

# Introduction to XML

Code: xml-intro-edit

## Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@tecfa.unige.ch
- Version: 0.3 (modified 22/4/07 by DKS)

## Prerequisites

- some XHTML
- some CSS

## Objectives

- Understand the purpose of XML
- Understand the role of XML in the W3C framework
- Understand the XML formalism: well-formedness and validity
- Being able to edit a moderately complex XML document with an XML editor
- Adapt a CSS stylesheet

***Chapters or sections marked "(optional)" will not be tested in the exam***

## Disclaimer

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

## Source:

These slides are a summary/simplification of my COAP 2180 slides

# Contents

1. Prerequisite - XHTML	6
Example 1-1:XHTML Page 6	
2. Goal and scope of XML	7
2.1 History	7
2.2 Two ways to look at XML	8
2.3 XML documents on the Web	9
2.4 Production of web and print contents	10
3. The XML language	11
3.1 Contents of an XML document	11
Example 3-1:XML data as tree (XHTML table example) 12	
3.2 “well-formed” and “valid XML documents”	13
A.“Well-formed” XML documents 13	
Example 3-2:A minimal well-formed XML document 14	
B.XML names and CDATA Sections 14	
C.Valid XML documents 15	
3.3 Name spaces	16
A.Declaring additional vocabularies 16	
Example 3-3:SVG within XHTML 16	
Example 3-4:Xlink 16	
B.Declaring the main vocabulary 17	
Example 3-5:SVG example 17	
C.Namespace URLs 17	
4. DTDs (Document Type Definitions)	18
Example 4-1:A simple DTD 18	
4.1 Using a DTD with an XML document	19
A.Document type declarations 19	
B.Syntax of the DTD declaration in the XML document 20	
C.Some examples of XML documents with DTD declarations: 21	
Example 4-2:Hello XML without DTD 21	
Example 4-3:Hello XML with an internal DTD 21	
Example 4-4:Hello XML with an external DTD 21	

Example 4-5:XML with a public external DTD (RSS 0.91) 21	
<b>4.2 Understanding DTDs by example</b>	<b>22</b>
Example 4-6:Hello text with XML 22	
Example 4-7:A recipe list in XML 23	
Example 4-8:A simple story grammar 25	
Example 4-9:Lone family DTD 26	
Example 4-10:RSS 27	
<b>4.3 Summary syntax of element definitions</b>	<b>29</b>
<b>5. Defining elements (optional chapter, not tested in exam)</b>	<b>30</b>
<b>5.1 Definition of elements</b>	<b>30</b>
<b>5.2 Combination rules</b>	<b>31</b>
<b>5.3 Combination rules for elements</b>	<b>32</b>
<b>5.4 Special contents</b>	<b>33</b>
<b>5.5 Defining attributes</b>	<b>34</b>
Example 5-1:Lone family DTD (file family.dtd) 36	
<b>5.6 General entities</b>	<b>37</b>
<b>5.7 Parameter entities</b>	<b>38</b>
Example 5-2:DTD entities to define reusable child elements 38	
Example 5-3:DTD entities to define reusable attribute definitions 39	
<b>5.8 Some advice for designing DTDs</b>	<b>40</b>
<b>6. XML and CSS</b>	<b>41</b>
<b>6.1 First operations when writing a CSS for XML</b>	<b>41</b>
<b>6.2 Useful CSS2 selectors (optional topic)</b>	<b>42</b>
Example 6-1:Simple XML+CSS page 44	
<b>7. Choosing and using an XML Editor (optional chapter, not tested in exam)</b>	<b>45</b>
<b>7.1 Minimal things your XML editor should be able to do</b>	<b>45</b>
<b>7.2 Additional criteria depending on the kind of XML:</b>	<b>46</b>
<b>7.3 Suggested free editors</b>	<b>47</b>
A.Exchanger XML Lite V3.2: 47	
B.XMLmind Standard Edition: 47	
C.Alternatives 48	
D.About Java 48	

8. Example DTDs and associated XML and CSS files	49
9. Next steps	52
9.1 Reading and self-review	52
9.2 Homework	52
9.3 Next module	52

# 1. Prerequisite - XHTML

## Example 1-1: XHTML Page

- XHTML is an XML application (i.e. it uses the XML formalism).
- Some XML principles:
  - tags (and a way to combine them), a single root, ...
  - XML declaration on top
  - a "grammar" (DTD), namespace declaration, ...

```
<?xml version = "1.0" encoding="UTF-8"?>
<!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>Object Model</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <h1>Welcome to our Web page!</h1>
    <p>Enjoy ! </p>
  </body>
</html>
```

**XML and Document type declaration**

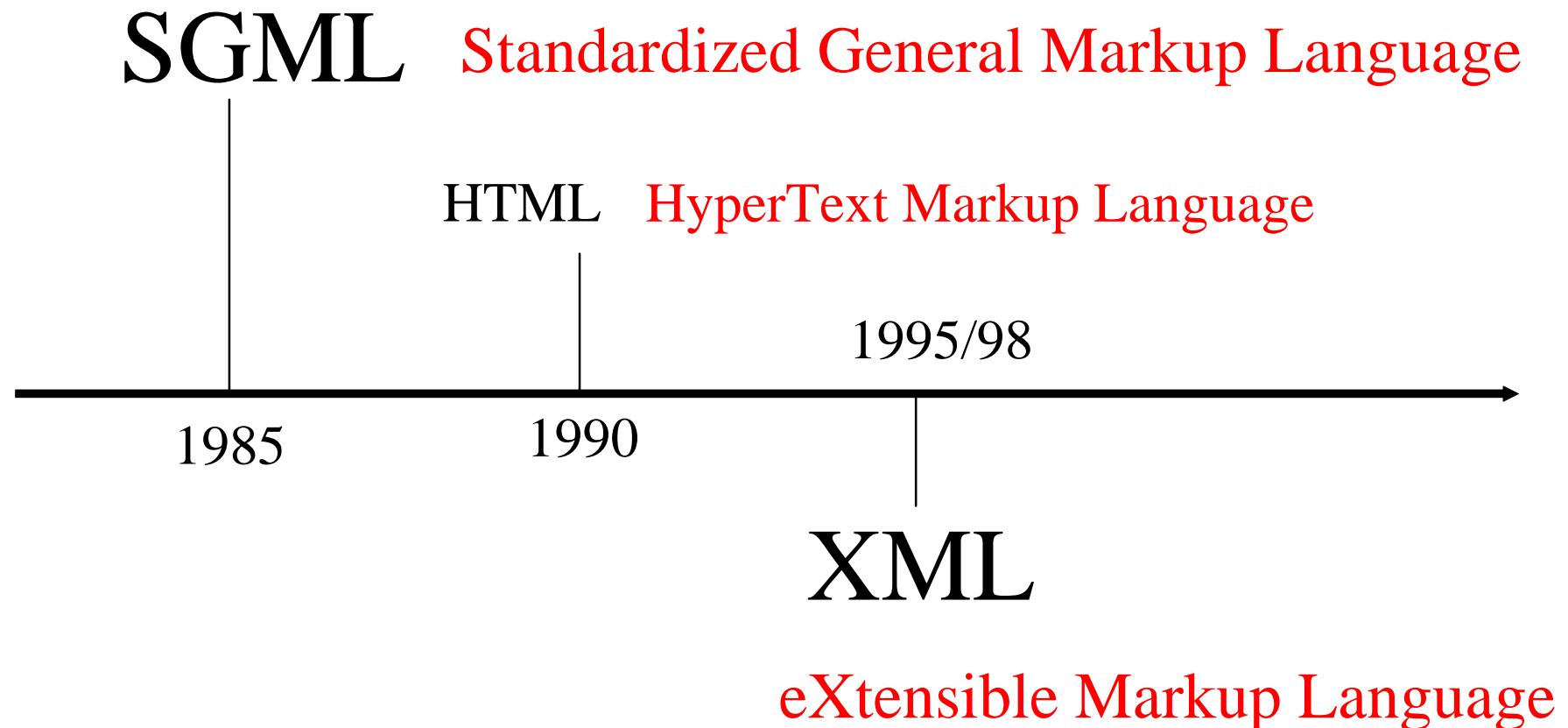
**Namespace declaration**

**Encoding declaration**

**Body (page contents)**

## 2. Goal and scope of XML

### 2.1 History



- T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C Recommendation 10- February-1998,  
<http://www.w3.org/TR/1998/REC-xml-19980210/>

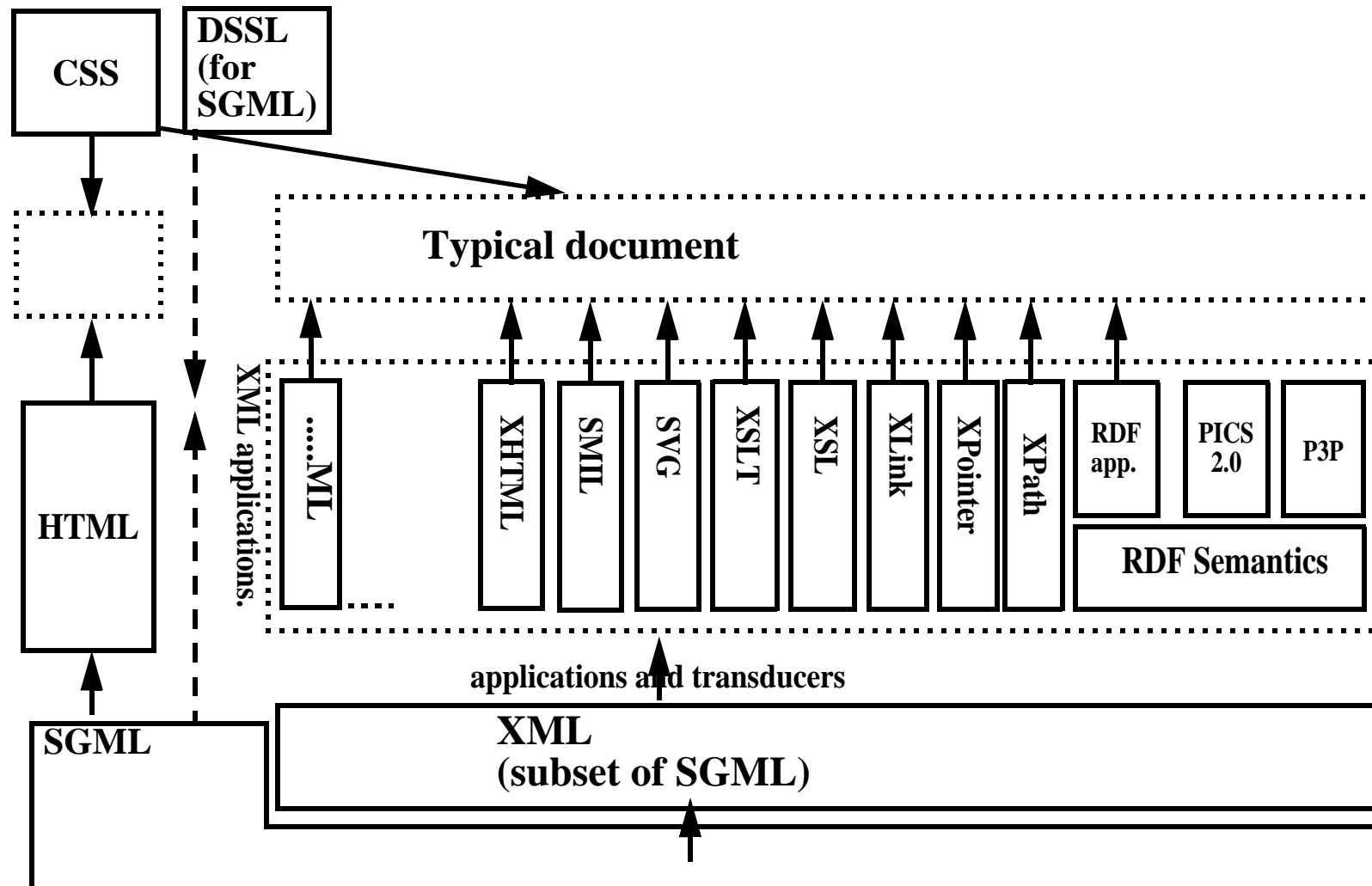
## 2.2 Two ways to look at XML

(1) XML as formalism to define vocabularies (also called applications)	(2) XML as a set of languages for defining:
<p><b>Example DTD :</b></p> <pre>&lt;!ELEMENT page       (title, content, comment?)&gt; &lt;!ELEMENT title (#PCDATA)&gt; &lt;!ELEMENT content (#PCDATA)&gt; &lt;!ELEMENT comment (#PCDATA)&gt;</pre> <p><b>Example of an XML document:</b></p> <pre>&lt;page&gt;   &lt;title&gt;Hello XML friend&lt;/title&gt;   &lt;content&gt;     Here is some content :)   &lt;/content&gt;   &lt;comment&gt;     Written by DKS/Tecfa,   &lt;/comment&gt; &lt;/page&gt;</pre>	<ul style="list-style-type: none"> <li>• Contents, Graphics, Style, Transformation and query languages, Data exchange protocols between machines, Programming languages-learning contents, Metadata, Administrative data</li> </ul> <p>... almost anything you can think of</p> <p><b>XML-languages can be categorized into:</b></p> <ol style="list-style-type: none"> <li>(1) XML <b>accessories</b>, e.g. XML Schema       <ul style="list-style-type: none"> <li>• Extend the capabilities specified in XML</li> <li>• Intended for wide, general use</li> </ul> </li> <li>(2) XML <b>transducers</b>: e.g. XSLT       <ul style="list-style-type: none"> <li>• Convert XML input data into output</li> <li>• Associated with a processing model</li> </ul> </li> <li>(3) XML <b>applications</b>, e.g. XHTML       <ul style="list-style-type: none"> <li>• Define grammars, constraints for a class of XML data</li> <li>• Intended for a specific application area</li> </ul> </li> </ol>



## 2.3 XML documents on the Web

- There is various XML markup on the Web (and increasingly so)
- Some documents may contain multiple vocabularies (e.g. HTML + SVG + MathML)
- Various transducers and accessories (XSLT, Xlink, XPointer, XPath, ... ) may intervene



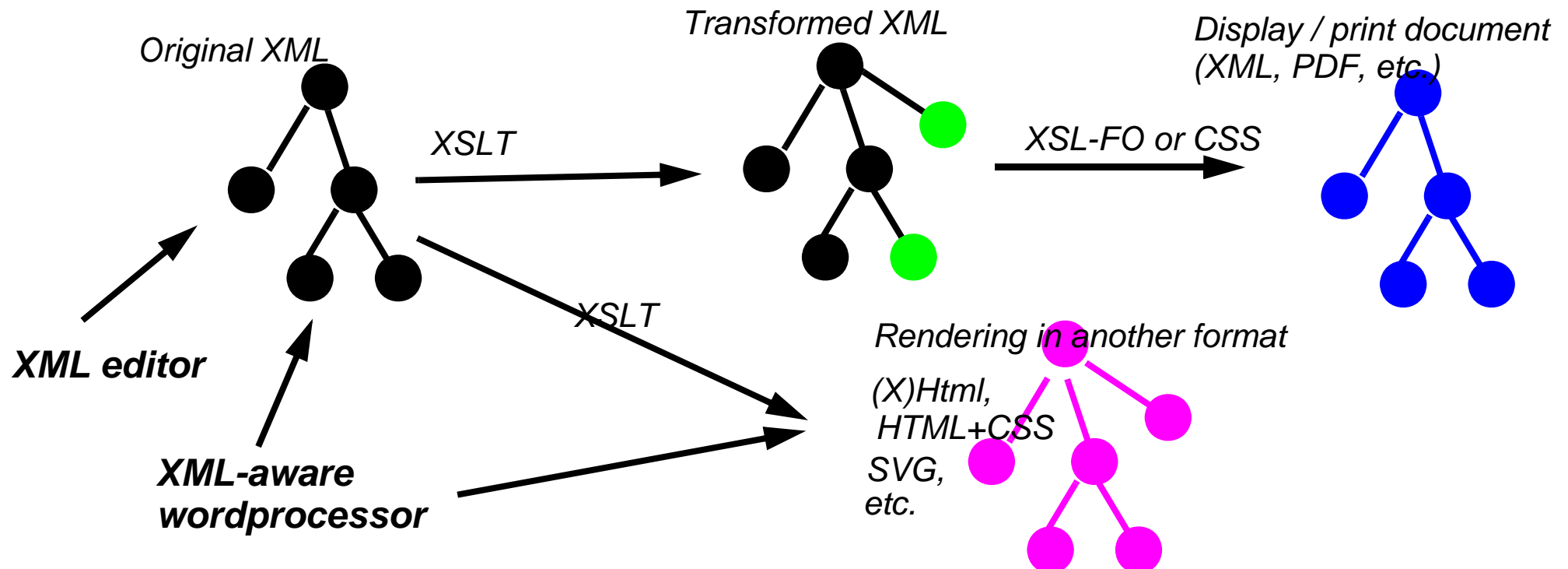
## 2.4 Production of web and print contents

- Any XML content can be displayed in most modern browsers & there are tools for printing.

### Ways to use XML:

- XHTML: HTML rewritten in XML
- Any XML document together with a CSS stylesheet or an XSLT transformation
- Specialized formats like SVG (vector graphics), X3D (3d vector graphics), MathML (formulas)
- Combinations of the above (more difficult !)
- A word processor plus output filters

### Typical document production/delivery pipelines



## 3. 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

### 3.1 Contents of an XML document

#### **An XML document includes:**

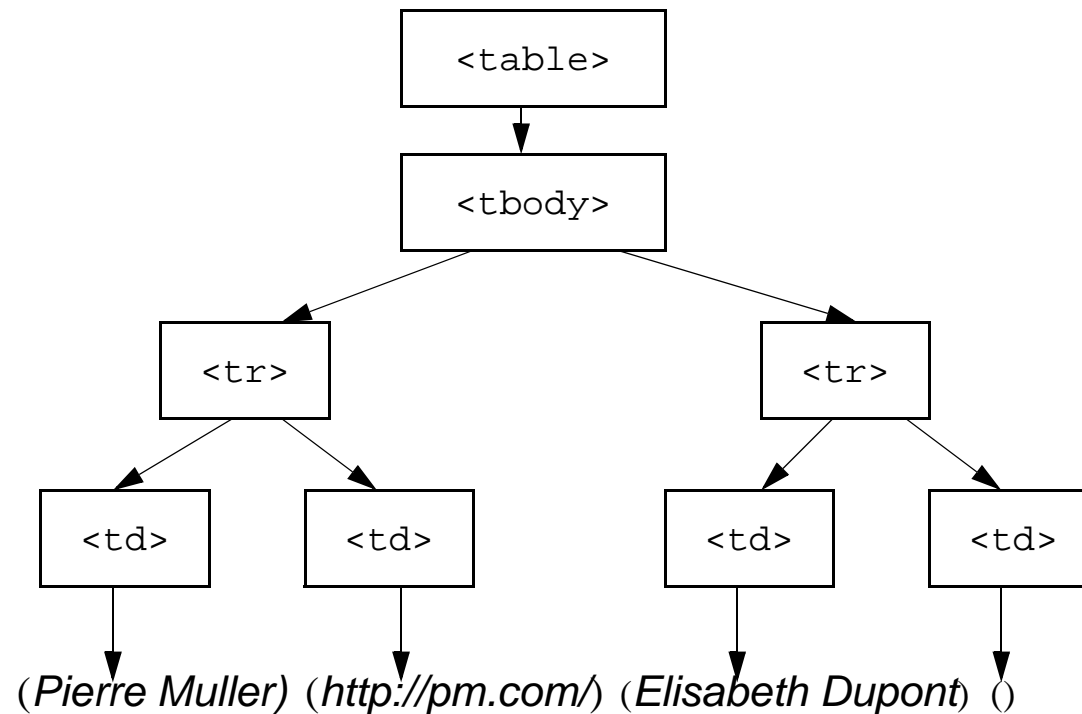
- Processing instructions (at least an XML declaration on top !)
- declarations (e.g. a Document Type Definitions)
- marked up contents (mandatory): elements
- marked up contents (optionally): attributes and entities
- comments: `<!-- . . . . -->`

## 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 3-1: XML data as tree (XHTML 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>
```



## 3.2 “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. E.g. this is bad: `<i>...<b>...</i> .... </b>`
- case sensitivity, e.g. "LI" is not "li"

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

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

- Attributes must ***have values*** and values are quoted:

- e.g. `<a href="http://scholar.google.com">` 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 `&lt;`; `&amp;`; `&gt;`; `&quot;`; `&apos;`; instead of `<`, `&`, `>`, `"`, `'`
- Applies also to URLs !!

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

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

## Example 3-2: 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. Most grammars are written with DTD.
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)

## 3.3 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 3-3: 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 3-4: Xlink

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

```
<STORY 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 (default) vocabulary can be introduced by the `xmlns` attribute, like this:

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 3-5: 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 purposes you can very well make up something like:

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

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

## 4. DTDs (Document Type Definitions)

**DTD grammars are a set of rules that define:**

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

### 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, write them out in a specification document
    - e.g. "the value of length attribute is a string composed of a number plus "cm" or "inch" or "em"
- example: `<size length="10cm">`

### Example 4-1: A simple DTD

*url: init/very-simple-page.dtd*

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

- A DTD document contains just rules .... nothing else (see later for explanations)

## 4.1 Using a DTD with an XML document

### A. Document type declarations

- A **valid** XML document includes a **declaration that specifies the DTD** used
- DTD is declared on top of the file after the XML declaration.
- XML declarations, DTD declaration etc. are part of the prologue
- 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">
<hello>Here we <strong>go</strong> ... </hello>
```

### 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**

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

## B. Syntax of the DTD declaration in the XML document

- A DTD declaration starts with the keyword "DOCTYPE":

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

- ... followed by the root element
  - 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 . . . . >
```

- ... followed by the DTD definition or a reference to a DTD file

### 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">
```

## C. Some examples of XML documents with DTD declarations:

### Example 4-2: Hello XML without DTD

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

### Example 4-3: 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 4-4: Hello XML with an external DTD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE hello SYSTEM "hello.dtd">
<hello> This is a very simple XML document </hello>
```

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

### Example 4-5: 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>
```

## 4.2 Understanding DTDs by example

- Recall that DTDs define all the elements and attributes and the way they can be combined

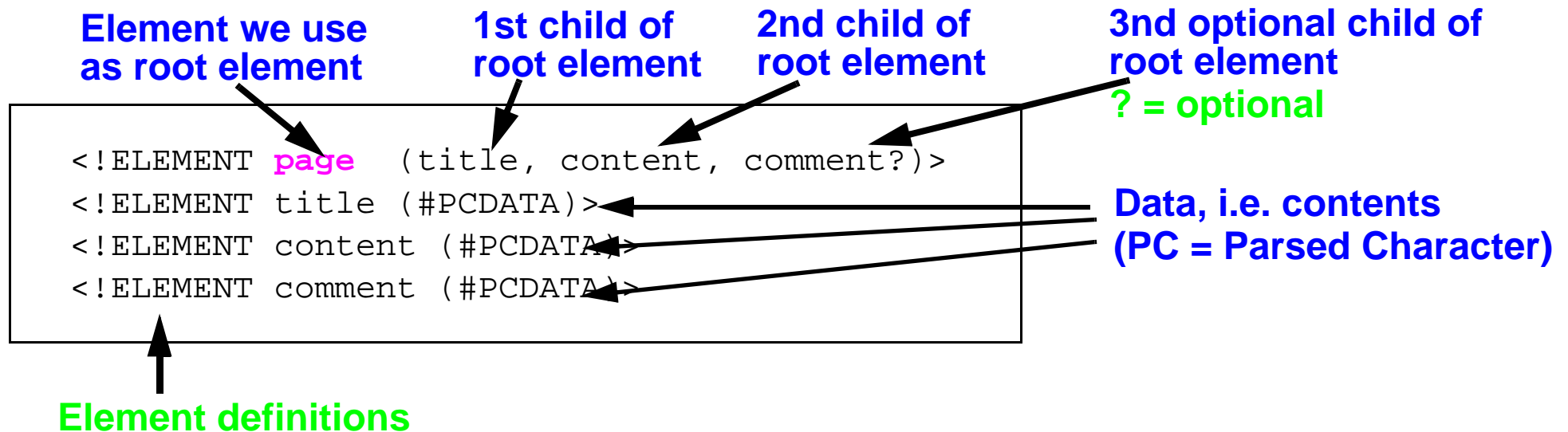
### Example 4-6: Hello text with XML

url: init/very-simple-page.xml

#### 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 4-7: A recipe list in XML

*url: init/simple\_recipe.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</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

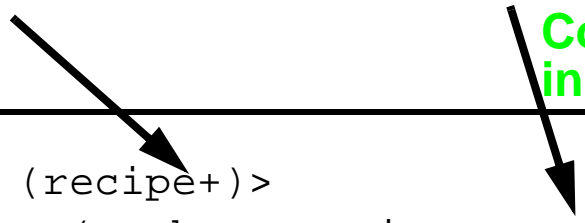
*url:* init/simple\_recipe.dtd

**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 4-8: A simple story grammar

url: medium/story-grammar.dtd

**The THREADS element  
must contain one or more  
EPISODE elements**  
**+ means "at least one"**

**All elements of STORY  
must be present  
in this order**

**Comment**

**Choose one  
| means "or"**

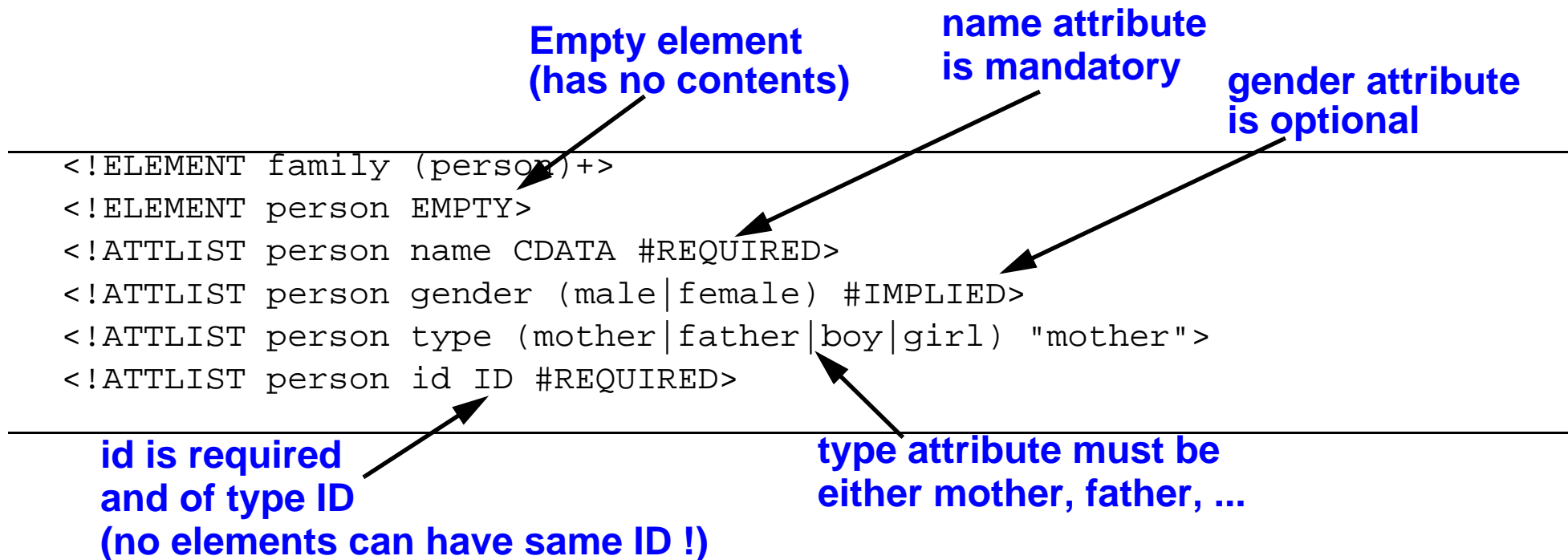
**Choose as  
many as  
you like  
in random  
order**

**These elements  
only contain  
text**

```
<?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 -->
<!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" >
```

## Example 4-9: Lone family DTD

url: simple/family.dtd and simple/family.xml



## 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 4-10: RSS

*url: complex/rss-0-92.dtd*

- 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

**url: complex/rss-0-92.xml**

```
<?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>
```

## 4.3 Summary syntax of element definitions

- The purpose of this module is not to teach you how to write DTDs
- To understand how to use DTDs, you just need to know how to read a DTD

syntax element	short explanation	Example Element definitions Valid XML example
,	<ul style="list-style-type: none"> <li>• elements in that order</li> </ul>	<code>&lt;!ELEMENT Name (First, Middle, Last)&gt;</code> <ul style="list-style-type: none"> <li>• Element Name must contain First, Middle and Last</li> </ul> <pre>&lt;Name&gt;   &lt;First&gt;D.&lt;/First&gt;&lt;Middle&gt;K.&lt;/Middle&gt;&lt;Last&gt;S.&lt;/Last&gt; &lt;/Name&gt;</pre>
?	<ul style="list-style-type: none"> <li>• optional element</li> </ul>	<code>&lt;!ELEMENT Name (First,Middle?,Last)&gt;</code> <ul style="list-style-type: none"> <li>• Middle is optional</li> </ul> <pre>&lt;Name&gt;&lt;First&gt;D.&lt;/First&gt;&lt;Last&gt;S.&lt;/Last&gt;&lt;/Name&gt;</pre>
+	<ul style="list-style-type: none"> <li>• at least one element</li> </ul>	<code>&lt;!ELEMENT list (movie+)</code> <pre>&lt;list&gt;&lt;movie&gt;Return of ...&lt;/movie&gt;   &lt;movie&gt;Comeback of ...&lt;/movie&gt; &lt;/list&gt;</pre>
*	<ul style="list-style-type: none"> <li>• zero or more elements</li> </ul>	<code>&lt;!ELEMENT list (item*)</code> <ul style="list-style-type: none"> <li>• almost as above, but list can be empty</li> </ul>
	<ul style="list-style-type: none"> <li>• pick one (or operator)</li> </ul>	<code>&lt;!ELEMENT major (economics   law)</code> <pre>&lt;major&gt; &lt;economics&gt; &lt;/economics&gt; &lt;/major&gt;</pre>
()	<ul style="list-style-type: none"> <li>• grouping construct, e.g. one can add ? or * or + to a group.</li> </ul>	<code>&lt;!ELEMENT text (para   list   title)*</code> <pre>&lt;text&gt; &lt;title&gt;Story&lt;/title&gt;&lt;para&gt;Once upon a time&lt;/para&gt; &lt;title&gt;The awakening&lt;/title&gt; &lt;list&gt; ... &lt;/list&gt; &lt;/text&gt;</pre>

## 5. Defining elements (optional chapter, not tested in exam)

### 5.1 Definition of elements

#### Rough syntax of a DTD rule to define elements:

Syntax: `<!ELEMENT tag_name child_element_specification>`

#### Child\_element\_specification may contain:

- A combination of child elements according to combination rules

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

- Mixed contents, i.e. child elements plus #PCDATA or ANY

```
<!ELEMENT para (strong | #PCDATA )*>
```

- #PCDATA (Just data)

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

- ANY (only used during development)

```
<!ELEMENT para (ANY)*>
```

- EMPTY (the element has no contents)

```
<!ELEMENT person EMPTY>
```

#### Tag names

- Each tag name must start with a letter or an underscore ('\_') followed by letters, numbers or the following characters: '\_', '-', '.', ':'

**BAD example:** `<!ELEMENT 1st ...>`

**BAD example:** `<!ELEMENT My Home ...>`

## 5.2 Combination rules

A and B = tags	Explanation	DTD example	XML example
A , B	A followed by B	<code>&lt;!ELEMENT person (name ,email?)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt; &lt;/person&gt;</code>
A?	A is optional, (it can be present or absent)	<code>&lt;!ELEMENT person (name, email?)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt;&lt;/person&gt;</code>
A+	At least one A	<code>&lt;!ELEMENT person (name, email+)&gt;</code>	<code>&lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;   &lt;email&gt;x@y.x&lt;/email&gt; &lt;/person&gt;</code>
A*	Zero, one or several A	<code>&lt;!ELEMENT person (name, email*)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt; &lt;/person&gt;</code>
A   B	Either A or B	<code>&lt;!ELEMENT person (email   fax)&gt;</code>	<code>&lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;fax&gt;123456789&lt;/fax&gt;&lt;/person&gt;</code>
(A, B)	Parenthesis will group and you can apply the above combination rules to the whole group	<code>&lt;!ELEMENT list (name, email)+ &gt;</code>	<code>&lt;list&gt;   &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;     &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;/list&gt;</code>

## 5.3 Combination rules for elements

A and B = tags	Explanation	DTD example	XML example
A , B	A followed by B	<code>&lt;!ELEMENT person (name ,email?)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt; &lt;/person&gt;</code>
A?	A is optional, (it can be present or absent)	<code>&lt;!ELEMENT person (name, email?)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt;&lt;/person&gt;</code>
A+	At least one A	<code>&lt;!ELEMENT person (name, email+)&gt;</code>	<code>&lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;   &lt;email&gt;x@y.x&lt;/email&gt; &lt;/person&gt;</code>
A*	Zero, one or several A	<code>&lt;!ELEMENT person (name, email*)&gt;</code>	<code>&lt;person&gt;   &lt;name&gt;Joe&lt;/name&gt; &lt;/person&gt;</code>
A   B	Either A or B	<code>&lt;!ELEMENT person (email   fax)&gt;</code>	<code>&lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;   &lt;fax&gt;123456789&lt;/fax&gt;&lt;/person&gt;</code>
(A, B)	Parenthesis will group and you can apply the above combination rules to the whole group	<code>&lt;!ELEMENT list (name, email)+ &gt;</code>	<code>&lt;list&gt;   &lt;person&gt; &lt;name&gt;Joe&lt;/name&gt;     &lt;email&gt;x@x.x&lt;/email&gt;&lt;/person&gt; &lt;/list&gt;</code>



## 5.4 Special contents

Special elements	Explanation	DTD examples	XML example
<b>#PCDATA</b>	"Parsed Character Data" Text contents of an element. It should not contain any <, >, & etc.	<code>&lt;!ELEMENT email (#PCDATA)&gt;</code>	<code>&lt;email&gt;Daniel.Schneider@tecfa.unige.ch&lt;/email&gt;</code>
<b>ANY</b>	Allows any non-specified child elements and parsed character data (avoid this !!!)	<code>&lt;!ELEMENT person ANY&gt;</code>	<code>&lt;person&gt;   &lt;c&gt;text&lt;/c&gt;   &lt;a&gt;some &lt;b&gt;bbb&lt;/b&gt;     inside &lt;/a&gt; &lt;/person&gt;</code>
<b>EMPTY</b>	No contents	<code>&lt;!ELEMENT br EMPTY&gt;</code>	<code>&lt;br/&gt;</code>

### Note about Mixed Content

- Mixed element contents contain both text and tags.

```
<para> here is <a href="xx">link</a>. <b>Check</b> it out </para>
```

- You have to use the "|" construct for these

- Good examples:

```
<!ELEMENT para (#PCDATA|a|ul|b|i|em)*>
```

```
<!ELEMENT p (#PCDATA | a | abbr | acronym | br | cite | code | dfn | em | img | kbd |  
             q | samp | span | strong | var )* >
```

```
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
```

- Bad examples:

```
<!ELEMENT p (name, first_name, #PCDATA)*>
```

```
<!ELEMENT p ( (#PCDATA) |a|ul|b|i|em)*>
```

## 5.5 Defining attributes

### Rough syntax of Attribute rules:

```
<!ATTLIST element_name attr_name Attribute_type Type_Def Default >
```

### Overview:

Type	Attribute types
<b>CDATA</b>	"Character Data" - Text data
<b>NMTOKEN</b>	A single word (no spaces or punctuations)
<b>ID</b>	Unique identifier of the element.
<b>IDREF</b>	Reference to an identifier.
<b>IDREFS</b>	Reference to one or more identifiers
<b>(A B C ..)</b>	List of values (from which the user must choose)

	Type Definition
<b>#IMPLIED</b>	Attribute is optional
<b>#REQUIRED</b>	Attribute is mandatory)
<b>#FIXED Value</b>	Attribute has a fixed value (user can't change it)

- See next slides for examples ....

## Illustrations:

DTD rule	example XML
<!ATTLIST person first_name CDATA #REQUIRED>	<person first_name="Joe">
<!ATTLIST person gender (male female) #IMPLIED>	<person gender="male">
<!ATTLIST form method CDATA #FIXED "POST">	<form method="POST">
<!ATTLIST list type (bullets ordered) "ordered">	<list type="bullets">
<!ATTLIST sibling type (brother sister) #REQUIRED>	<sibling type="brother">
<!ATTLIST person id ID #REQUIRED>	<person id="N1004">

### Shortcut to define multiple attributes for an element:

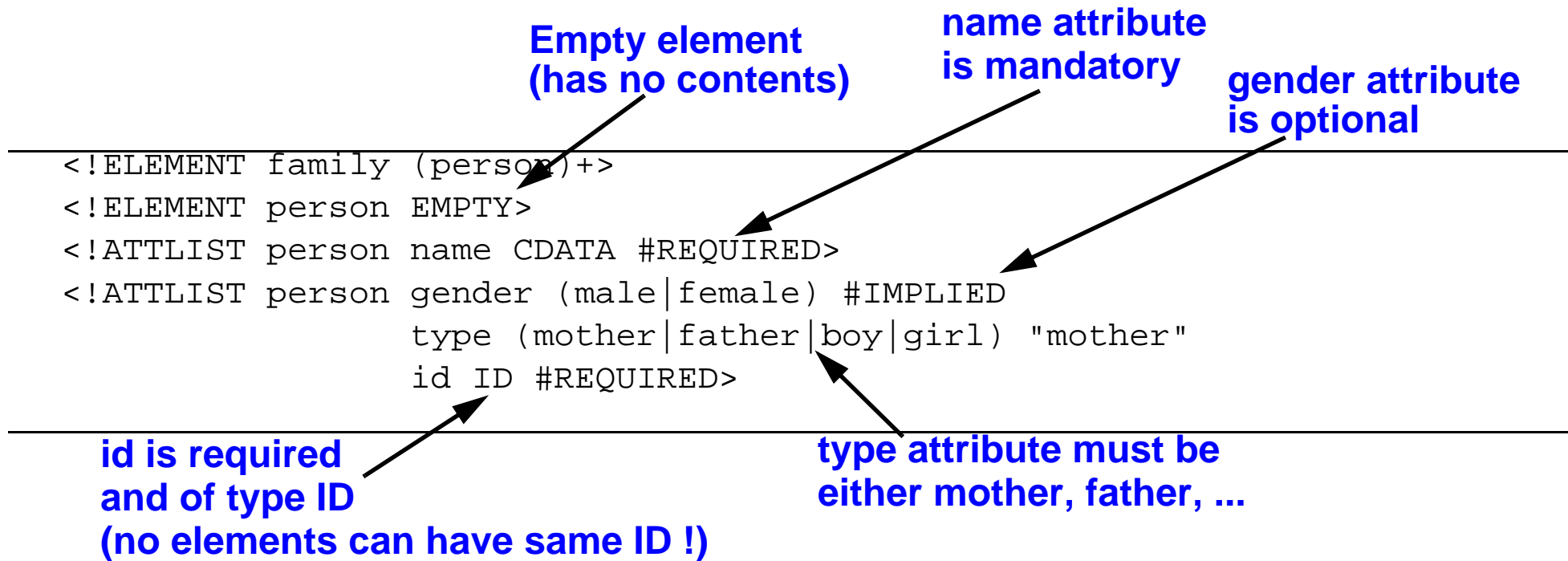
Syntax: <!ATTLIST     *target\_tag*

<i>attr1_nom</i>	<i>TypeAttribut</i>	<i>TypeDef</i>	<i>Default</i>
<i>attr2_nom</i>	<i>TypeAttribut</i>	<i>TypeDef</i>	<i>Default</i>
...			

>

### Shortcut illustrations:

```
<!ATTLIST person
    ident      ID          #REQUIRED
    gender     male|female) #IMPLIED
    nom        CDATA       #REQUIRED
    prenom     CDATA       #REQUIRED
    relation   brother|sister) #REQUIRED >
<!ATTLIST portable
    owner      IDREF       #REQUIRED >
```

**Example 5-1: Lone family DTD (file 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="N123456789" />
  <person name="Josette Miller" gender="female"
          type="girl" id="N123456987" />
</family>
  
```

## 5.6 General entities

Consider entities as abbreviations for some other content. An entity must be defined in the DTD and its contents are substituted when encountered in the XML file.

- Recall that XML initially only defines 5 entities and that HTML does many more...
- Use the `&lt;`, `&amp;`, `&gt;`, `&quot;`, `&apos;` entities to refer to `<`, `&`, `>`, `"` and `'`

Syntax of an internal entity definition: `<!ENTITY entity_name "content">`

Syntax of an external entity definition: `<!ENTITY entity_name SYSTEM URI>`

Syntax of using an entity: `&entity_name;`

### Illustrations of entity definitions:

DTD rule	XML example	Result
<code>&lt;!ENTITY jt "Joe Test"&gt;</code>	<code>&lt;para&gt; &amp;jt; is here&lt;/para&gt;</code>	<code>&lt;para&gt; Joe Test is here&lt;/para&gt;</code>
<code>&lt;!ENTITY space "&amp;#160;"&gt;</code>		
<code>&lt;!ENTITY copyright "&amp;#xA9;"&gt;</code>	<code>&amp;copyright; D. Schneider</code>	
<code>&lt;!ENTITY explanation SYSTEM "project1a.xml"&gt;</code>	<code>&lt;citation&gt; &amp;explanation; &lt;/citation&gt;</code>	<code>&lt;citation&gt; contents of project1a.xml ... &lt;/citation&gt;</code>

## 5.7 Parameter entities

- Most professional DTDs use parameter entities.
- These are used to simplify DTD writing

Syntax: `<!ENTITY % entity_name "content">`  
`<!ENTITY % entity_name SYSTEM "URI">`

### Example 5-2: DTD entities to define reusable child elements

- More complex DTD often use same structures all over. Instead of typing these several times for each element definition, one can use an 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 XML parser will then simply translate these `%Content;` and we get:

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

### Example 5-3: DTD entities to define reusable attribute definitions

- You may use the same procedure to define "bricks" for attribute definitions
- Entity example that defines part of an attribute definition

```
<!ENTITY % stamp '  
  id ID #IMPLIED  
  creation-day NMTOKEN #IMPLIED  
  .....  
  mod-by NMTOKEN #IMPLIED  
  version NMTOKEN #IMPLIED  
  status (draft|final|obsolete) #IMPLIED  
  approval (ok|not-ok|so-so) #IMPLIED  
  main-author CDATA #IMPLIED  
'  
>
```

ATTLIST definitions below use %stamp;

```
<!ELEMENT main-goal (title, content, (after-thoughts)?, (teacher-comments)?)>  
<!ATTLIST main %stamp; >  
<!ELEMENT title (...)>  
<!ATTLIST main %stamp; >
```

## 5.8 Some advice for designing DTDs

### Don't forget elements and be liberal

- Each element needs to be defined, but only once !
- Only make elements mandatory if they really are wanted, else use e.g. `element?`

### Plan the global structure

- Before you start writing out DTDs, use some simple "language" to draft the structure, e.g. use a notation like:

```
name      ==> family + given
family    ==> "text"
```

- In most cases, each "object" of your "information domain" becomes an element
  - Use child elements to model components
  - Use attributes to describe properties of components

### Start from the root element and work your way down:

1. Root element
2. Child elements of root element
3. Child elements of the other elements, etc.



## 6. XML and CSS

- XML shows "as is" in a web browser. You either have to write a CSS stylesheet for **each** element or translate contents to HTML with XSLT ....
- To apply a CSS stylesheet to a document, use a processing instruction in your XML file:

```
<?xml-stylesheet type="text/css" href="my_style.css"?>
```

- CSS properties and values are the same as for HTML

### 6.1 First operations when writing a CSS for XML

1. Use the root element to define margins, default font, etc.
2. Decide which elements are blocks and which ones are inline
3. Identify "special elements" like titles and lists

#### Some example CSS rules

```
/* title and para elements are blocks. They have an extra margin */  
title, para {display: block; margin: 0.5em;}  
/* title element font is 1.5 bigger and red */  
title {font-size: 1.5em; color:red;}  
/* item elements are list elements, we use bullet style */  
item {display: list-item;list-style-type: disc;}  
/* strong is an inline element. Uses italic style and blue color */  
strong {display: inline; font-style: italic; color: rgb(000,000,128);}
```

## 6.2 Useful CSS2 selectors (optional topic)

- XML needs a navigator that supports at least partially CSS2
- For the stylesheet use the usual CSS syntax:  
`selector {property1:value1; property2:value2; ...}`
- A simple **selector** would be an element name from the XML document (see above)
- These selectors also work with HTML ...

### ***selection of an element (mostly you will use this)***

**Syntax:** `element`

example:

```
Step {  
    display: list-item;  
    list-style-type: decimal;  
}
```

### ***selection of a child element***

**Syntax:** `mother_element > child_element`

Example:

```
Step > Title { .... }
```

### ***selection of descendant element (child, great-child, etc.)***

**Syntax:** `mother_element element`

example:

```
Step Title { .... }
```

## **combinations**

example:

```
DIV OL>LI P {.... }
```

## ***selection siblings (elements next to each other sharing the same parent)***

**Syntax:** `sister_element + sister_element`

example:

```
H1 + H2 { margin-top: -5mm }
```

## ***selection of an element that has a certain attribute***

**Syntax:** `element[attribute]`

example:

```
Title[status] { color: blue; }
```

(all titles that have a status attribute are rendered in blue )

## ***selection of an element that has an attribute with a given value***

**Syntax:** `element[attribute="value"]`

example:

```
Title[status="draft"] { color: red; }
```

## ***selection of an element that has an attribute with a given value in a comma-sep. list***

```
Title[status~="draft"] { color: blue; }
```

## Example 6-1: Simple XML+CSS page

### XML: simple/simple-page.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet href="simple-page.css" type="text/css"?>
<page updated="jan 2007">
  <title>Hello friend</title>
  <content> Here is some content  </content>
  <content> Here is some more content :) </content>
  <comment> Written by DKS/Tecfa </comment>
</page>
```

### CSS: simple/simple-page.css

```
/* Definitions that apply to the whole hierarchy */
page { font-family:Times; line-height:1.5;}
/* Margins for the box of the root element */
page { margin-top:3cm; margin-left:3cm; margin-right:3cm; }

/* Block elements */

title, content, comment { display:block; }

title { font-family: Arial; font-size:1.5em;}
content { }
comment { font-style:italic; }
```

## 7. Choosing and using an XML Editor (**optional chapter, not tested in exam**)

- 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](http://edutechwiki.unige.ch/en/XML_editor)

### 7.1 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)

## 7.2 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)

## 7.3 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".



## 8. Example DTDs and associated XML and CSS files

### Instructions

- Produce one of the following XML documents 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

### Remarks

- Most of the XML files are almost empty (and just help you to get started !)
- CS majors and minors may try a medium or complex DTD
- There may be little mistakes .... try to fix them
- Most CSS files are not really complete, some may be missing

DTD (difficulty)	Purpose	file names	Additional directions
Recipe DTD (easy)	Write simple recipes	simple/recipe.xml simple/recipe.dtd simple/recipe.css	Use all tags. Write at least one recipe. Make sure that there is enough information to really use it.

DTD (difficulty)	Purpose	file names	Additional directions
Recipe DTD 2 (easy)	Write simple recipes	simple/recipe2.xml simple/recipe2.dtd simple/recipe2.css	Use all tags. Write at least one recipe. Make sure that there is enough information to really use it.
Addressbook	Manage an addressbook	simple/address_book.xml simple/address_book.dtd simple/address_book.css	Use all tags, Write at least 2 entries. Complete the CSS stylesheet.
Simple letter (easy)		simple/letter.xml simple/letter.dtd simple/letter.css	Try to write a "real" letter. Improve the CSS file, e.g. try to position header and footer elements.
Addressbook (medium)	Simple addressbook that includes HTML elements.	namespaces/address.xml address.dtd address.css	Use all tags. Enter at least 2 addresses. You may adapt the CSS file
Instructions (medium)	Write "how-to" instructions	medium/instructions.xml medium/instructions.dtd medium/instructions.css	Imagine that you have to explain how to do something. Try to write a "real" text.
Minutes (medium)	Write minutes	medium/minutes.xml medium/minutes.dtd medium/minutes.css	Imagine a real meeting and write down notes. Try to adapt the DTD to specific needs.
StepbyStep (medium)	Write "how-to" instructions (similar to above)	namespaces/ stepbystep03.dtd	Make up a good "how-to problem". Only use tags you need..

DTD (difficulty)	Purpose	file names	Additional directions
Story grammar (medium)	Write simple fairy tales	medium/story-grammar.dtd medium/story-grammar.xml medium/story-grammar.css	Write a nice fairy tale. Doesn't need to be your own. Make sure to understand the structure of the story.
Recipe Markup Language (difficult)	Write complex recipes	complex/recipe.xml.dtd	Only use appropriate tags. Hint: find the website of its creator
RSS 0.92 (difficult)	News syndication (usually machine generated)	complex/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 !
ePBL project and paper (very difficult)	Term project and paper DTD	complex/ePBLpaper11.dtd complex/ePBLproject11.dtd complex/ibtwsh6_ePBL.dtd	Try to understand what the ePBLxxx DTDs do. Then figure out what ibtwsh6 does ...
Simple Docbook (very difficult)	Write "real" articles	complex/sdocbook.dtd	Do not use all tags, only the needed ones. Copy/paste from a text you already have. Find stylesheet on the Internet

## 9. Next steps

### 9.1 Reading and self-review

- Reading  
Deitel, Internet and World Wide Web, How to Program, chapter 20, p.630-658,
- Self-exercising  
Deitel, Internet and World Wide Web, How to Program, chapter 20, p.682-684

### 9.2 Homework

1. Create a valid XML page from the list of DTDs described in «Example DTDs and associated XML and CSS files» [p. 49]
  - 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
2. Modify the CSS stylesheet or create one if there is none
3. For CS majors (optional): Modify the DTD
  - Homework will not be evaluated. However, doing it will help you pass the exams ...

### 9.3 Next module

- Introduction to XSLT