XML Tutorial

Learn About Key e-Commerce Trends and Technologies at Your Own Pace.

A GXS Tutorial for the Active Business

GXS

XML—this primer on Extensible Markup Language (XML) will help you understand why this new language is being called the 'lingua franca' of the Internet.

Welcome

This tutorial is the first in a series that provides information about Extensible Markup Language, or XML.

The topics covered in this module are:

- The development of XML
- The basic concepts to structure, create and read XML code
- Relationship of XML to other markup languages and applications

How To Use This Tutorial

This tutorial provides information about basic XML concepts. It should be used as a prerequisite to understanding what XML is, and how XML is developing to meet the need of providing electronic commerce solutions.

It is only the beginning. This tutorial will introduce terms and concepts that you will find necessary to form an awareness of XML. Future modules will build upon this knowledge, allowing you to create, read and interpret XML documents.

The best way to use this tutorial is to read through a module and its subtopics, though not necessarily at one sitting. This tutorial does build on information presented in earlier modules, but individual subtopics can be used as reference outside of the linear progression of the course. You will find self-checks at the end of each module so you can evaluate your understanding of the material.

Tutorial Objectives

After completing this tutorial, you should be able to discuss:

- What is XML
- How and why XML was developed
- What is the basic structure of an XML document
- How XML documents are read
- What are the components of an XML application
- How XML may be used in providing electronic commerce solutions

What is XML?

Extensible Markup Language (XML) is a way to apply structure to a web page. XML provides a standard open format and mechanisms for structuring a document so that it can be exchanged and manipulated. Here is an example of an XML document:

```
<?xml version=”1.0”?>
<basket>A Fruit Basket
<oranges>navel</oranges>
<apples>granny smith</apples>
<peaches>red haven</peaches>
<grapes>concord</grapes>
</basket>
```

If you are familiar with HyperText Markup Language (HTML), you will be pleased to learn that XML is not going to replace everything you've already learned. In fact, XML complements your HTML knowledge by allowing you to structure your data by “marking up” the text and data to “define” the data content.

Like HTML, XML uses tags to “mark up” data content. Unlike HTML, in XML you define your own tags that meet the exact needs of your document. The custom tags make your data easier to organize and search.

XML will not change the way your web pages look, but it will change the way the documents are read, and the way documents are filed and stored.

XML History 101

The concept of XML is over 30 years old, beginning in the 1960's. Its origins are in the standardized typesetting codes, GENCODE, used by the publishing industry.

In the 1970's, Dr. C. F. Goldfarb proposed a method of describing text that was not specific to an application or hardware. He created Generalized Markup Language (GML). The basic tenents of GML were:

- Markup should emphasize the document structure, not format or style.
- Simple input syntax should be used for markup using <> and </> tags.
- Markup syntax rules should be strictly controlled so that the code could be easily read by humans or software programs.

Originally, the number of document types supported by GML was limited, so the addition of any new tags and document types was relatively simple. By the 1980's, however, these numbers grew to such an extent that GENCODE and GML proponents formed the ANSI Committee on Computer Languages for the Processing of Text.

In 1986, this committee promulgated Standardized Generalized Markup Language (SGML), which standardized the use use of <> and </> tags, as well as Document Type Definitions (DTD). As with GENCODE and GML, the primary use of SGML was for large-scale publishing.

As interest in the Internet grew, and the functionality of Internet browsers evolved, the need for a standardized hypertext application increased. In the early 1990's, the World Wide Web Consortium (W3C) adopted HyperText Markup Language (HTML) as the standard. HTML is a subset of SGML because it borrowed existing tags from SGML and DTD.

As web communities have grown, so has the need to publish new types of documents. Many of these documents are community specific. Unfortunately, HTML cannot be extended to accommodate new document types, and browsers will not support SGML. These needs prompted the W3C to sponsor the development of an "Extensible Markup Language."

XML Design Goals

The design goals for XML were proposed by the World Wide Web Consortium (W3C) and published in January 1998. A synopsis of these design goals is as follows:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents should be easy to create.
- Terseness in XML markup is of minimal importance.

In proposing these design goals, the W3C has endeavored to combine the sophistication of SGML with the ease of use offered by HTML.

XML offers a standard open format for communities to develop structured mechanisms for marking text and data to facilitate the exchange and manipulation of documents among the community members. The focus of XML is:

- To carry the power of SGML through its ability to create user-defined information about data, its ability to adapt to specific user needs, and the ability to maintain document changes.

- To carry HTML's ease of use by the ability to link, its simplicity in the use of user-defined tags, and its portablity among different platforms.

XML Today

At present, XML is still evolving. Here are some key points to remember:

- The primary functional purpose of XML is to transfer structured text and data among systems in multiple organizations.
- XML is a derivative of SGML, as is HTML.
- XML, unlike HTML does not have a fixed format. There are no pre-defined tags; you create your own.
- XML tags define the meta information (information about information) and are distributed throughout the document.
- Like HTML, XML uses tags. Tags are always enclosed within angled brackets (< >).

The Basics

It is important to remember that, like HTML, XML is a Markup Language.

What do we mean by 'markup language'?

The term 'markup' is used to identify anything put within a document which either adds or provides special meaning (e.g., italicized text).

A markup language is the set of rules. It declares what constitutes markup in a document, and defines exactly what the markup means. It also provides a description of document layout and logical structure.

There are three types of markup:

- **Stylistic:** How a document is presented (e.g., the HTML tags <I> for italics, <B> for bold, and <U> for underline)
- **Structural:** How the document is to be structured (e.g., the HTML tags <P> for paragraph, <SPAN> for creating ad hoc styles in a document, and <DIV> for grouping structures aligned in the same way.
- **Semantic:** Tells about the content of the data (e.g., the HTML tags <TITLE> for page title, <HEAD> for page header information, and <SCRIPT>to indicate a JavaScript in a page)

In XML, the only type of markup that we are concerned with is structural.

When creating an XML document, you must begin with the XML declaration statement. This statement alerts the browser or other processing tools that this document contains XML tags. The declaration looks like this:

```
<?xml version=”1.0”?>
```

This is the first line of your document.

Tags, Elements, Attributes

To define words, you mark up pages with descriptive tags that you create. In theory, you do not need to describe or explain what these tags mean. However, in order for others to use your data effectively, you will have to define and agree upon the meaning of each tag.

Tags carry the smallest unit of meaning signifying structure, format or style of the data. They are always enclosed within angled brackets ‘< >’. Tags are case-sensitive. This means that the tags <basket>, <Basket>, <BASKET> carry different meanings and cannot be used interchangeably. In addition, all tags must be paired so that they have a start <basket> and an end </basket>. Tags combined with data form elements.

Elements are the building blocks of a document. An element consists of a start-tag, an end-tag and the content between them:

```
<basket>A Fruit Basket</basket>
```

It may seem odd that a single element contains all the data in the XML document. However, within this single element there may be multiple levels of nested sub-elements which keep the individual pieces of data in a logical and easy to manage structure:

```
<?xml version=”1.0”?>
<basket>A Fruit Basket
<oranges>navel</oranges>
<apples>granny smith</apples>
<peaches>red haven</peaches>
<grapes>concord</grapes>
</basket>
```

The organization of elements in this fashion is called nesting. Element nesting creates parent/child relationships, and every child element resides completely within its parent element.

In the example, <basket> is the parent of <oranges>, <apples>, <peaches> and <grapes>. In turn, <oranges>, <apples>, <peaches> and <grapes> are siblings. They could also be parents if sub-elements were identified and resided within the appropriate tags.

In our example, we have identified the tag <basket> as the document element, because it is the highest level of the hierarchy. It is not the child of any element. In the structure of a document there can only be one document element.

To any element you may want to add attributes. An attribute gives information about an element. Attributes are embedded in the element start tag. An attribute consists of an attribute name and and attribute value. The attribute name precedes its value, and they are separated by an equal sign. Also, the attribute value is enclosed in quotes to delimit multiple attributes in the same element.

Let's add the attribute 'count=' to <oranges>, <apples>, <peaches> and <grapes> to tell us how many of each we have in the basket. Our example would look like this:

```
<?xml version="1.0"?>
<basket>A Fruit Basket
<oranges count="5">navel</oranges>
<apples count="3">granny smith</apples>
<peaches count="2">red haven</peaches>
<grapes count="1">concord</grapes>
</basket>
```

Document Type Definitions

Up until now, we have been creating "well-formed" XML documents. That means that we have been creating tags, elements and attributes in a correct nesting structure without really providing further definitions. In order to qualify as an XML document, all documents must be at least well-formed.

In addition to well-formed documents, there are "valid" XML documents. This means the documents follow a more formal structure. The main difference between well-formed XML and valid XML is the Document Type Definition (DTD). The DTD is a set of rules that define the elements that may be used, and where they may be applied in relation to each other.

If a DTD is not in use, it is impossible to know for certain that an element has only element content. In our example:

```
<?xml version="1.0"?>
<basket>A Fruit Basket
<oranges count="5">navel</oranges>
<apples count="3">granny smith</apples>
<peaches count="2">red haven</peaches>
<grapes count="1">concord</grapes>
</basket>
```

A human reader could deduce from the names of the tags and attributes the meaning of the elements, but software could not. The DTD is where we declare our specific element tags and attributes.

The DTD for our example would be written as follows:

```
<?xml version="1.0"?>
<!DOCTYPE Basket [
<!ELEMENT Basket
(Oranges|Apples|Peaches|Grapes|#PCDATA)+>
<!ELEMENT Oranges (#PCDATA)>
<!ATTLIST Oranges count CDATA #REQUIRED>
<!ELEMENT Apples (#PCDATA)>
<!ATTLIST Apples count CDATA #REQUIRED>
<!ELEMENT Peaches (#PCDATA)>
<!ATTLIST Peaches count CDATA #REQUIRED>
<!ELEMENT Grapes (#PCDATA)>
<!ATTLIST Grapes count CDATA #REQUIRED>
]>
```

The DTD can be either an external DTD or an internal DTD or both.

The external DTD exists outside the content of a document and carries the extension .dtd. This type of DTD could be created for use by a particular community, providing a standardized document format for all members. The DTD reference, added at the beginning of the XML file, tells the XML processor where to find the external DTD, information about its creator, the purpose of the DTD, and the language used:

```
<!DOCTYPE catalog PUBLIC "-//flowers//DTD Stan-
dard /EN" "http://www.fruitbaskets.com/dtd/frutbskt.
dtd">
The internal DTD is written directly in the XML
document:
<?xml version="1.0"?>
<!DOCTYPE Basket [
<!ELEMENT Basket
(Oranges|Apples|Peaches|Grapes|#PCDATA)+>
<!ELEMENT Oranges (#PCDATA)>
<!ATTLIST Oranges
count CDATA #REQUIRED>
<!ELEMENT Apples (#PCDATA)>
<!ATTLIST Apples
count CDATA #REQUIRED>
<!ELEMENT Peaches (#PCDATA)>
<!ATTLIST Peaches
count CDATA #REQUIRED>
<!ELEMENT Grapes (#PCDATA)>
<!ATTLIST Grapes
count CDATA #REQUIRED>
]>
<basket>A Fruit Basket
<oranges count="5">navel</oranges>
<apples count="3">granny smith</apples>
<peaches count="2">red haven</peaches>
<grapes count="1">concord</grapes>
</basket>
```

Reading DTDs

To read the DTD for our fruit basket example, let’s first view the file.

```
<?xml version=”1.0”?>
<!DOCTYPE Basket [
<!ELEMENT Basket
(Oranges|Apples|Peaches|Grapes|#PCDATA)+>
<!ELEMENT Oranges (#PCDATA)>
<!ATTLIST Oranges
count CDATA #REQUIRED>
<!ELEMENT Apples (#PCDATA)>
<!ATTLIST Apples
count CDATA #REQUIRED>
<!ELEMENT Peaches (#PCDATA)>
<!ATTLIST Peaches
count CDATA #REQUIRED>
<!ELEMENT Grapes (#PCDATA)>
<!ATTLIST Grapes
count CDATA #REQUIRED>
]>
<basket>A Fruit Basket
<oranges count=”5”>navel</oranges>
<apples count=”3”>granny smith</apples>
<peaches count=”2”>red haven</peaches>
<grapes count=”1”>concord</grapes>
</basket>
```

Here are the points to remember in reading the file:

- This is an internal DTD, so the DTD and the XML code are contained in one document. The file would have the extension .xml (e.g., basket.xml) If it was an external DTD there would be two files: the DTD with extension .dtd and the XML with the extension .xml. The external DTD would be referenced at the beginning of the XML file.

- The DTD says that basket contains oranges, apples, peaches and grapes. These are identified as the elements of basket, so these are the only tags that can appear in the XML document. <!ELEMENT basket (oranges|apples|peaches|grapes|#PCDATA)+> [

- There may be information content within the basket, oranges, apples, peaches and grapes tags. (#PCDATA):
  – Basket has information, the name, “A Fruit Basket”

  – Oranges has “navel” to denote the type of oranges

– Apples has “granny smith” to denote the type of apples

– Peaches has “red haven” to denote the type of peaches

– Grapes has “concord” to denote the type of grapes

- Oranges, apples, peaches and grapes have the attribute (!ATTLIST) value “count=”, which is used to tell us how many we have in the basket. The values for count must be in the XML document (#REQUIRED).

```
<!ATTLIST oranges
count CDATA #REQUIRED>
```

XML Style Sheets

We have discussed how XML allows you to create your own tags, how the meaning of these tags is defined by you, and how the definition you supply explains how to structure the contents of a document. Now we will examine what is required for XML documents to be processed.

XML can be applied in many different ways and for many different purposes. It can be pages viewed with an Internet browser. Perhaps it is used for computer-to-computer data exchange, or to provide database updates. Whatever the end result, it is necessary that XML-aware tools be available to process XML documents. Here is a list of some of the components that are necessary for an XML application to process XML documents:

- Document Type Definitions (DTD)
- XML-aware browsers
- Cascading Style Sheets (CSS) and/or Extensible Stylesheet Language (XSL)
- XML parsers
- Extensible Linking Language (XLL)
- Namespaces

As we have previously stated, XML does not describe how a document is to be displayed for human interpretation (i.e., how it will look on a computer screen, on paper, or through an audio device.) XML documents are plain text files. You use a text editor (Notepad, Wordpad, etc.) to create and view them, but you cannot format them using these tools.

In order to format and view an XML document, you must combine the document with a style sheet. The document can then be viewed in the appropriate browser. Currently, Internet Explorer and Netscape Communicator will support XML documents with associated style sheets. Browsers 4.x or lower (Internet Explorer, Communicator, or Navigator) cannot directly view formatted XML.

Style sheets contain the rules that declare how the data of an XML document should appear or be interpreted by the user agent (browser, printer, text-to-speech converter, etc.) This is done by assigning a style to a tag. The style is then applied to the data contained within the tag.

Style sheets can be written in several languages. Two of these are:

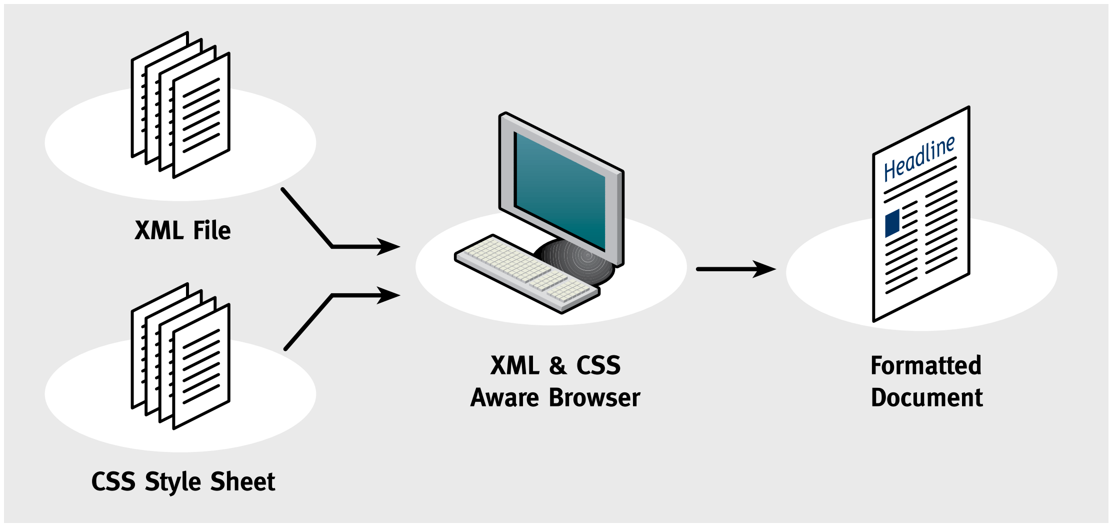
- Cascading Style Sheets (CSS), an extension of HTML
- Extensible Stylesheet Language (XSL), an XML-specific styling language

Let's look further into how these formatting languages work with XML.

Cascading Style Sheets

Cascading Style Sheets (CSS) are used to display an XML file in a consistent format. As a rule, CSS's are kept separate from the XML document. By using a browser capable of processing the XML file and the CSS, you will get a formatted document that you can view in the browser window. The browser is able to combine the XML document and format it according to the rules of the CSS:

Here is an example of an XML file (xmsampdoc.xml):

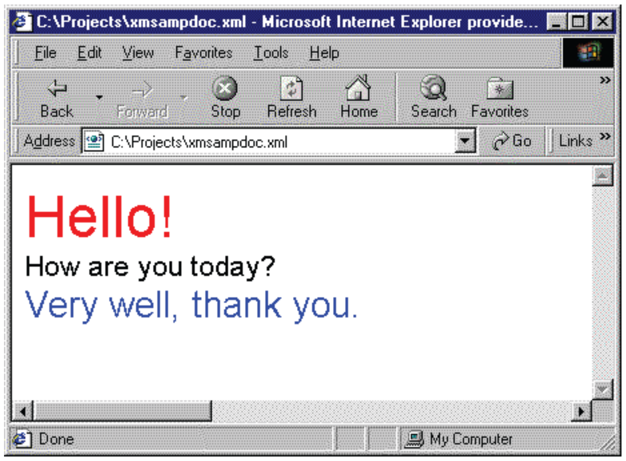


When Internet Explorer processes the xmsampdoc.xml and xmsampdoc.css files it:

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/css" href="xmsampdoc.
css"?>
<xsampdoc>
<greeting>Hello!</greeting>
<question>How are you today?</question>
<answer>Very well, thank you.</answer>
</xsampdoc>
Here is an example of the Cascading Style Sheet
(xmsampdoc.css) that will format the XML file:
xsampdoc
{
display:block;
font-family: Arial, Helvetica, sans-serif;
font-size: 32pt;
width: 30em;
color: red
}
question {
display:block;
font-size: x-large;
color: black;
}
answer {
display: block;
font-size: 20pt;
color: blue
}
```

- Parses xmsampdoc.xml into its elements
- Applies the styles and formats elements using xmsampdoc.css

This is what you see in the browser window:


C:\Projects\xmsampdoc.xml - Microsoft Internet Explorer provide...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address C:\Projects\xmsampdoc.xml Go Links
Done
My Computer

Hello!

How are you today?

Very well, thank you.

While CSS can format your XML document for display, there are a number of problems with using CSS and XML:

- The structure of the XML content must be virtually identical to the structure of the presentation.
- The order of the elements in the XML document is the way they are displayed.
- The CSS simply paints the document to the browser.
- If you change the document you must change the CSS.

Remember that one of XML's goals is to separate content from display. This is not entirely accomplished using CSS.

What we need is an XML processor that applies sophisticated transformations, allowing extensive reordering of elements, generated text, and calculations without modifications to the XML source. This is what Extensible Stylesheet Language (XSL) does.

XSL

Extensible Stylesheet Language (XSL) is a language for expressing style sheets specifically for use with XML. It consists of two parts:

- **Extensible Stylesheet Language Transformer (XSLT)**—a language for transforming XML documents. XSLT engines are typically used to convert XML into a strict form of HTML known as XHTML.
- **Formatting Objects (FO)**—an XML vocabulary for specifying formatting semantics. Formatting Objects are not widely implemented.

These two parts form the basis of the XSL style sheet that describes how the structured content of the XML data file should be displayed. This includes how the source content should be laid out and paginated onto some presentation medium: a web browser; a physical page in a book, report, pamphlet, memo; or in an alternative format such as a text-to-speech converter.

As with the CSS, an XSL style sheet is a template with rules that govern an XML document. The difference is that the XSL style sheet is itself an XML document. As an XML document, the XSL style sheet contains tags, elements and attributes.

The template rules of the XSL document has two parts:

- **Patterns:** used to select specific elements in the document
- **Actions:** used to describe what styling is to be applied to the element

By using an XSL processor, the formatting specified in the style sheet is applied to the XML document elements. Let's look at an example of an XML document and its XSL style sheet.

We'll use a one element XML document (greet.xml):

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

The XML file contains: 1 element, 1 tag (<greeting>,</greeting>), and no attributes.

```
<greeting>Hello! How are you?</greeting>
```

The XML file contains: 1 element, 1 tag (<greeting>,</greeting>), and no attributes.

The XSL file (greet.xsl) looks like this:

```
<xsl:stylesheet>
<xsl:template match="greeting">
<fo:block color = "red" font-size="16pt">
<xsl:apply-templates/>
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

The XSL file specifies the style of the tag <greeting>,</greeting>. It will format and display the content as 16 point font in red color.

Let's parse the XSL file to see exactly what it's telling the processor to do.

- <xsl:template starts the beginning of a pattern/action rule.
- match="greeting"> is the pattern that selects the element tag <greeting>.
- <fo:block color = "red" font-size="16pt"> is the formatting object that describes what styling is applied to the element tag.
- <xsl:apply-templates/> applies the format and styling to the text contained in the element tag <greeting>.

XSL has scripting, filtering, sorting and coping capabilities that give you everything you need to transform one document into another.

XML Parsers

Any system that uses XML requires two components:

- An XML processor and parser
- The system application (this could be a browser, an XSL processor, a document translator, or a program that updates a database).

The processor checks to make sure that the XML file follows the XML 1.0 specification; in other words, it makes sure that the file is well-formed XML. It is the job of the XML parser to read the XML file and create its document tree. A document tree is a hierarchical organization of the data content. The application then processes the data in the tree.

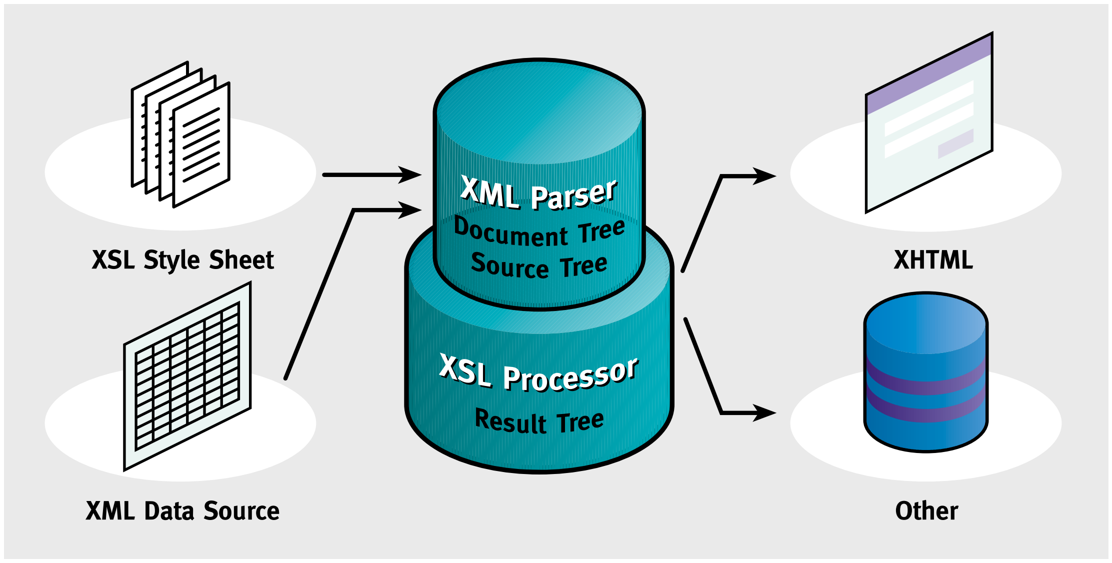
Most parsers are written in C++ and Java (object-oriented languages).

Parsers can also be used for checking the XML document's syntax and structure. This is done by checking the document's validity against its DTD (if available.) If the parser reads the file against the DTD description, it is known as a 'validating parser.'

Whether the parser checks against a DTD or not, the next step is to break the document into its elements and store them in the source tree. If an XSL processor is used, this data is then passed on to the XSL processor, which applies the formatting rules according to the template rules. This creates a result tree.

The result tree is then formatted, enhanced and manipulated by the user agent (browser, etc.), and displayed.

Here is an illustration of the process:



Namespaces

A namespace is a collection of names that can be used in XML documents as element or attribute names. They identify the name as being from a particular domain (standards group, company, industry, etc.)

Namespaces are identified in XML by a Uniform Resource Identifier (URI). The URI includes both a Uniform Resource Name (URN) and a Uniform Resource Locator (URL). URL's have become very common in the Internet world. The URN is a universally unique number or name that identifies something in a universally-unique way. While not as common as URL's, they will be used more as XML is adopted and used.

Why do we need to use namespaces? As people start reusing standard DTD's and extending them, there will be conflicts in the XML parser as documents are exchanged. If you extend a DTD using an element or attribute name that's already in the DTD, the parser won't know which one you're using. Namespaces help standardize and uniquely brand elements and attributes.

Namespaces employ the URI to instruct the user-agent (browser, XML parser, XML application, etc.) where to go to find the DTD against which the XML document is checked for validity.

We've already seen an example of a namespace in our example in the XSL overview.

```
<xsl:stylesheet>
<xsl:template match="greeting">
```

Since XSL is written in XML syntax (it uses tags, elements and attributes), it creates a potential conflict with the data source tags. XSL uses namespaces to distinguish elements that are instructions to the XSL processor from elements that specify literal result tree structure. XSL uses the prefix xsl: for elements in the XSL namespace:

```
<xsl:stylesheet>
<xsl:template match="greeting">
```

The namespace syntax may also use the reserved attribute 'xmlns'. In that case the complete syntax looks like this:

```
xmlns:[prefix]="[URI of namespace]"
```

The prefix can be any characters allowed in an XML tag, except it may not start with xml. Here is an example using the xmlns syntax:

```
<message xmlns: rnif="http://www.rosettanet.org/dtds/edi-xml.dtd'>
```

In the XML document, the following statements occur:

```
<rnif:code>12345678</rnif:code> <code>9876543</code>
```

When the document is processed, it tells the parser:

- For the element <rnif:code> use the element tag from the RosettaNet DTD.
- For the element <code> use the (possibly) undeclared <code> element tag.

Whenever you see a colon preceded by some letter or word (like 's:' or 'xml:'), it signifies that the element or attribute is defined in another location.

Eventually there will be online repositories of:

- XML documents
- XML DTD's
- XSL style sheets
- XML schemas

The use of URI's with these repositories will ensure that tags, elements and attributes are unique across the whole world.

Points to remember:

- Remember the XML declaration.
- Do what the DTD instructs.
- Be careful with capitalization.
- Put attribute values in quotes.
- Close all tags, including empty tags.
- All XML documents must at least be well-formed.