (DTD); the XML document corresponds to the content and structure defined within the DTD; or
  
  – the XML document corresponds to the content and structure of an XML schema; the latter one existing as a separate file

What are the benefits of valid XML documents?
DTD Examples

- XHTML strict DTD
  - http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-strict.dtd
- XML specification DTD
- Docbook XML DTD
  - http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd
The Document Type Declaration

- the document type declaration must be inserted between the xml declaration and the root element by using the `<!DOCTYPE>` tag
- the tag `<!DOCTYPE>` is an XML keyword and therefore has to be written in capital letters
- the correct syntax is `<!DOCTYPE name [DTD]>` for internal DTDs and `<!DOCTYPE name SYSTEM "filename">` for external DTDs where `name` has to be exactly the name of the document's root element and `SYSTEM` is an XML keyword too
Example: External / Internal DTD

- files_02/external_dtd_declaration.xml
- files_02/internal_dtd_declaration.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Example for external DTD. -->
<!DOCTYPE movie SYSTEM "movie.dtd">
<!-- It doesn't matter whether there are any comments inserted before or after the DOCTYPE element or not. -->
<movie></movie>

<?xml version="1.0" encoding="UTF-8"?>
<!-- Example for internal DTD. -->
<!DOCTYPE movie [ [ ELEMENT movie ANY ] ]>
<!-- It doesn't matter whether there are any comments inserted before or after the DOCTYPE element or not. -->
<movie></movie>
```
Declaring Element Types

- within the XML document only element types declared in the DTD may be used, otherwise the validation will fail
- the correct syntax is `<!ELEMENT name specification>` where `name` is the element type
- allowed specifications are
  - `EMPTY` – element may not have content
    - `<!ELEMENT name EMPTY>`
  - `ANY` – element may include character data and other elements without limitations or even nothing at all
    - `<!ELEMENT name ANY>`
  - element content
  - mixed content
Element Content I

• defined as a sequence
  - the element contains subordinated elements in a specified, comma-separated sequence; others than the specified sequence will cause a validation error
  - `<!ELEMENT name (subelement1, subelement2, subelement3)>`

• defined as a selection
  - the element may contain one of the given subordinated elements
  - `<!ELEMENT name (subelement1 | subelement2 | subelement3)>`

• modifications
  - '?' – once the previous or no element
  - '+' – one or several of the previous elements
  - '*' – none or several of the previous elements
  - in specified order: `<!ELEMENT name (sub1?, sub2+, sub3*)>`
  - in unspecified order: `<!ELEMENT name (sub1 | sub2 | sub3)+>`
Element Content II

- What does it mean?
  - `<!ELEMENT test (sub1+, sub2, sub3)>`
  - `<!ELEMENT test (sub1, sub2, sub3)>`

- Is that useful?
  - `<!ELEMENT test (sub1 | sub2+ | sub3)>`

- Is that correct?
  - `<!ELEMENT test (sub1* | sub2 | sub3)>`
    
    ... 

    `<test />`

- Specify an element type that shall include in a certain order: sub1, sub2 and one of sub3, sub4 and sub5, where sub3 is optional.
Element Content III

- solution:

  - `<!ELEMENT test (sub1, sub2, (sub3* | sub4 | sub5))>"`


Mixed Content

• only character data
  - `<!ELEMENT name (#PCDATA)>`
  - the XML keyword `#PCDATA` (parsed character data) means, that the XML processor parses the content of the specified element, looking for XML markup code; if you want to include character data that would be interpreted as being markup code, use `<![[CDATA[ ]]>` sections for the element's content

• character data and subordinated elements
  - `<!ELEMENT name (#PCDATA, (sub1 | sub2)?)>`
  - this element must include character data and either the element sub1 or the element sub2 or no subordinated element
Declaring Attributes

• all attributes used within a valid XML document must be declared in the DTD

• the correct syntax is:

  - `<!ATTLIST element_name att_name att_type standard_declaration>`

• the `ATTLIST` contains all attributes for the corresponding element

  - `<!ATTLIST movie`

  category CDATA "horror"

  year CDATA #REQUIRED>`
Attribute Types

- type character data $\rightarrow$ CDATA
- type token
- type enumeration

- see the file files_02/dtd_full.xml for examples
Type Token

- **ID**
  - the attribute must have a unique identifier for each element, the first character of the value may not be a number

- **IDREF**
  - the attribute refers to another element's attribute with type ID

- **IDREFS**
  - same as IDREF, but can refer to several other elements

- **ENTITY**
  - refers to a declared external unparsed entity

- **ENTITIES**
  - can refer to several declared external unparsed entities

- **NMTOKEN**
  - name token (letters, numbers, ., -, _, : (not as first character))

- **NMTOKENS**
  - several name tokens, divided by space
Type Enumeration

- **by defining name tokens**
  - the value of the attribute must be one of the given words within the brackets
  - `<!ATTLIST movie category (horror | fiction | documentation) #REQUIRED>`

- **by defining a NOTATION**
  - a notation must be defined in the DTD; it describes a (file) format or identifies a program that processes a certain format
  - empty elements may not contain a NOTATION
  - other elements may only contain one NOTATION
  - `<!ATTLIST document format NOTATION (HTML | DOC | RTF) #REQUIRED>`
Standard Declarations

- **#REQUIRED**
  - the attribute's value for the corresponding element must be specified, there's no pre-set value
- **#IMPLIED**
  - the attribute's value for the corresponding element may be specified, there's no pre-set value (the value is optional)
- **AttValue**
  - stands for a pre-set value which is used if no value is specified by the user
- **#FIXED AttValue**
  - only a pre-set value may be specified which is used either; this makes only sense as it increases the legibility of the XML document when creating it
Declaring Namespaces

- explicit and standard namespaces for a specific element are declared the following way (files_02/dtd_namespaces.xml):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<-- filename: dtd_namespaces.xml -->
<DOCTYPE collection [
  <ELEMENT collection (item | cd:item)> 
  <ATTLIST collection >
    | xmlns:CDATA #REQUIRED
    | xmlns:cd:CDATA #REQUIRED
  <ELEMENT item (title, author)> 
  <ELEMENT cd:item (cd:title, cd:interpret)> 
  <ELEMENT title (#PCDATA)> 
  <ELEMENT author (#PCDATA)> 
  <ELEMENT cd:title (#PCDATA)> 
  <ELEMENT cd:interpret (#PCDATA)> 
] >

<collection>
  xmlns="http://myhomepage.com/books"
  xmlns:cd="http://myhomepage.com/cds"
  
  <item>
    <title>The Adventures Of Huckleberry Finn</title>
    <author>Mark Twain</author>
  </item>
  
  <cd:item>
    <cd:title>Selling England By The Pound</cd:title>
    <cd:interpret>Genesis</cd:interpret>
  </cd:item>
</collection>
```
Combining DTDs

- it is possible to combine external and internal DTDs
- if there exists an element, attribute, entity or notation which is declared in both internal and external DTD under the same name, only the internal declaration is used
- the correct syntax is:
  
  ```
  <!DOCTYPE name SYSTEM "filename.dtd"
  [
  <!ELEMENT test (#PCDATA)>
  ...
  ]
  >
  ```
• for deactivating a block with markup code temporarily (e.g. when developing) use the XML keyword `<! [IGNORE[ ]]>`

• for activating a block with markup code temporarily (e.g. when developing) use the XML keyword `<! [INCLUDE[ ]]>`