TEI By Example

TEI by Example. Module 0

Edward Vanhoutte

Ron Van den Branden

Melissa Terras

Centre for Scholarly Editing and Document Studies (CTB) , Royal Academy of Dutch Language and Literature, Belgium, Gent, 9 July 2010

Last modified on: 2018-04-28

TABLE OF CONTENTS

4. XML: Ground Rules..... 1
4.1 Recommendation..... 1
4.2 Components..... 1
4.2.1 XML declaration..... 2
4.2.2 Elements and Attributes..... 2
4.2.3 Entity References..... 2
4.2.4 (P)CDATA..... 5
4.3 Using XML..... 5

4. XML: Ground Rules

4.1 Recommendation

XML is a metalanguage by which one can create separate markup languages for separate purposes. It is platform-, software-, and system-independent and no one 'owns' XML, although specific XML markup languages can be owned by its creators. Generally speaking, XML empowers the content provider and facilitates data integration, exchange, maintenance, and extraction. XML is currently the de facto standard on the World Wide Web partly because HTML (Hypertext Markup Language) was rephrased as an XML encoding language. XML is edited and managed by the W3C which also published the specification as a recommendation in 1998. [1]

The big selling point of XML is that it is text based. This means that each XML encoding language is entirely built up in ASCII (American Standard Code for Information Interchange), or plain text, and can be created and edited using a simple text-editor like Notepad or its equivalents on other platforms. However, when you start working with XML, you will soon find that it is better to edit XML documents using a professional XML editor. While plain text-editors don't know that you're writing TEI, XML editors will help you to write error-free XML documents, validate your XML against a DTD or a schema, force you to stick to a valid XML structure, and enable you to perform transformations.

Since ASCII only provides for characters commonly found in the English language, different character encoding systems have been designed such as Isolat-1 (ISO-8859-1) for Western languages and Unicode (UTF-8 and UTF-16). By using these character encoding systems, non-ASCII characters such as French é, à, ç, Norwegian æ ø å, or Hebrew # can be used in XML documents. These systems rely on ASCII notation for the expression of these non-ASCII characters. The French à, for instance, is represented by the string 'agrave' in Isolat-1 and by the number '00E0' in Unicode.
[2]

4.2 Components

Any XML encoding language consists of five components.

- Processing Instructions
- Elements
- Attributes (optional)
- Entity References

.....

1 Bray, Tim; Paoli, Jean; Sperberg-McQueen, C.M. Extensible Markup Language (XML) 1.0.W3C Recommendation 10-February-1998. http://www.w3.org/TR/1998/REC-xml-19980210 (accessed September 2008)

2 See Section 1. Editors in the TBE Toolkit.

- (P)CDATA

For example, a simple two-paragraph document could be encoded as follows in XML:

```
<?xml version="1.0" encoding="UTF-8"?> <document>
  <!-- paragraphs go here -->
  <paragraph number="1">Paragraph one of <title>an XML example</title>.</paragraph>
  <paragraph number="2">Paragraph two of this example.</paragraph>
</document>
```

4.2.1 XML declaration

An XML document is introduced by the XML declaration.

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

The question mark `?` in the XML declaration signals that this is a processing instruction. The following bits state that what follows is XML which complies with version 1.0 of the recommendation and the used character encoding is UTF-8 or Unicode.

4.2.2 Elements and Attributes

The two-paragraph document above is an example of an XML document, representing both information and meta-information. Information (plain text) is contained in XML elements, delimited by start tags (e.g. **<document>**) and end tags (e.g.**</document>**). Additional information to these XML elements can be given in attributes, consisting of a name (e.g. **@number**) and a value (e.g. `1`). XML comments are delimited by start markers (`<!--`) and end markers (`-->`).

4.2.3 Entity References

Entity references are predefined strings of data that a parser must resolve before parsing the XML document. [3] Entity references may be useful in a number of cases:

- representing character data which cannot easily be keyboarded or which is illegal in XML because some characters are reserved.
- escaping illegal characters in XML

.....

3 A parser is a piece of software that recognises a programming or an encoding language with the possible intent to process or interpret it. An XML parser can for instance be used to validate an XML document, transform it to another format, or process information from the document.

- providing 'boilerplate text ', that is text which is or can be reused in new contexts or applications

An entity reference starts with an ampersand & and closes with a semicolon ;. The entity name is the string between these two symbols. For instance, the entity reference for the less than sign < is < the entity reference for the ampersand is &.

Not all computers support the Unicode encoding scheme XML works with. Portability of individual characters from the Unicode system, however, is supported by entity references that refer to their numeric or hexadecimal notation. For example, the character ø is represented within an XML document as the Unicode character with hexadecimal value 00F8 and decimal value 0248. For exporting an XML document containing this character, it may be represented by the character (or entity) reference ø or ø respectively, with the 'x' indicating that what follows is a hexadecimal value. References of this type do not need to be predefined, since the underlying character encoding for XML is always the same.

For legibility purposes, however, it is also possible to refer to this character by use of a mnemonic name, such as oslash., provided that each such name is mapped to the required Unicode value by means of an ENTITY declaration.

```
<!ENTITY oslash "#x00F8">
```

The ENTITY declaration uses a non-XML syntax inherited from SGML and starts with an opening delimiter < followed by an exclamation mark ! signalling that this is a declaration. The keyword ENTITY names that an entity is being declared here. What follows next is the entity name - here the mnemonic name ø - for which a declaration is given and the declaration itself inside quotation marks. In this example, it is the hexadecimal value of the character.

The same character can also be declared in the following ways.

```
<!ENTITY oslash "ø"> <!ENTITY oslash "#0248">
```

Character entities must also be used in XML to escape the less than sign < and the ampersand & which are illegal because the XML parser could mistake them for markup.

```
<p>Gimme pepper &amp;amp; salt!</p>
<p>A &amp;lt; B</p>
```

Entities are not only capable to refer to character declarations but can also refer to strings of text with an unlimited extent. This way repetitive keying of repeated information can be avoided (aka string substitution), or standard expressions or formulae can be kept up to date. The first is

useful, for instance, for the expansion of &TBE; to "TEI by Example" before the test is validated. This corresponding ENTITY declaration is as follows:

```
<!ENTITY TBE "TEI by Example">
```

The second is used in contracts, books of laws etc. in which updating would otherwise mean the complete rekeying of the same (extensive) string of text. For example, the expression "This contract is concluded between &party1; and &party2; for the duration of 10 years starting from &date;" in legal texts can be updated simply by changing the value of the ENTITY declarations:

```
<!ENTITY party1 "Rev Knyff "> <!ENTITY party2 "Lt Rosen"> <!ENTITY date "2010-01-01">
```

The substitution of the entities by their values in the given example results in the following expression "This contract is concluded between Rev Knyff and Lt Rosen for the duration of 10 years starting from 2007-01-01"
[4]

ENTITY declarations are placed inside a DOCTYPE declaration which follows the XML declaration at the beginning of the XML document.

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE rootelement [ <!ENTITY party1 "Rev Knyff "> <!
ENTITY party2 "Lt Rosen"> <!ENTITY date "2010-01-01"> ]>
```

Since the DOCTYPE declaration is a processing instruction, it starts with the opening delimiter `<!` which is followed by the keyword `DOCTYPE`. Next part is the name of the root element of the document. In the case of a TEI document, this will be `TEI`. The entity references which must be interpreted by the XML processor are put inside square brackets. An XML parser encountering this DOCTYPE declaration will expand the entities with the values given in the ENTITY declaration before the document itself is validated.

.....

4 The term boilerplate text dates back to the early 1900s, referring to the thick, tough steel sheets used to build steam boilers. From the 1890s onwards, printing plates of text for widespread reproduction such as advertisements or syndicated columns were cast or stamped in steel (instead of the much softer and less durable lead alloys used otherwise) ready for the printing press and distributed to newspapers around the United States. They came to be known as 'boilerplates'.

4.2.4 (P)CDATA

All text in an XML document will normally be parsed by a parser. When an XML element is parsed, the text between the XML tags is also parsed. The parser does that because XML elements can nest as in the following example:

```
<paragraph number="1">Paragraph one of <title>an XML example</title>.</paragraph>
```

The XML parser will break this string up into an element **<paragraph>** with a subelements **<title>**. Text data that will be parsed by an XML parser is called parsed character data or PCDATA.

An XML document often contains data which need not be parsed by an XML parser. For instance, characters like `<` and `&` are illegal in XML elements because the parser will interpret them as the beginning of new elements or the beginning of an entity reference which will result in an error message. Therefore, these characters can be escaped by the use of the entity references `<` and `&`. When programming or scripting code, like Javascript that contains many occurrences of `<` and `&;` is included in an XML document, it should not be parsed by the XML parser. We can avoid this by treating it as unparsed character data or CDATA in the document:

```
<script> <![CDATA[ function matchwo(a,b) { if (a < b && a < 0) then { return 1; } else { return 0; } } ]]>
</script>
```

A CDATA section starts with `<![CDATA[` and ends with `]]>`. Everything inside a CDATA section is ignored by the parser.

4.3 Using XML

Depending on the nature of your XML documents and what you want to use them for, you will need different tools, ranging from freely available open source tools to highly priced industrial software. In principle, the simplest plain text editor suffices to author or edit XML. In order to validate or transform XML, additional tools will be needed which often come included in dedicated XML editors: validating parser, XSLT processor, tree-structure viewer etc. For publishing purposes, XML documents may be transformed to other XML formats, HTML or PDF - to name just a few of the possibilities - using XSLT and XSLFO scripts which are processed by an off the shelf or custom made XSL processor. These published documents can be viewed in generic web browsers or PDF viewers where it considers transformations to HTML or PDF. XML documents can further be indexed, excerpted, questioned and analysed with tools specifically designed for the job.