

**Concept of XML**

**Budapest, Hungary**

**7th CEEnet workshop on networks technology**

**August 2001**

**Sébastien 'sbi' BLONDEEL (<sbi@IDEALX.com>)**

**Table of Contents**

- 1 Who am I?
- 2 Preamble
- 3 Pre-requisites to follow this and do the exercises
- 4 Examples of Unix shell manipulation
- 5 Example of a Makefile
- 6 How to type characters not present on the keyboard
- 7 What is XML?
- 8 The components of XML
- 9 Special characters in XML
- 10 The XML header
- 11 Example: the header of this presentation

- 12 Syntactic rules of XML
- 13 Example: an XML file
- 14 XML compared to SGML
- 15 The goals pursued in the making of XML
- 16 Well-formedness and validity
- 17 Syntax of a DTD
- 18 Example: the DTD of this presentation
- 19 When to stop tagging?
- 20 Encoding issues
- 21 Lab on Concept of XML

**Abstract**

Networks and the speedup and ease in communications brought the possibility of interchanging documents without using paper to transmit them. XML is a new trend a few years old, and one of its aims is to facilitate establish a gap between what a document looks like, and the content it holds, (re)opening the way to the idea of writing once a document, and using several times in different formats. This slideshow uses an XML file and a Makefile transforms it to four different formats. You will be given a pointer to all the files and compilation procedures used to produce this presentation.

**Who am I?**

- Picture of the place where I was met
- Personal interests and websites
- What I may famous for, or proud (?) of

**Preamble**

- XML is a new, poorly understood, and over-hyped concept.
- It is hardly anything more than syntactical sugar
- One should distinguish well between XML as a communications media and data storage, and XML as a way to write documents
- The storage format of your documents is important
- I suggest your relation to content and markup to be of a higher level than just focusing on the way things look
- Virtualizing markup helps create different versions of a document: compare them
- Semantic markup ensures more coherence than most WYSIWYG environments default settings
- You are independent from software clients and interface and can profit

or the same invested time to edit efficiently mail, documents, programs

- If coupled with research or database tools, semantic markup can help better qualification of information: "python" can be a snake, a man, or a programming language

**Pre-requisites to follow this and do the exercises**

- quite a transversal and complete talk
- this works in (free) unix systems, and should in others
- we will try to use proprietary systems and software, or I can leave the comparison as an exercise to the students once back home
- know UNIX shell manipulation
- know some window manager manipulation
- know how to type, including accents not present on keyboard (ask me)
- know how to set up environment variables
- know how to read, write and understand a Makefile
- know how to ftp and manipulate archives
- never fall behind, tell me when you don't follow

**Examples of Unix shell manipulation**

```
$ ls -al
```

```
$ apropos xml
```

```
$ man nsgmls | grep -2i variable
```

```
$ grep -2h usepackage 'locate tex | \  
grep '\.tex$' | sort -u
```

```
$ for file in `locate xml`; do grep -li \  
DOCTYPE $file /dev/null; done
```

```
$ CLASSPATH=/path/to/xt/xt.jar: \  
java com.jclark.xml.sax.Driver
```

**Example of a Makefile**

```
ROOT_DIR=$(shell pwd)
FILE=CEEnet2001_xml
XSLT=$(JAVA) com.jclark.xsl.sax.Driver
[...]
TEXINPUTS=$(ROOT_DIR)/figure/:$(ROOT_DIR)/sty/:
LATEX=TEXINPUTS=$(TEXINPUTS) latex \\scrollmode\\input

all:    valid

valid:
    $(NSGMLS) $(FILE).xml
[...]
```

**How to type characters not present on the keyboard**

- Historically: computers speak English: ASCII charset
- Address space too small (1 octet = 256 possibilities).
- Different pages: ISO-LATIN-1, ISO-LATIN-2, ...
- Trick: commands
  - setfont, setmetamode (in console)
  - setxkbmap (under X Window System)
- How to type Chinese, Japanese? Ex: cterm
- Unicode coming slowly.

**What is XML?**

The eXtensible Markup Language<sup>a</sup> is a few years old.

It is being normalized by the W3C: drafts and recommendations.

Standards are still being worked on.

It is a meta-language.

It is just syntactic sugar.

---

<sup>a</sup><http://w3c.org/XML/>

**The components of XML**

- a header
- tags surrounding elements
- attributes
- entities
- comments
- text content
- CDATA sections
- and some more (directives, ...)

**Special characters in XML**

Because XML has a syntax similar to that of HTML, some characters may not be reproduced literally. These are:

- left angle bracket (actually, "less than"): &lt;
- right angle bracket ("greater than"): &gt;
- ampersand: &amp;
- single quote (or "apostrophe"): &apos;
- double quote: &quot;

**The XML header**

Must specify:

- the version of XML (mandatory)
- the encoding used (if non-ASCII characters)

Can specify:

- other attributes (stand-alone, ...)
- a DTD, with a PUBLIC and SYSTEM identifier
- parameter entities to ignore parts (like draft) or include the contents of a file

**Example: the header of this presentation**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE presentation PUBLIC
"-//Sebastien Blondeel//DTD presentation slides//EN"
"dtd/slides.dtd" [
  <!-- extra entities -->
]>
<presentation>
  <title>Concept of XML</title>
  <conference>7th CEEnet workshop on networks tech-
nology</conference>
  <place>Budapest, Hungary</place>
  ...

```

**Syntactic rules of XML**

- Size (of letters) does matter: tags are case sensitive
- Some characters are forbidden in tag names
- Tags must be closed
- Special case: empty tags
- Elements must be properly nested
- Attributes must be quoted (or double-quoted)

**Example: an XML file**

```
<?xml version='1.0'?>
<root>
  <foo lang="hu">
    <anchor id="fooAnchor"/>
    Friend or foo?
    <bar>I started after, I must finish first.
    </bar> <!-- no attributes in closing tags -->
  </foo>
  <blah>
    Another element directly under the root element.
  </blah>
</root>
```

**XML compared to SGML**

The Standard Generalized Markup Language is older and much more complicated:

- tag delimiters can be redefined
- opening or closing tags can be omitted
- empty tags can be implied
- some elements can be empty if and only if some attribute has a special value
- and (probably!) more... I am not sure I want to know

**The goals pursued in the making of XML**

(courtesy of "A Technical Introduction to XML"<sup>a</sup>, by Norman Walsh)

- straightforward to use over the Internet (when on the fly XML-aware browsers exist and work)
- shall support many applications (more than just web content: authoring, content analysis, ...)
- easy to parse: two weeks for a competent graduate student
- compatible with SGML for backward-compatibility reasons
- as few optional features as possible (for portability)
- human-readable and clear
- design must be achieved quickly

---

<sup>a</sup><http://www.xml.com/pub/98/10/guide1.html>

- design must be clear and concise (EBNF)
- documents easy to create: text editors, scripts...
- terseness does not matter: compressing software exists, and saving typing complicates the design and the parsing

### Well-formedness and validity

When an XML document satisfies the syntactic rules cited above, then it is called *well-formed*.

The whole point of XML is to give semantics to content. There are ways to describe the structure of the tags and elements an XML document must follow. This is being done using a Document Type Definition, or DTD.

DTDs are very poor schemes. They are not XML documents, therefore cannot be easily checked (against some super-DTD!) with usual XML tools.

DTDs have no typing of data and very little expressivity. Most DTDs published are full of mistakes and cannot possibly be used by those who publish them as "examples" on their websites.

When the structure of a document obeys the rules given in the DTD that accompanies it, then this document is called *valid*.

DTDs exist for historical reasons. Efforts are being made to replace them

with something better. XML-Schemas are a good candidate.

**Syntax of a DTD**

A DTD defines the structure of a tree. For each element of the tree, it will give the possible attributes it can or must have, and the possible sons it may have, in what order, and in what number.

The element of the document is not given in the DTD but in the header of the document.

A DTD may be referred to, in the header of the document, by a *SYSTEM* identifier, or by a *PUBLIC* identifier, which must then be defined in one of the catalog files used (either in command line or in the appropriate environment variable).

Choosing between attributes and subelements is not always easy. Choosing to use layers of wrapping elements or to use a flatter organization may as well influence maintenance and development. Big DTDs use entities and families of elements.

There are two types of elements: block elements, and inline elements.

**Example: the DTD of this presentation**

```
<!ENTITY % my-entities SYSTEM "../entities/all_entities">
%my-entities;

<!ELEMENT presentation (title, conference, place, date, au-
thor,
                        abstract, slide+)>
<!ELEMENT title        (#PCDATA|emph|kbd|code|strong|file|ulink)*>

<!ELEMENT author       (firstname, nickname?, middle-
name?, lastname, email?)>
<!ELEMENT firstname    (#PCDATA)>

<!ELEMENT abstract     (#PCDATA|emph|kbd|code|strong|file|ulink)*>
```

```
<!ELEMENT slide      (title,content)>
<!ELEMENT content    ((para|itemizedlist|screen|more)+)>

<!ELEMENT ulink      (#PCDATA|emph|kbd|code|strong|file|ulink)*>
<!ATTLIST ulink      url CDATA #IMPLIED>
```

**When to stop tagging?**

- Defining inline tags is important: all your needs should be covered.
- The library of Congress distinguishes between keywords and subjects, for example.
- the more you tag, the longer it will take to produce your document. If you use many different inline documents, changes will not be noticeable in the fonts used.
- DTDs and stylesheets are interesting if used in a coherent family of documents, like in a website. The more inline elements, the less chances are you will remain coherent, yielding to false negatives.
- However, a minimal tagging may be useful (see personal anecdote in full version)

### Encoding issues

The standard, default encoding for XML and XML tools is UTF-8<sup>a</sup>, one of the Unicode transformation formats. UTF-8 works as follows:

The binary representation of the character's integer value is thus simply spread across the bytes and the number of high bits set in the lead byte announces the number of bytes in the multibyte sequence:

bytes	bits	representation
1	7	0vvvvvvv
2	11	110vvvvv 10vvvvvv
3	16	1110vvvv 10vvvvvv 10vvvvvv
4	21	11110vvv 10vvvvvv 10vvvvvv 10vvvvvv

---

<sup>a</sup><http://czyborra.com/utf/>

**Lab on Concept of XML**

In the laboratory session following up this presentation, you will do the following things:

- Learn how to check the well-formedness and/or validity of an XML document, and interpret the mistakes
- Edit and create by hand simple XML documents
- Edit and create with more advanced tools simple XML documents
- Come up with a DTD for a specific and simple application
- Create from data given in another format, valid XML documents for that DTD