

XML editing and understanding DTDs

Code: xml-edit

Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@tecfa.unige.ch
- Version: 0.9 (modified 17/1/07 by DKS)

Prerequisites

- HTML
- Some idea of what XML is about

Availability (including example files)

url: <http://connections.webster.edu/> COAP 2180 Course Homepage

These slides: Files / teaching_materials / slides / **xml-edit.pdf**

Example files: Files / teaching_materials / example_dtds /



Disclaimer

- There may be typos (sorry) and mistakes (sorry again)
- Please also consult a textbook

Objectives

- Understand the XML formalism: well-formedness and validity
- Being able to edit a moderately complex XML document with an XML editor

1. Table of contents

1. Table of contents	3
2. The XML language	5
2.1 Contents of an XML document	5
Example 2-1:XML data as tree (CALS table example) 6	
2.2 “well-formed” and “valid XML documents”	7
A.“Well-formed” XML documents 7	
Example 2-2:A minimal well-formed XML document 8	
B.XML names and CDATA Sections 8	
C.Valid XML documents 9	
2.3 Name spaces	10
A.Declaring additional vocabularies 10	
Example 2-3:SVG within XHTML 10	
Example 2-4:Xlink 10	
B.Declaring the main vocabulary 11	
Example 2-5:SVG example 11	
C.Namespace URLs 11	
3. DTDs (Document Type Definitions)	12
Example 3-1:A simple DTD 12	
3.1 Using a DTD with an XML document	13
A.Document type declarations 13	
B.Syntax of the DTD declaration in the XML document 14	
C.Some examples of XML documents with DTD declarations: 15	
Example 3-2:Hello XML without DTD 15	
Example 3-3:Hello XML with an internal DTD 15	
Example 3-4:Hello XML with an external DTD 15	
Example 3-5:XML with a public external DTD (RSS 0.91) 15	
3.2 Understanding DTDs by example	16
Example 3-6:Hello text with XML 16	
Example 3-7:A recipe list in XML 17	
Example 3-8:A simple story grammar 19	
Example 3-9:Lone family DTD 20	
Example 3-10:RSS 21	

3.3 Summary syntax of element definitions	23
4. Entities	24
Example 4-1: DTD entities	24
5. Choosing and using an XML Editor	25
5.1 Minimal things your XML editor should be able to do	25
5.2 Additional criteria depending on the kind of XML:	26
5.3 Suggested free editors	27
A. Exchanger XML Lite V3.2:	27
B. XMLmind Standard Edition:	27
C. Alternatives	28
D. About Java	28
6. Next steps	29
6.1 Reading	29
6.2 Next modules	29
7. Homework: mini-project 1	30
7.1 Task	30
7.2 Approximate evaluation grid	31
7.3 Submission format and procedure	32

2. The XML language

- XML = Extensible Markup Language
- XML is a formalism to structure contents with markup.
- An XML language (such as XHTML) is defined with a "grammar", e.g. a DTD
 - Markup usually is formalized by a "grammar" (schema, grammar, language)
 - There are dozens of important XML languages
 - ... and thousands of more "local" applications
- Examples:
 - vector graphics (SVG)
 - web pages (XHTML)
 - web services (SOAP)
 - cooking recipes

2.1 Contents of an XML document

An XML document includes:

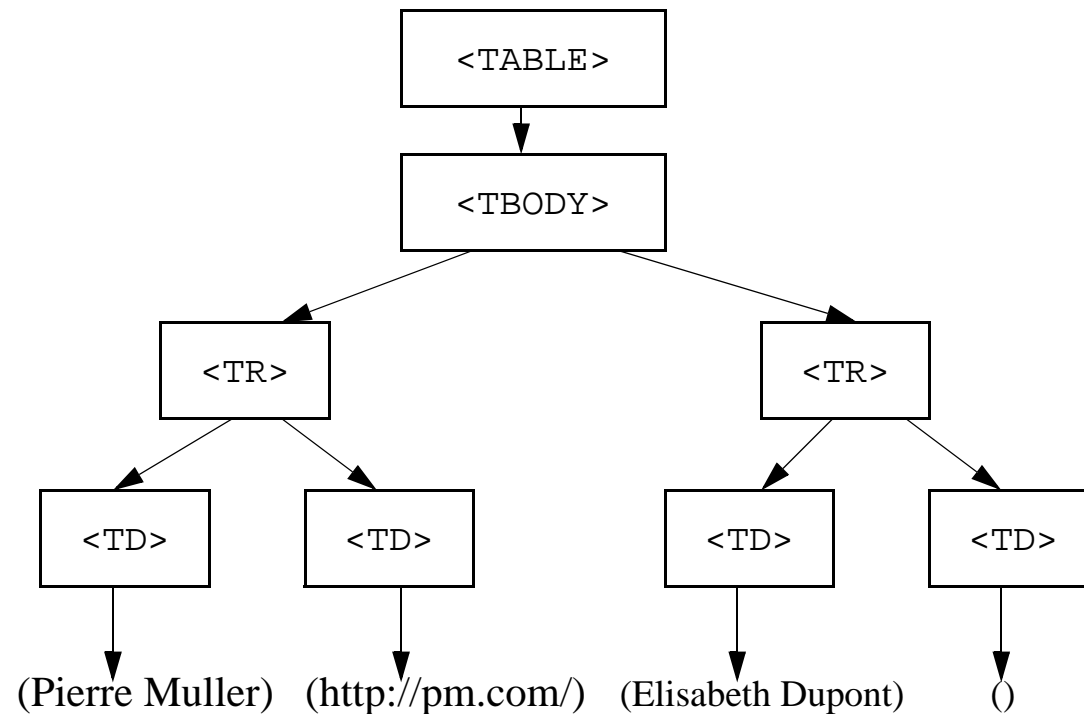
- Processing instructions (at least an XML declaration on top !)
- declarations (in particular a DTD)
- marked up contents (mandatory): elements
- marked up contents (optionally): attributes and entities
- comments: `<!-- -->`

XML documents are trees

- For a computer person, a XML document is a tree (“boxes within boxes”)
- ... and inside a browser (i.e. the DOM) the document is a tree-based data structure

Example 2-1: XML data as tree (CALS table example)

```
<TABLE>
  <TBODY>
    <TR> <TD>Pierre Muller</TD> <TD>http://pm.com/</TD> </TR>
    <TR> <TD>Elisabeth Dupont</TD> <TD></TD> </TR>
  </TBODY>
</TABLE>
```



2.2 “well-formed” and “valid XML documents”

A. “Well-formed” XML documents

- A document must start with an XML declaration (including version number !)

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

- You may specify encoding (default is utf-8) and you have to stick to an encoding !

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

- Structure must be hierarchical:

- start-tags and end-tags must match
- no cross-overs: `<i>......</i> `
- case sensitivity, e.g. "LI" is not "li"

- "EMPTY" tags must use "self-closing" syntax:

- e.g. `
</br>` should be written as `
`, a lonely "`
`" would be illegal

- Attributes must have values and values are quoted:

- e.g. `` or `<person status="employed">`
- e.g. `<input type="radio" checked="checked">`

- A single root element per document

- Root element opens and closes content
- The root element should not appear in any other element

- Special characters (!!): `<`, `&`, `>`, `"`, `'`

- Use `<`; `&`; `>`; `"`; `'`; instead of `<`, `&`, `>`, `"`, `'`
- Applies also to URLs !!

bad: `http://truc.unige.ch/programme?bla&machin`

good: `http://truc.unige.ch/programme?bla&machin`

Example 2-2: A minimal well-formed XML document

```
<?xml version="1.0" ?>
<page updated="jan 2007">
  <title>Hello friend</title>
  <content> Here is some content :) </content>
  <comment> Written by DKS/Tecfa </comment>
</page>
<hello> Hello <important>dear</important> reader ! </hello>
```

- It has an XML declaration on top
- It has a root element (i.e. `page`)
- Elements are nested and tags are closed
- Attribute has quoted value

B. XML names and CDATA Sections

- Names used for elements should start with a letter and only use letters, numbers, the underscore, the hyphen and the period (no other punctuation marks) !
 - Good: `<driversLicenceNo>` `<drivers_licence_no>`
 - Bad: `<driver's_licence_number>` `<driver's_licence_#>` `<drivers licence number>`
- When you want to display data that includes "XMLish" things that should not be interpreted you can use so called CDATA Sections:

```
<example>
  <![CDATA[ (x < y) is an expression
    <svg xmlns="http://www.w3.org/2000/svg">
]]> </example>
```


C. Valid XML documents

Un valid document must be:

1. “well-formed” (see above)
2. conform to a grammar, .e.g.
 - only use tags defined by the grammar
 - respect nesting, ordering and other constraints

Kinds of XML grammars

1. **DTDs** are part of the XML standard
2. **XML Schema** is a more recent W3C standard, used to express stronger constraints
3. **Relax NG** is a OASIS standard (made by well known XML experts and who don't like XML Schema ...)
4. **Schematron** (yet another alternative)

Daniel Schneider likes Relax NG best (it's relatively elegant and powerful)

2.3 Name spaces

- It is possible to use several vocabularies within a document if the markup language says so:
 - E.g. XHTML + SVG + MathML + XLink.
- In order to avoid naming conflicts (e.g. "title" does not means the same thing in XHTML and SVG), one can prefix element and attribute names with a name space.

A. Declaring additional vocabularies

- The "`xmlns:name_space`" attribute allows to introduce a new vocabulary. It tells that all elements or attributes prefixed by "`name_space`" belong to a different vocabulary

Syntax: `xmlns:name_space="URL_name_of_name_space"`

Example 2-3: SVG within XHTML

```
<html xmlns:svg="http://www.w3.org/2000/svg">
  <svg:rect x="50" y="50" rx="5" ry="5" width="200" height="100" ....
```

- `xmlns:svg = "..."` means that `svg:` prefixed elements are part of SVG

Example 2-4: Xlink

- XLink is a language to define links (only works with Firefox-based browsers)

```
<RECIT xmlns:xlink="http://www.w3.org/1999/xlink">
<INFOS>
  <Date>30 octobre 2003 - </Date><Auteur>DKS - </Auteur>
  <A xlink:href="http://jigsaw.w3.org/css-validator/check/referer"
    xlink:type="simple">CSS Validator</A>
</INFOS>
```

B. Declaring the main vocabulary

- The main vocabulary can be introduced by an attribute like:

Syntax: `xmlns="URL_name_of_name_space"`

- Note: some specifications (e.g. SVG) require a name space declaration in any case (even if do not use any other vocabulary) !

Example 2-5: SVG example

```
<svg xmlns="http://www.w3.org/2000/svg">
  <rect x="50" y="50" rx="5" ry="5" width="200" height="100" ....
```

C. Namespace URLs

- URLs that define namespaces are **just names**, there doesn't need to be a real link
- E.g. for your own puporses you can very well make up something like:

```
<account xmlns:pein = "http://joe.miller.com/pein">
  <pein:name>Joe</pein:name>
</account>
```

... and the URL `http://joe.miller.com/pein` doesn't need to exist.

3. DTDs (Document Type Definitions)

DTD grammars are just a set of rules that define:

- a set of elements (tags) and their attributes that can be used;
- how elements can be embedded;
- different sorts of entities (reusable fragments, special characters).
- DTDs can't define what the character data (element contents) and most attribute values look like.

Specification of a markup language

- The most important part is usually the DTD, but in addition other constraints can be added !
- The DTD does not identify the root element !
 - you have to tell the users what elements can be root elements
- Since DTDs can't express data constraints, you may write out additional ones in a specification document
 - e.g. "the value of length attribute is a string composed of a number one of "inch", "em"
<size length="10cm">

Example 3-1: A simple DTD

```
<!ELEMENT page (title, content, comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

- A DTD document is just definition rules nothing else (see later for explanations)

3.1 Using a DTD with an XML document

A. Document type declarations

- A valid XML document includes a declaration that identifies the DTD
- So: The <!DOCTYPE...> declaration is part of the XML file, **not** the DTD

Example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<!DOCTYPE hello SYSTEM "hello.dtd">
```

4 ways of using a DTD

1. No DTD (XML document will just be well-formed)
2. DTD rules are defined inside the XML document
 - We get a "standalone" document (the XML document is self-sufficient)
3. "Private/System" DTDs, the DTD is located on the system (own computer or the Internet)
 - ... that's what **you** are going to use when you write your own DTDs
4. Public DTDs, we use a name for the DTD.
 - means that both your XML editor and user software know the DTD
 - strategy used for common Web DTDs like XHTML, SVG, MathML, etc.

Place

- DTD is declared on top of the file after the XML declaration.
- XML declarations, DTD declaration etc. are part of the prologue

B. Syntax of the DTD declaration in the XML document

- Start of a DTD declaration:

```
<!DOCTYPE .... >
```

- The root element must be specified first

- Remember that DTDs don't know their root element, root is defined in the XML document !
- Note: DTDs must define this root element just like any other element ! (you can have more than one)

```
<!DOCTYPE hello .... >
```

- Syntax for internal DTDs (only !)

- DTD rules are inserted between brackets [...]

```
<!DOCTYPE hello [  
    <!ELEMENT hello (#PCDATA)>  
]>
```

- Syntax to define "private" external DTDs:

- DTD is identified by the URL after the "**SYSTEM**" keyword

```
<!DOCTYPE hello SYSTEM "hello.dtd">
```

- Syntax for public DTDs:

- after the "**PUBLIC**" keyword you have to specify an official name and a backup URL that a validator could use.

```
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"  
    "http://my.netscape.com/publish/formats/rss-0.91.dtd">
```

Recall

- The DTD file itself does not contain any DTD declaration, just rules !!

C. Some examples of XML documents with DTD declarations:

Example 3-2: Hello XML without DTD

```
<?xml version="1.0" standalone="yes"?>
<hello> Hello XML et hello cher lecteur ! </hello>
```

Example 3-3: Hello XML with an internal DTD

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE hello [
  <!ELEMENT hello (#PCDATA)>
]>
<hello> Hello XML et hello chère lectrice ! </hello>
```

Example 3-4: Hello XML with an external DTD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE hello SYSTEM "hello.dtd">
<hello> Hello XèMèLè et hello cher lectrice ! </hello>
```

- That's what you should with your own home-made DTDs

Example 3-5: XML with a public external DTD (RSS 0.91)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
  "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
<channel> ..... </channel>
</rss>
```

3.2 Understanding DTDs by example

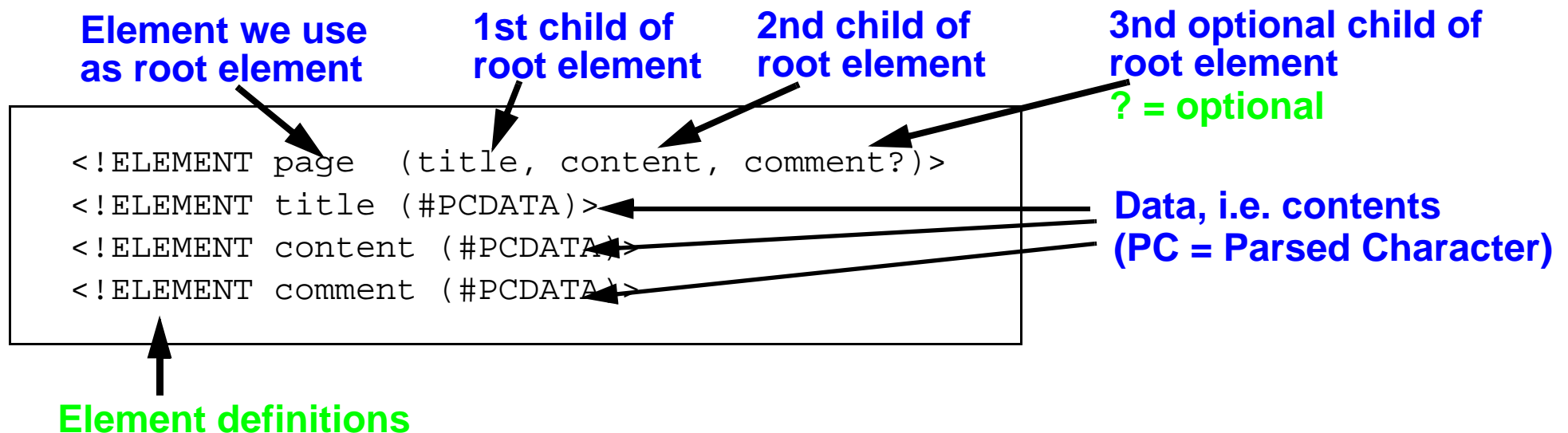
Example 3-6: Hello text with XML

url: <http://tecfa.unige.ch/guides/xml/examples/simple/>

A simple XML document of type <page>

```
<page>
  <title>Hello friend</title>
  <content>
    Here is some content :)
  </content>
  <comment>
    Written by DKS/Tecfa, adapted from S.M./the Cocoon samples
  </comment>
</page>
```

A DTD that would validate the document



Example 3-7: A recipe list in XML

- Source: Introduction to XML by Jay Greenspan (now dead URL)

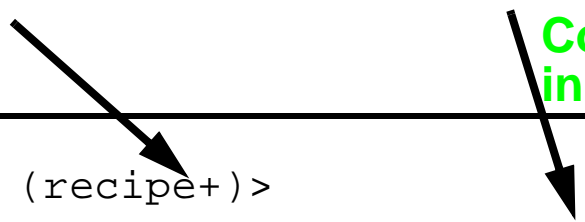
```
<?xml version="1.0"?>
<!DOCTYPE list SYSTEM "simple_recipe.dtd">
<list>
  <recipe>
    <author>Carol Schmidt</author>
    <recipe_name>Chocolate Chip Bars</recipe_name>
    <meal>Dinner
      <course>Dessert</course>
    </meal>
    <ingredients>
      <item>2/3 C butter</item>      <item>2 C brown sugar</item>
      <item>1 tsp vanilla</item>    <item>1 3/4 C unsifted all-purpose flour</item>
      <item>1 1/2 tsp baking powder</item>
      <item>1/2 tsp salt</item>      <item>3 eggs</item>
      <item>1/2 C chopped nuts</item>
      <item>2 cups (12-oz pkg.) semi-sweet choc. chips</item>
    </ingredients>
    <directions>
      Preheat oven to 350 degrees. Melt butter; combine with brown sugar and vanilla in large
      mixing bowl. Set aside to cool. Combine flour, baking powder, and salt; set aside. Add
      eggs to cooled sugar mixture; beat well. Stir in reserved dry ingredients, nuts, and
      chips.
      Spread in greased 13-by-9-inch pan. Bake for 25 to 30 minutes until golden brown; cool.
      Cut into squares.
    </directions>
  </recipe>
</list>
```

Contents of the DTD (simple_recipe.dtd)

The list element (root)
must contain one or more
recipe elements
+ = at least one

Mandatory children (subelements)
of the recipe element

**Comma separated elements must appear
in the same order !**



```
<!ELEMENT list (recipe+)>
<!ELEMENT recipe (author, recipe_name, meal, ingredients, directions)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT recipe_name (#PCDATA)>
<!ELEMENT meal (#PCDATA)>
<!ELEMENT ingredients (item+)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT directions (#PCDATA)>
```

Example 3-8: A simple story grammar

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!-- DTD to write simple stories - VERSION 1.0 1/2007
```

```
    Made by Daniel K. Schneider / TECFA / University of Geneva -->
```

Comment

```
<!ELEMENT STORY (title, context, problem, goal, THREADS, moral, INFOS)>
```

```
<!ATTLIST STORY xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
```

```
<!ELEMENT THREADS (EPISODE+)>
```

```
<!ELEMENT EPISODE (subgoal, ATTEMPT+, result) >
```

```
<!ELEMENT ATTEMPT (action | EPISODE)*>
```

```
<!ELEMENT INFOS ( ( date | author | a )* )>
```

```
<!ELEMENT title (#PCDATA) >
```

```
<!ELEMENT context (#PCDATA) >
```

```
<!ELEMENT problem (#PCDATA) >
```

```
<!ELEMENT goal (#PCDATA) >
```

```
<!ELEMENT subgoal (#PCDATA) >
```

```
<!ELEMENT result (#PCDATA) >
```

```
<!ELEMENT moral (#PCDATA) >
```

```
<!ELEMENT action (#PCDATA) >
```

```
<!ELEMENT date (#PCDATA) >
```

```
<!ELEMENT author (#PCDATA) >
```

```
<!ELEMENT a (#PCDATA)>
```

```
<!ATTLIST a
```

```
    xlink:href CDATA #REQUIRED
```

```
    xlink:type CDATA #FIXED "simple" >
```

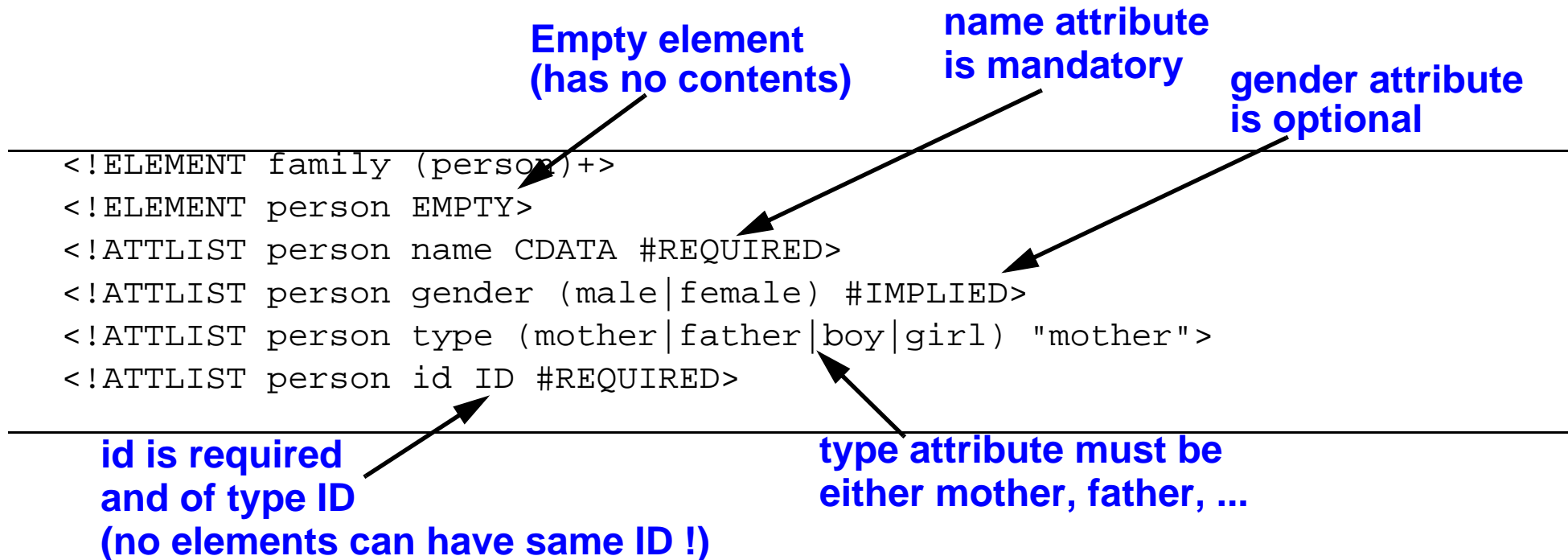
Choose one

| = or

**Choose as
many as
you like
in random
order**

Example 3-9: Lone family DTD

family.dtd



A valid XML file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE family SYSTEM "family.dtd">
<family>
  <person name="Joe Miller" gender="male"
    type="father" id="123.456.789"/>
  <person name="Josette Miller" gender="female"
    type="girl" id="123.456.987"/>
</family>
```

Example 3-10: RSS

- There are several RSS standards. RSS 0.91 is Netscape's original (still being used)

```
<!ELEMENT rss (channel)>
<!ATTLIST rss version CDATA #REQUIRED> <!-- must be "0.91"> -->
<!ELEMENT channel (title | description | link | language | item+ | rating? | image? |
textInput? | copyright? | pubDate? | lastBuildDate? | docs? | managingEditor? |
webMaster? | skipHours? | skipDays?)*>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT link (#PCDATA)>
<!ELEMENT image (title | url | link | width? | height? | description?)*>
<!ELEMENT url (#PCDATA)>
<!ELEMENT item (title | link | description)*>
<!ELEMENT textinput (title | description | name | link)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rating (#PCDATA)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT width (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT copyright (#PCDATA)>
<!ELEMENT pubDate (#PCDATA)>
<!ELEMENT lastBuildDate (#PCDATA)>
<!ELEMENT docs (#PCDATA)>
<!ELEMENT managingEditor (#PCDATA)>
<!ELEMENT webMaster (#PCDATA)>
<!ELEMENT hour (#PCDATA)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT skipHours (hour+)>
<!ELEMENT skipDays (day+)>
```

Possible XML document for RSS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE rss SYSTEM "rss-0.91.dtd">
<rss version="0.91">
  <channel>
    <title>Webster University</title>
    <description>Home Page of Webster University</description>
    <link>http://www.webster.edu</link>
    <item>
      <title>Webster Univ. Geneva</title>
      <description>Home page of Webster University Geneva</description>
      <link>http://www.webster.ch</link>
    </item>
    <item>
      <title>http://www.course.com/</title>
      <description>You can find Thomson text-books materials (exercise data) on this web
site</description>
      <link>http://www.course.com/</link>
    </item>
  </channel>
</rss>
```

3.3 Summary syntax of element definitions

- We will come back to this when we will learn how to write our own DTDs (don't worry too much about unexplained details)

syntax element	short explanation	Example
,	• order of elements	<!ELEMENT Name (First, Middle, Last)>
?	• optional element	MiddleName?
+	• at least one element	movie+
*	• zero or more elements	item*
	• pick one (or operator)	economics law
()	• grouping construct	(A,B,C)

4. Entities

- Most professional DTDs use entities.
- Entities are just symbols that contain some information which substitutes when the symbol is used ...
- 2 kinds: XML entities and DTD entities

Example 4-1: DTD entities

- Some more complex DTD use the same structures all over. Instead of typing these several times one can use a ENTITY construction like this:

```
<!ENTITY % Content "(Para | List | Listing)*">
```

Later in the DTD we then can have Element definitions like this:

```
<!ELEMENT Intro (Title, %Content; ) >  
<!ELEMENT Goal (Title, %Content; ) >
```

The computer will then simply translate these into:

```
<!ELEMENT Intro (Title, (Para | List | Listing)*) >  
<!ELEMENT Goal (Title, (Para | List | Listing)* ) >
```

... think of these entities as shortcuts.

5. Choosing and using an XML Editor

- There are lots of XML editors and there is no easy choice !
- Depending on your needs you may choose a different editor:
 - To edit strongly structured data (i.e. data-centric XML) a sort of "tree" or "boxed" view is practical
 - To edit text-centric data (e.g. an article) you either want a text-processor like tool or a structure editor.
- Really good XML editors cost a lot ...

Here is my own little comparison of XML editors:

url: http://edutechwiki.unige.ch/en/XML_editor

5.1 Minimal things your XML editor should be able to do

- Check for XML well-formedness
- Check for validity against several kinds of XML grammars (DTD, Relax NG, XML Schema)
- Highlight errors (of all sorts)
- Suggest available XML tags (in a given context). Also clearly show which ones are mandatory and which ones are optional, and display them in the right order.
- Allow the user to move/split/join elements in a more or less ergonomic way (although it is admitted that these operations need some training)
- Include support for XSLT and XQuery (However, if you have installation skills you can easily compensate lack of support by installing a processor like Saxon)

5.2 Additional criteria depending on the kind of XML:

For data-centric XML:

- Allow viewing and editing of XML documents in a tree view or boxed view (or both together)
- Provide a context-dependent choice of XML tags and attributes (DTD/XSD awareness)

For text-centric XML:

- Allow editing of XML documents in a structure view
- Allow editing of XML documents in somewhat WYSIWYG view. Such a view can be based on an associated CSS (most common solution) or XSLFO (I am dreaming here) or use some proprietary format (which is not very practical for casual users!). Also allow users to switch on/off tags or element boundary markers.
- Provide a context-dependent choice of XML tags and attributes (DTD/XSD awareness). The user should be able to right-click within the XML text and not in some distant tree representation.
- Automatically insert all mandatory sub-elements when an element is created.
- Automatically complete XML Tags when working without a DTD or other schema.
- Indent properly (and assist users to indent single lines as well as the whole document)

5.3 Suggested free editors

A. Exchanger XML Lite V3.2:

url: <http://www.freexmleditor.com/>

- I suggest to try this editor first, try the other one if you are unhappy with it or if you plan to edit "data-centric" XML documents.

Hints for editing

- To insert an element or attribute:
 - In the contents window press Ctrl-T to insert an element.
 - Pressing "<" in the editing window gives more options and you can do it in any place.
 - To insert an attribute, position the cursor after the element name and press the space bar
- Alternatively (and better if you don't know your DTD): Select the Helper pane to the left. Then (in the editing window) click on the element tag you wish to edit or put your cursor in a location between child elements. The helper pane will then display the structure of the current parent element and list available elements on which you can click to insert.

B. XMLmind Standard Edition:

url: <http://www.xmlmind.com/xmleditor/download.shtml>

Hints for editing

- Element manipulation is through the "tree view"
- After selecting an element
 - you can insert elements either by selecting (tiny) before/after/within buttons in the top right elements

pane

- or use shortcuts: (ctrl-h = insert before, ctrl-i = insert within, ctrl-j = insert after). Same principle for the attributes pane.

C. Alternatives

- Firstly, any XML editor is difficult to learn (because XML editing is not so easy). So make an effort to learn the interface, e.g. read the help !
- Programmers also may consider using a programmer's editor. However make sure:
 - that there is an XML plugin
 - that the editor is "DTD aware" (can show elements to insert in a given context)
 - that it can validate.... otherwise forget it !!

D. About Java

- Most XML editors are written in Java and rely on the "Java RunTime engine".
- Both websites give you a choice: Download an editor with or without Java. If you don't have Java installed on your own PC, I suggest taking it **first** from:

<http://www.java.com/> ... and always download the "no java vm" versions

- To test if you have java, open a command terminal and type "Java".
- To open a command terminal: Start Menu -> Execute and then type "cmd".

6. Next steps

6.1 Reading

- These slides may not be enough to understand, so please read:
Carey (pp. 1-21, pp. 28-39)
Optional: 1 or 2 case problems

6.2 Next modules

Module 2

- Display XML data with CSS in a web browser

Module 3

- Learning how to write a DTD
- Some more details about DTDs (we didn't cover everything)

7. Homework: mini-project 1

Due: Monday jan 22 16:00h

7.1 Task

Edit an XML document with the suggested DTDs below

- Respect the semantics of the elements and the attributes
- Validate your document
- Try to use as many different elements as you can (if appropriate)
- Follow additional directions for each suggested DTD
- Warning: Grab these slides and the example files (ALLFILES.zip) now !

You can choose among these DTDs available at <http://connections.webster.edu/>:

DTD (difficulty)	Purpose	file name	Additional directions
Recipe DTD (easy)	Write simple recipes	recipe.dtd	Use all tags. Write at least one recipe. Make sure that there is enough information to really use it.
Recipe Markup Language (medium)	Write complex recipes	recipeml.dtd	As above, but only use appropriate tags. Hint: find the website of its creator

DTD (difficulty)	Purpose	file name	Additional directions
RSS 0.92 (medium)	News syndication (usually machine generated)	rss-0-92.dtd	Use enough tags to display this in an aggregator. Enter at least 4 URLs. Hint: look at a RSS news feed first !
Simple Docbook (hard)	Write "real" articles	sdocbook.dtd	Do not use all tags, only the needed ones. Copy/paste from a text you already have.
StepbyStep (medium)	Write "how-to" instructions	stepbystep03.dtd	Make up a good "how-to problem". Only use tags you need..
Story grammar (medium)	Write simple fairy tales	story-grammar.dtd	Write a nice fairy tale. Doesn't need to be your own.

7.2 Approximate evaluation grid

Minimal work required:	Probable grade
Wellformed (but not valid) document using the DTD's elements	D
Valid document	C
Extra features:	Probable bonus (depends on quality)
Inserted comments <!-- ... --> in the XML and/or the DTD	+
You produce some interesting content	+ .. ++
You respect the semantics of the DTD	+

Write a 1-2 page report that discusses the architecture of the DTD and your opinion of it, e.g.

- describe architecture of the DTD (without going into detailed description of every element !)
- discuss what you would like to improve, what you liked/disliked, your difficulties, etc.

+++

Examples:

- To get a B+: Firstly produce a valid document, then (a) either write a nice report or (b) you document XML/DTD code and produce a nicely filled-in XML document
- To get an A: do all of the above very well

7.3 Submission format and procedure

I require a double submission:

1. Paper copies to be turned in a start of Monday lesson. Don't forget to write your name on it.
2. Electronic copies: <http://connections.webster.edu/>
Go to the COAP 2180 course. Then **Files / homework/project_1**.
Please make sure to name files according to the following rules:

File name	Example	when ?
your_name.xml	vasta.xml	This is a mandatory file !
xxx_your_name.dtd	vasta-sdocbook.dtd	only if you add comments to the DTD
your_name.{doc pdf html}	vasta.pdf	only if you decide to write one