XML Basics[1]

1 Introducing XML

1.1 What is XML?

XML stands for e**X**tensible **M**arkup **L**anguage. It is a markup language much like HTML, but there are several differences between them:

- HTML includes a collection of predefined tags that you can use right away in editing your HTML files, e.g. `<font>` and `<h1>`, but XML tags are not predefined.

  To use XML, users have to define their own tags for their specific application before using them. For example, to describe a note, `<note>`, `<to>`, `<from>`, `<heading>`, and `<body>` are defined in advance and used in a nested fashion to present the following XML file as a note:

  ```
  <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget the party!</body>
  </note>
  ```

  With XML, one can define whatever tags needed, which together compose a user-defined markup language similar to HTML, and then use the language to describe data. Specifically XML uses *Document Type Definition* (DTD) or an XML Schema to define tags. In this sense, XML is viewed as a meta-language since it can be used to define and describe a markup language instead of concrete data directly. That is also why it is called extensible.

- XML was designed to describe data while HTML was designed for displaying data.

  If you remember, HTML tags control the way data is presented to browser users, color, font size, spacing, etc. Differently XML aims to deal with the logic meaning of data, or semantics. In the above example, the text wrapped in `<from>` and `</from>` is the name

[1] This note is created based on the XML tutorial on http://www.w3schools.com/.

of the sender of the note. This enables the fulfillment of the task of finding all the notes written by a specific person. So XML was designed to describe data and to focus on what data is while HTML was designed to display data and to focus on how data looks.

Actually XML and HTML can complement each other. For example, we use XML files to store data on a web server machine and when a request arrives, a servlet runs to retrieve data in XML, compose a HTML file, and finally output it to the client. This way you can concentrate on using HTML for data layout and display, and be sure that changes in the underlying data will not require any changes to your HTML.

Besides XML's role of storing data, with XML, data can be exchanged between incompatible systems.

In the real world, computer systems and databases contain data in incompatible formats. One of the most time-consuming challenges for developers has been to exchange data between such systems over the Internet.

Converting the data to XML can greatly reduce this complexity and create data that can be read by many different types of applications, since XML data is stored in plain text format, which is software- and hardware-independent.

The best description of XML may be this: XML is a cross-platform, software and hardware independent tool for transmitting information. Since the creation of XML, it has been amazing to see how quickly the XML standard has been developed and how quickly a large number of software vendors have adopted the standard. It is strongly believed by the IT community at large that XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.

1.2 XML Syntax

The syntax rules of XML are very simple and very strict. The rules are very easy to learn, and very easy to use. Because of this, creating software that can read and manipulate XML is very easy to do.

1.2.1 An Example XML Document

XML documents use a self-describing and simple syntax. For example, the following is a complete XML file presenting a note:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
```

```
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The first line in the document – the XML declaration – defines the XML version and the character encoding used in the document. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

The next line and the last line describe the root element of the document (like it was saying: "this document is a note"). So when you look at the document, you easily detect this is a note to Tove from Jani. So XML is pretty self-descriptive.

1.2.2 Rigid Syntax

Different from HTML, XML has a syntax that is much more rigid.

- With XML, it is illegal to omit the closing tag.

  In HTML some elements do not have to have a closing tag to be able to present content in the way the user wants them to. For example:

  ```
  <p>This is a paragraph
  <p>This is another paragraph
  ```

  In XML, however, all elements must have a closing tag, like this:

  ```
  <p>This is a paragraph</p>
  <p>This is another paragraph</p>
  ```

  Note that you might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself. It is not an XML element, and it should not have a closing tag.

- Unlike HTML, XML tags are case sensitive.

  With XML, the tag `<Letter>` is different from the tag `<letter>`.

  Opening and closing tags must therefore be written with the same case:

  ```
  <Message>This is incorrect</message>
  <message>This is correct</message>
  ```

- Improper nesting of tags makes no sense to XML.

  In HTML some elements can be improperly nested within each other and still display content in the desired way like this:

```
<b><i>This text is bold and italic</b></i>
```

In XML all elements must be properly nested within each other like this:

```
<b><i>This text is bold and italic</i></b>
```

- All XML documents must contain a single tag pair to define a root element.

  All other elements must be within this root element. All elements can have sub elements (child elements). Sub elements must be correctly nested within their parent element:

  ```
  <root>
    <child>
      <subchild>.....</subchild>
    </child>
  </root>
  ```

- With XML, it is illegal to omit quotation marks around attribute values.

  XML elements can have attributes in name/value pairs just like in HTML. In XML the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <note date=12/11/2002>
  <to>Tove</to>
  <from>Jani</from>
  </note>
  ```

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <note date="12/11/2002">
  <to>Tove</to>
  <from>Jani</from>
  </note>
  ```

  The error in the first document is that the date attribute in the note element is not quoted.

- With XML, the white space in your document is not truncated.

  This is unlike HTML. With HTML, a sentence like this:

  ```
  Hello              my name is Tove,
  ```

  will be displayed like this:

```
Hello my name is Tove,
```

because HTML strips off the white space.

- With XML, CR/LF is converted to LF.

  Do you know what a typewriter is? Well, a typewriter is a mechanical device used in the previous century to produce printed documents. :-)

  After you have typed one line of text on a typewriter, you have to manually return the printing carriage to the left margin position and manually feed the paper up one line.

  In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF). The character pair bears some resemblance to the typewriter actions of setting a new line. In Unix applications, a new line is normally stored as a LF character. Macintosh applications use only a CR character to store a new line.

- Comments in XML The syntax for writing comments in XML is similar to that of HTML.

  ```
  <!-- This is a comment -->
  ```

However as you see, there is nothing special about XML. It is just plain text with the addition of some XML tags enclosed in angle brackets.

Software that can handle plain text can also handle XML. In a simple text editor, the XML tags will be visible and will not be handled specially.

In an XML-aware application, however, the XML tags can be handled specially. The tags may or may not be visible, or have a functional meaning, depending on the nature of the application.

1.3 XML Elements

XML elements define the framework of an XML document and the structure of data items.

1.3.1 XML Elements are Extensible

XML was invented to be extensible and XML documents can be extended to carry more information.

Take the above note as an example again. Let's imagine that we created an application that extracted the `<to>`, `<from>`, and `<body>` elements from the XML document to produce this output:

```
MESSAGE
To: Tove
From: Jani

Don't forget me this weekend!
```

Imagine that the author of the XML document added some extra information to it:

```
<note>
<date>2002-08-01</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash? No. The application should still be able to find the `<to>`, `<from>`, and `<body>` elements in the XML document and produce the same output.

XML documents are Extensible.

1.3.2 XML Elements have Relationships

Elements are related as parents and children.

To understand XML terminology, you have to know how relationships between XML elements are named, and how element content is described.

Imagine that this is a description of a book:

`My First XML`

`Introduction to XML`

- `What is HTML`
- `What is XML XML`

`Syntax`

- `Elements must have a closing tag`
- `Elements must be properly nested`

Imagine that this XML document describes the book:

```
<book>
<title>My First XML</title>
<prod id="33-657" media="paper"></prod>
<chapter>Introduction to XML
<para>What is HTML</para>
<para>What is XML</para>
</chapter>
<chapter>XML Syntax
<para>Elements must have a closing tag</para>
<para>Elements must be properly nested</para>
</chapter>
</book>
```

`book` is the root element. `title`, `prod`, and `chapter` are child elements of `book`. `book` is the parent element of `title`, `prod`, and `chapter`. `title`, `prod`, and `chapter` are siblings (or sister elements) because they have the same parent.

1.3.3 Elements have Content

Elements can have different content types.

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can have *element content*, *mixed* content, *simple* content, or *empty* content. An element can also have *attributes*.

In the example above, `book` has element content, because it contains other elements. `chapter` has mixed content because it contains both text and other elements. `para` has simple content (or text content) because it contains only text. `prod` has empty content, because it carries no information.

In the example above only the `prod` element has attributes. The attribute named id has the value "`33-657`". The attribute named `media` has the value "`paper`".

1.3.4 Element Naming

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names must not start with a number or punctuation character
- Names must not start with the letters xml (or XML or Xml ..)

- Names cannot contain spaces

Take care when you "invent" element names and follow these simple rules:

Any name can be used, no words are reserved, but the idea is to make names descriptive. Names with an underscore separator are nice, for example `<first_name>` and `<last_name>`.

Avoid "-" and "." in names. For example, if you name something "`first-name`," it could be a mess if your software tries to subtract `name` from `first`. Or if you name something "`first.name`," your software may think that "`name`" is a property of the object "first."

Element names can be as long as you like, but don't exaggerate. Names should be short and simple, like this: `<book_title>` not like this: `<the_title_of_the_book>`.

XML documents often have a corresponding database, in which fields exist corresponding to elements in the XML document. A good practice is to use the naming rules of your database for the elements in the XML documents.

Non-English letters like òç are perfectly legal in XML element names, but watch out for problems if your software vendor doesn't support them.

The ":" should not be used in element names because it is reserved to be used for something called namespaces.

1.4 XML Attributes

XML elements can have attributes in the start tag, just like HTML. Attributes are used to provide additional information about elements.

From HTML you will remember this: `<IMG SRC="computer.gif">`. The `SRC` attribute provides additional information about the `IMG` element.

In HTML (and in XML) attributes provide additional information about elements:

```
<img src="computer.gif">
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

As we said before, attribute values in XML must always be enclosed in quotes, but either single or double quotes can be used. Note that if the attribute value itself contains double quotes it is necessary to use single quotes and vice versa, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

Sometimes information may be placed as attributes or child elements. Take a look at these examples:

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

and

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

In the first example `sex` is an attribute. In the last, `sex` is a child element. Both examples provide the same information.

There are no rules about when to use attributes, and when to use child elements. Generally attributes are handy in HTML, but in XML you should try to avoid them. Use child elements if the information feels like data.

Here are some of the problems using attributes:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use elements to describe data. Use attributes only to provide information that is not relevant to the data.

Don't end up like this (if you think this looks like XML, you have not understood the point):

```
<note day="12" month="11" year="2002" to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!"> </note>
```

1.5 XML Validation

Similar to HTML, XML with correct syntax is *Well Formed XML*. That is a well formed XML document is a document that conforms to the XML syntax rules that were described in the previous sections.

More specifically, to be well formed, an XML document must be validated against a *Document Type Definition* (DTD). The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. a DTD can be specified internally or externally. The following is an example of internal DTD for the above note example:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this: `!DOCTYPE note` (in line 2) defines that this is a document of the type `note`. `!ELEMENT note` (in line 3) defines the `note` element as having four elements: "`to`,`from`,`heading`,`body`". `!ELEMENT to` (in line 4) defines the to element `to` be of the type "`#PCDATA`". `!ELEMENT from` (in line 5) defines the from element to be of the type "`#PCDATA`" and so on ...

If the DTD is external to your XML source file, it should be wrapped in a `DOCTYPE` definition with the following syntax:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

And the following is a copy of the file "`note.dtd`" containing the DTD:

```
<!ELEMENT note (to,from,heading,body)>
```

```
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

W3C supports an alternative to DTD called *XML Schema*. If interested, you may read more about XML Schema in related books.

The W3C XML specification states that a program should not continue to process an XML document if it finds a validation error. The reason is that XML software should be easy to write, and that all XML documents should be compatible.

With HTML it was possible to create documents with lots of errors (like when you forget an end tag). One of the main reasons that HTML browsers are so big and incompatible, is that they have their own ways to figure out what a document should look like when they encounter an HTML error.

With XML this should not be possible.

1.6 Viewing XML Files

To view an XML document in IE 5.0 (and higher) you can click on a link, type the URL in the address bar, or double-click on the name of an XML file in a files folder. If you open an XML document in IE, it will display the document with color coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. If you want to view the raw XML source, you must select ”View Source” from the browser menu.

To view an XML document in Netscape 6, you'll have to open the XML file and then right-click in XML file and select “View Page Source”. If you open an XML document in Netscape 6, it will display the document with color coded root and child elements.

If an erroneous XML file is opened, the browser will report the error.

Since XML tags are “invented” by the author of the XML document, browsers do not know if a tag like `<table>` describes an HTML table or a dining table.

Without any information about how to display the data, most browsers will just display the XML document as it is. In the next section, we will take a look at different solutions to the display problem, using CSS and XSL.

1.7 Displaying XML with CSS

Before we have learned that CSS files may work together with HTML files in the way that the former is in charge of display and the latter provides concrete information. CSS can do the same thing with XML.

Below is a fraction of the XML file, with an added CSS style sheet reference. The second line, `<?xml-stylesheet type="text/css" href="cd_catalog.css"?>`, links the XML file to the CSS file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  .
  .
  .
  .
</CATALOG>
```

The CSS file `cd_catalog.css` goes as follows:

```
CATALOG {
  background-color: #ffffff;
  width: 100%;
}

CD {
  display: block;
  margin-bottom: 30pt;
  margin-left: 0;
}

TITLE {
  color: #FF0000;
  font-size: 20pt;
}

ARTIST {
  color: #0000FF;
  font-size: 20pt;
}

COUNTRY,PRICE,YEAR,COMPANY {
  display: block;
  color: #000000;
  margin-left: 20pt;
}
```

where it is specified how to display each kind of elements.

1.8 Displaying XML with XSL

Besides CSS, XSL was invented just for displaying XML. The *e**X**tensible **S**tylesheet **L**anguage* (XSL) is far more sophisticated than CSS.

XSL consists of three parts:

- XSLT (XSL Transformations) is a language for transforming XML documents.

  XSLT is the most important part of the XSL Standards. It is the part of XSL that is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

  XSLT can also add new elements into the output file, or remove elements. It can rearrange and sort elements, and test and make decisions about which elements to display, and a lot more.

  A common way to describe the transformation process is to say that XSLT transforms an XML source tree into an XML result tree.

  XSLT uses XPath to define the matching patterns for transformations. In the transformation process, XSLT uses XPath to define parts of the source document that match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. The parts of the source document that do not match a template will end up unmodified in the result document.

- XPath is a language for defining parts of an XML document.

  XPath uses path expressions to identify nodes in an XML document. These path expressions look very much like the expressions you see when you work with a computer file system:

  ```
  w3schools/xpath/default.asp
  ```

  Look at the following simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?> <catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

The XPath expression below selects the ROOT element catalog:

```
/catalog
```

The XPath expression below selects all the cd elements of the catalog element:

```
/catalog/cd
```

The XPath expression below selects all the price elements of all the cd elements of the catalog element:

```
/catalog/cd/price
```

Note: If the path starts with a slash ('/') it represents an absolute path to an element!

XPath also defines a library of standard functions for working with strings, numbers and Boolean expressions. The XPath expression below selects all the cd elements that have a price element with a value larger than 10.80:

```
/catalog/cd[price>10.80]
```

- XSL-FO is a language for formatting XML documents.

Think of XSL as set of languages that can transform XML into XHTML, filter and sort XML data, define parts of an XML document, format XML data based on the data value, like displaying negative numbers in red, and output XML data to different media, like screens, paper, or voice.

One way to use XSL is to transform XML into HTML before it is displayed by the browser. Below is a fraction of the XML file, with an added XSL reference. The second line, `<?xml-stylesheet type="text/xsl" href="simple.xsl"?>`, links the XML file to the XSL file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
       two of our famous Belgian Waffles
    </description>
    <calories>650</calories>
  </food>
</breakfast_menu>
```

And the corresponding XSL file is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v4.2 -->
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/TR/xhtml1/strict">
  <body style="font-family:Arial,helvetica,sans-serif;font-size:12pt;
        background-color:#EEEEEE">
    <xsl:for-each select="breakfast_menu/food">
      <div style="background-color:teal;color:white;padding:4px">
        <span style="font-weight:bold;color:white">
        <xsl:value-of select="name"/></span>
        - <xsl:value-of select="price"/>
      </div>
      <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
        <xsl:value-of select="description"/>
        <span style="font-style:italic">
          (<xsl:value-of select="calories"/> calories per serving)
        </span>
      </div>
    </xsl:for-each>
  </body>
</html>
```

2 Programming with XML

Since Java has been commonly used for processing XML files, this section presents one popular Java-based method of parsing XML: the *Document Object Model* (DOM).

DOM represents an entire XML document in a tree-like data structure that can be easily manipulated by a Java program. The advantage of DOM is that it is relatively simple to use and you can modify the data structure in addition to extracting data from it. However, the disadvantage is that DOM always parses and stores the entire document, even if you only care about part of it. The *Simple API for XML* (SAX) is an alternative to avoid this problem.

To use DOM, you need have a DOM-compliant parser. A list of such parsers are given at http://www.xml.com/pub/rg/Java_Parsers. Besides, the *Java API for XML Processing* (JAXP) is needed and available from http://java.sun.com/. This API provides a small layer on top of DOM that lets you plug in different vendor's parsers without making any changes to your basic code.

The use of DOM goes as follows:

1. Tell the system which parser you want to use. There are many ways to do so, of which the system property is the easiest method. For example the following code permits users to specify the parser on the command line with `-D` option to `java`, and uses the Apache Xerces parser otherwise.

```
public static void main(String args[]) {
  String jaxPropertyName = "javax.xml.parsers.DocumentBuilderFactory";
  if (System.getProperty(jaxPropertyName) == null) {
    String apacheXercesPropertyValue =
      "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl";
    System.setProperty(jaxPropertyName, apacheXercesPropertyValue);
  }
  ...
}
```

2. Create a JAXP document builder. This is basically a wrapper around a specific XML parser.

```
DocumentBuilderFactory builderFactory =
  DocumentBuilderFactory.newInstance();
DocumentBuilder builder =
  builderFactory.newDocumentBuilder();
```

3. Invoke the parser to create a `Document` representing an XML document.

```
Document document = builder.parse(someInputStream);
```

4. Normalize the tree.

```
document.getDocumentElement().normalize();
```

5. Obtain the root node of the tree. This returns `Element`, which is a subclass of the more general `Node` class that represents an XML element.

```
Element rootElement = document.getDocumentElement();
```

6. Examine various properties of the node. Various methods are available for access: `getNodeName()`, `getNodeValue()`, `getAttributes()`, `getChildNodes()`, etc..

As you may image, you may simply replace the `Property` format configuration file for your own web server with an XML one, and use the above process to access the configuration information.