XML Handout and Exercises

1.Handouts

1. *What is XML?*

   XML (*Extensible Markup Language*) is based on SGML (*Standard Generalized Markup Language*) and describes the (data-) structure of documents. In XML, *data, structure and format of documents are handled separate*. The rules are defined in a *Document Type Definition* (DTD) or a XML-Schema.

2. *Structure of documents*

   Like in HTML, the XML concept is based on elements, attributes and entities. The most important thing are the tags (= < >).

3. *Markup*

   Markup can be used for transferation or presentation of text. The single parts of the markup are not part of the data.

4. *Elements*

   Elements are clear identifiable units and build the heart of XML. They have to have a clear marking (*elementname*), which must begin with a letter, an underscore character ("_") or a colon (":") and may additionally contain digits (1, 2, 3, ...), points (".") or dashes ("-"). It must be differenced between capitals and small letters. There is no limit for the length of the name and there are no pre-defined elements. Elements can contain text other further *subelements* (tree-structure). They have to have a *start-tag* and an *end-tag* (except empty elements), between them is the *element content*. To make a sense, they have to be in order (sequence).

5. *Parsing*

   Software has to distinguish between markup and content of documents. The process of this interpretation is called *parsing*. Parser can be used to check documents for errors (well-formed parser) or to pass on a required information.

6. *Document modelling*

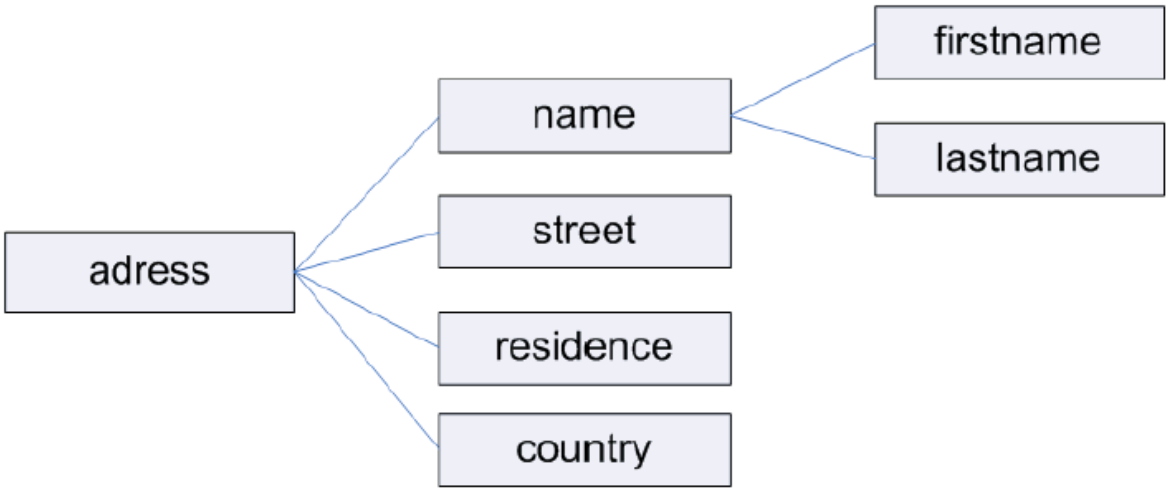
   Documents with similar or identical structures should use the same element types, so they can build groups and a document model can be created. A *document model* is a number of rules, they define which

elements can be used, which are required or optional, what attributes they contain, and so on. One example for such a data file is the *Document Type Definition* (DTD), another is *XML Schema*. A *Validating Parser* can read the DTD (or schema) and afterwards a document (that refers to the DTD), then it reports any discrepancies it may find.

7. *Element hierarchies*

   An element can contain other elements, this is termed an *element hierarchy*. *There is a one element that is the parent of all elements* called *root element* (it has no parents), and if the structure is visualized as a tree, it is the root of the tree.

Figure1.1 An example of an XML tree



8. *Attributes*

   Elements can contain one or more *attributes*. These attributes contain meta-data and they have got a *name* and *value* (`language="en-US"`). Attributes are embedded within the start-tag or an empty tag.

9. *Processing Instructions*

   Some applications need to perform a specific process on the XML document. XML documents can contain instructions for this purpose that is *processing instructions*. These informations are bounded by `<?` and `?>` and the contents are a keyword (significant for the application), a space and the instruction.

10. *XML declaration*

An *XML-declaration* is a processing instruction for important information about an XML-document (for software applications). The keyword (or *target name*) is `xml` (`<?xml ...?>`). If an XML declaration is present, it must occur before everything else in the document.

XML-declarations contain some parameters:

- *Version parameter* (XML-version) `<?xml version="1.0" ...?>`
- *Encoding parameter* (character encoding scheme), `<?xml encoding="UTF-8" ...?>`
- *Standalone parameter* (if an externally defined set of declarations (in a DTD) is used the value is `yes`), `<?xml standalone="yes" ?>`. If the standalone parameter is used, it must follow the other ones.

11. *Markup declaration*

The markup declaration is used to identify or specify important features of the document, to create comments and re-usable document components, to identify the DTD the document refers to and to construct the DTD itself. The delimitation takes place with `<!` and `>`.

12. *Whitespace*

Whitespaces are a small number of miscellaneous characters without visual appearance, namely space, tab and carriage return/line-feed. They assist the tagging of the document. Sometimes whitespaces are significant for the content, sometimes they are insignificant. In some situations, this is a problem. At certain locations in the body of documents, the significance can be set by the author, at other places it is determined by the processing application.

These following rules should be noticed at the handling with whitespace:

- Block and line-in elements must be identified
- Whitespace surrounding block element should be removed
- Line with only declarations or comments should be removed
- Leading and trailing whitespace inside of block elements also
- A sequence of whitespace characters should be reduced to a single space
- Line-end codes in block elements should be converted into a space

13. An example

```
<?xml version="1.0" encoding="UTF-8"?>
<news-article>
 <entry no="1">
  <title>Article no="1"</title>
  <information>Here should been inserted the content of the first article!
  </information>
 </entry>
 <entry no="2">
  <title>Article no="2"</title>
  <information>And here should been inserted the content of the second article!
  </information>
 </entry>
</news-article>
```

What can you say about the `no="1"` from the `entry` tag ? What about `no="1"` from the content of the `title` element ?

2.Design Principles

2.1.Naming things

Naming elements, attributes and controlled vocabulary terms in XML content is one of the most important exercises in design.

- Never assume that it is not important for XML to be readable.
- Always use very explicit and unambiguous element and attribute names.
- Consider using hyphens or underscores in naming rather than "hump case", for example "*first-name*" rather than "*FirstName*".
- Try to group elements logically so that when pretty printed they stand out more cleary rather than having endless runs of sibling elements.

2.2.Principle of core content

- *If you consider the information in question to be part of the essential material that is being expressed or communicated in the XML, put it in an element.* For human-readable documents this generally means the core content that is being communicated to the reader. For machine-oriented records formats this generally means the data that comes directly from the problem domain.
- *If you consider the information to be peripheral or incidental to the main communication, or purely intended to help applications process the main communication, use attributes*. This avoids cluttering up the core content with auxiliary material. For machine-oriented records formats, this generally means application-specific notations on the main data from the problem-domain.
- You might have heard the principle *data goes in elements, metadata in attributes*.

2.3. Principle of structured information

- *If the information is expressed in a structured form, especially if the structure may be extensible, use elements*. Elements are the extensible engine for expressing structure in XML. Almost all XML processing tools are designed around this fact, and if you break down structured information properly into elements, you'll find that your processing tools complement your design, and that you thereby gain productivity and maintainability.
- *If the information is expressed as an atomic token, use attributes.Attributes are designed for expressing simple properties of the information represented in an element. If you work against the basic architecture of XML by shoehorning structured information into attributes you may gain some specious terseness and convenience, but you will probably pay in maintenance costs.*

2.4. Principle of readability

- *If the information is intended to be read and understood by a person, use elements*. In general this guideline places prose in element content.
- *If the information is most readily understood and digested by a machine, use attributes*. In general this guideline means that information tokens that are not natural language go in attributes.

2.5. Principle of element/attribute binding

Use an element if you need its value to be modified by another attribute. XML establishes a very strong conceptual bond between an attribute and the element in which it appears. An attribute provides some property or modification of that particular element. Processing tools for XML tend to follow this concept and it is almost always a terrible idea to have one attribute modify another.

For example, if you are designing a format for a restaurant menu and you include the portion sizes of items on the menu, you may decide that this is not really important to the typical reader of the menu format so you apply the Principle of core content and make it an attribute.

2.6. No substitute for study and experience

XML design is a matter for professionals, and if you want to gain value from XML you should be willing to study XML design principles. Many developers accept that programming code benefits from careful design, but in the case of XML they decide it's OK to just do what seems to work . This is a distinction that I have seen lead to real and painful costs down the road. All it takes for you to learn sound XML design is to pay attention to the issues.

3. Basic XML

1. *Simple questions:*
    - How has to look an element with the name `testelement` and the content "This is our first element!"?
    - Please show (in xml-syntax) the hierachy of the element `book` with the subelements `chapters` and `author`. `author` has the further subelements `name` and `adress`.
    - How looks an element with the name `entry` which has got 2 attributes: `no` with the value `24` and `date` with the value `27.10.2004`?
2. *XML tree structures*

    Analyse the XML file below. Draw a tree diagram of its structure.

```
<?xml      version='1.0'      encoding='ISO-8859-1'
standalone='yes'?>

<publication>

 <newsitem>

  <source>
```

```
   <agency>STT</agency>
   <editor>Miller</editor>
  </source>
  <class>local</class>
  <article>
   <heading>Snowstorms in Lappland</heading>
   <text>snow.html</text>
   <summary>snowx.html</summary>
  </article>
 </newsitem>
</publication>
```

Create the file for example with Notepad and save as *Ex1.xml*. Open the file with Internet Explorer 6. Compare the display with Mozilla.

3. *XML structure*

   Analyse the following XML file. Draw a tree diagram and correct errors in design.

```
<?xml      version='1.0'      encoding='ISO-8859-1'
standalone='yes'?>

<phonebook>

 <company>

  <cname>Microsoft</cname>

  <exchange>09-999000</exchange>

 </company>

 <president>Bill Gates

 <extension>09-9990011</president></extension>

 <secretary>Katharine      Finch      <extension>09-
9990012</secretary></extension>

 <company>

  <cname>Oracle</cname>
```

```
  <exchange>09-888000</exchange>
 </company>
 <president>Larry Ellison
 <extension>09-8880011</president></extension>
 <secretary>Helen Calhoun
 <extension>09-8880012</secretary></extension>
</phonebook>
```

Save your file as *Ex2.xml*. Try also how browsers react to the XML declaration with different character encodings (assuming that your editor is able to save in these encodings):

```
<?xml version='1.0' encoding='UTF-16' standalone='yes'?> <?xml version='1.0' encoding='UTF-8' standalone='yes'?>
```

4.
   - Mark up the following email message to identify its information content:

```
o From: Simon North <north@synopsys.com>
o To: Nick <sintac@xs4all.nl>
o Subject: Hi
o Hi Nick, this is just a quick message
o to say I got the material. Thanks.
```

   - Now mark up the same message to identify its appearance:

```
o From: Simon North <north@synopsys.com>
o To: Nick <sintac@xs4all.nl>
o Subject: Hi
o Hi Nick, this is just a quick message
o to say I got the material. Thanks.
```

   - Compare your two marked-up messages. Give two reasons why one type of markup could be more useful than the other.
5. *XML file, CD information*

Plan an XML description for a music CD. You could use a root element called `<CD>`. Information about the contents should include at least name of the record, artists, songs, publishing year, and music category. You can add more information e.g. about each track (number, duration, name, etc.) Plan first the tree structure of your XML data, and then find some data from the Web. Save as *Ex3a.xml* and test with IE6.

Next, add some information as attributes, like below `<CD serial=B6B41B disc-length='36:55'>` Save as *Ex3b.xml* and test. Fix errors. Add the following comment: `<!-- This CD has more tracks than in this file -->`

6. *XML: a letter template*

   Create a letter structure in XML, including sender and receiver information, date, text body and a signature. Make a document instance with contents. Save your file as *Ex4.xml*.

7. See the following XML file:

```
8. <?xml version="1.0"?>

9. <memo>

10.    <meta>

11.     <from>P. Hermans</from>

12.     <to>S. North</to>

13.     <regarding>deadlines</regarding>

14.    </meta>

15.    <body>

16.     <dear>Simon</dear>

17.     <p>I will <verystrong>not</verystrong> be able
   to finish all chapters before leaving

18.      on 11..holidays.</p>

19.     <p>Please advise what to do.</p>

20.     <close>Paul</close>

21.    </body>

22.   </memo>
```

Using this file:

- Draw the tree structure.
- Indicate in which order the tree is navigated.
- For the last `p` element in the tree, indicate which other nodes still are accessible in the event-driven approach.
- For the `body` element in the tree, indicate which other nodes are still accessible in the tree-based approach.

4. XML structure. DTD's

1. There are two mistakes in the following fragment of code. What are they?

```
2. <![CDATA [This is the hidden &ampmarkup!] ]>
```

3. *A simple address book*

   One wishes design an address book in XML. For each entry of the address book, one wants to store the following information:

   - The person name
   - Home address
   - Telephone number (can be more just one)
   - E-mail address (it can be more e-mail addresses)
   - Birth date

   To do list:

   - Write a DTD for the address book.
   - Write a valid XML file with at least two entries that must illustrate all the possibilities of writting data. Put all in a file and save as *Ex6.xml*.
4. *An advanced address book*

   Make a professional address book improving the addres book created in the previous exercise. For each entry of the address book we want to store the following informations:

   - The person first name and last name.
   - Home address (street, no, zip, county, country)
   - Telephone number (in a complete fashion, like 0049 355 123456) (can be more just one)
   - E-mail address (it can be more e-mail addresses)
   - Company. The following informations must be stored:

        - Company name
        - The list of company web sites. For each web site you must store the following:
            - The name of the site
            - The site Internet address

    Requirements:

    - Write a new version of the DTD and save as Ex7.dtd.
    - Write a valid XML file with at least two entries that must illustrate all the possibilities of writting data. Write the corresponding `DOCTYPE` declaration in the XML file. Save as *Ex7.xml* and test the file with IE6 and Mozilla.

5. *Another address book*

    Looking at the previous exercise, create an XML and DTD file for the following case. An adresse is a data set. The data set includes first name, last name and street. The street has two attributes:

    - ype (big, small), default value is small
    - character, which is not necessary for all streets

6. *Design and implementation*

    Look at the following XML file:

```
<?xml version="1.0" encoding="UTF-8"?>

<cd_review_list>

 <cd_review>

  <author>Joe</author>

  <review>

   I heard all of the latest reports of the
forthcoming which has been since released,

Silenoz said that it was going to be the most
apocalyptic and bombastic CD.

To be honest, I think it's more majestic, symphonic
and grandiose. ...

  </review>

  <author>Dimmu Borgir</author>

  <cd>
```

```
   <author>Dimmu Borgir</author>

   <title>Death Cult Armageddon</title>

  </cd>

 </cd_review>

 <cd_review>

  <author>Alex</author>

  <review>

   Whoa, the Dutch team has changed quite a bit...

From the beginning of the opening "Grip" you realise
that the guitars and bass sound

very modern, hm. Add the harsh voice of George
Oosthoek and you have a

completely un-atmospheric atmosphere, which...

  </review>

  <author>Orphanage</author>

  <cd>

   <author>Orphanage</author>

   <title>Inside</title>

  </cd>

 </cd_review>

</cd_review_list>
```

Requirements:

- Make an XML tree of the elements that will encode a good document model.
- Write a DTD for this document model, put in the file *ex9.dtd* and rewrite the xml file according with this dtd.
- Test the file with IE6 and Mozilla.

7. *Encode structure in DTD*:

   The following fragment of XML code is taken from a simple parts catalog. Add an internal DTD subset to it.

```
<?xml version="1.0"?>
<parts>
 <part>
  <name>Widget</name>
  <catalog.number>11037</catalog.number>
  <price.code>4</price.code>
  <quantity>d</quantity>
 </part>
 <part>
  <name>Bolt</name>
  <catalog.number>11497</catalog.number>
  <thread>w</thread>
  <price.code>1</price.code>
  <quantity>100</quantity>
 </part>
 <part>
  <name>Screw</name>
  <catalog.number>10020</catalog.number>
  <type>countersunk</type>
  <price.code>7</price.code>
  <quantity>c</quantity>
 </part>
</parts>
```

8. *Validating XML data related to a DTD:*

   See the following XML file:

```
<?xml version="1.0"?>
<!DOCTYPE functiondescription [
```

```
 <!NOTATION GIF SYSTEM "" >

 <!ENTITY idhelp782 SYSTEM "idhelp782.txt">

 <!ENTITY idhelp785 SYSTEM "idhelp785.txt">

 <!ENTITY idhelp645 SYSTEM "idhelp645.txt">

 <!ENTITY              buttonleft              SYSTEM
"http://www.navigation.org/button.gif" NDATA GIF>

]>

<function>

 <title>ctime</title>

 <funcsynopsis>

  <funcdef>

   <function>ctime</function>

  </funcdef>

  <paramdef>

   <parameter>time</parameter>

   <parameter role="opt">gmt</parameter>

  </paramdef>

 </funcsynopsis>

 <para>This    function    converts    the    value
<parameter>time</parameter>,

  as returned by <function>time()</function>

  <function>file_mtime()</function>, into a string of
the form

  produced by <function>time_date()</function>.  If
the optional

  argument  <parameter>gmt</parameter>  is  specified
and non-zero,

  the time is returned in <parameter>gmt</parameter>.

  Otherwise, the time is given in the local time zone.

 </para>
```

```
 <para><emphasis                          role="strong"
targetgroup="beginners role="strong">

  Related topics</emphasis>

 </para>

 <para>&ampdoubleclick;    the    <mousebutton>LEFT
&ampbuttonleft;</mousebutton>

  mouse button on a topic

 </para>

 <itemizedlist>

  <listitem>

   <para><ulink
url="&ampidhelp782;"><function>file_mtime()</functio
n>built-in function</ulink>

   </para>

  </listitem>

  <listitem>

   <para><ulink
url="&ampidhelp785;"><function>time()</function>
built-infunction</ulink>

   </para>

  </listitem>

 </itemizedlist>

</function>
```

Using this XML file:

- Detect the errors and predict the error messages that the parser of your choice will generate
- Check the file with that parser
- Correct all problems discovered

9. Create a content-based XML DTD for a simple email message.
10. Create (part of a) DTD to deal with lists. Try creating two versions of the element structures, one using an attribute to describe the type of numbering (for example, bulleted or numbered), and one that uses separate item elements (such as numberlist and bulletlist). Now compare the two and consider the two versions and their

advantages and disadvantages relating to ease of authoring and ease of formatting via a style sheet.

11. Design a DTD for a basic Web home page.
12. Using your home page DTD as the basis, expand the DTD so it can be used to cover a complete Web site. Think in terms of internal DTD subsets and external parameter references.
13. Take any DTD you like and sketch it as a tree diagram (you can use whichever symbols you like). Now set the DTD elements in a table in which the first column contains the root element, the second column contains its children, and so on. Compare the two representations. This should convince you of the importance of modeling a DTD using a clear visual representation.
14. Develop a simple DTD for a basic business letter. It need to be complex just enough for a letterhead, address blocks, and basic text.
15. Create a simple letter in XML using your letter DTD. Extend the XML code in the document and the DTD to include a standard letterhead that contains a GIF-format company logo.