Introduction to XML: *The Syntax of XML, XML Document Structure, Document Type Definitions, Name Space, XML Schemas, Displaying raw XML Documents, Displaying XML Documents with CSS, XSLT Style Sheets and XML Processors.*

XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- XML is extensible: XML allows you to create your own self-descriptive tags or language, that suits your application.
- XML carries the data, does not present it: XML allows you to store the data irrespective of how it will be presented.
- XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Does Not Use Predefined Tags

The XML language has no predefined tags.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

XML Usage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML Document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human - Readable and machine -readable. So ,what exactly is a markup language? Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
<text>
Hello, world!
</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

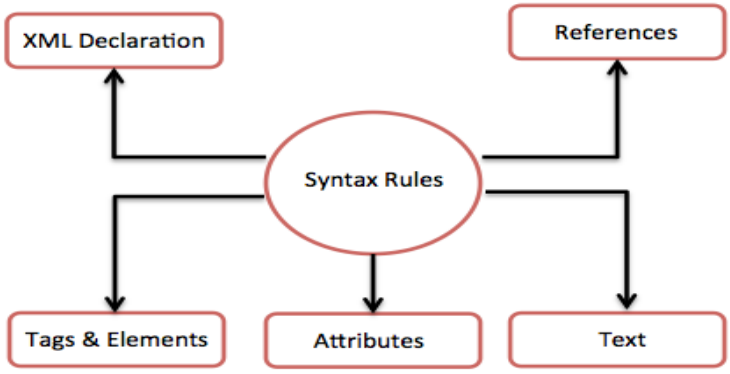
Following is a complete XML document:

```
<? xml version="1.0"?>
<contact -info>
<name> Tanmay Patil </name>
<company> TutorialsPoint </company>
<phone> (011) 123-4567 </phone>
</contact-info>
```

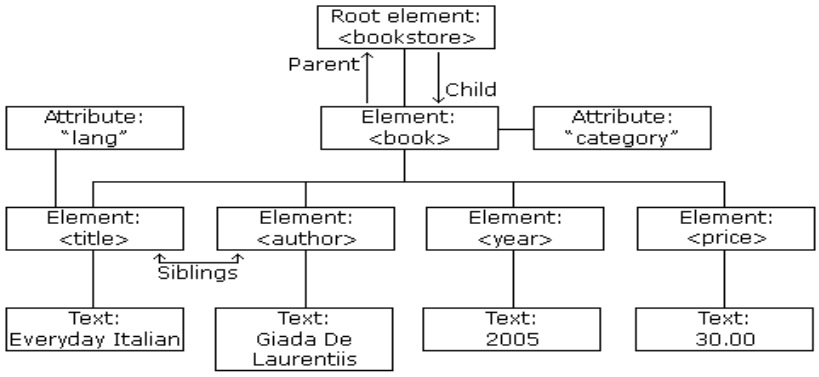
You can notice, there are two kinds of information in the above example:

- Markup, like <contact-info>
- The text, or the character data, Tutorials Point and (040) 123-4567

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



XML documents form a tree structure that starts at "the root" and branches to "the leaves".



XML Tree Structure

XML documents are formed as **element trees**.

An XML tree starts at a **root element** and branches from the root to **child elements**.

All elements can have sub elements (child elements):

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

Self-Describing Syntax

XML uses a much self-describing syntax.

A prolog defines the XML version and the character encoding:
<?xml version="1.0" encoding="UTF-8"?>

The next line is the root element of the document:
<bookstore>

The next line starts a <book> element:
<book category="cooking">

The <book> elements have 4 child elements: <title>, <author>, <year>, <price>.
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>

The next line ends the book element:
</book>

You can assume, from this example, that the XML document contains information about books in a bookstore.

XML Document example

A simple document is given in the following example:
<? xml version="1.0"?>
<contact -info>
<name> Tanmay Patil </name>
<company> TutorialsPoint </company>
<phone> (011) 123-4567 </phone>
</contact-info>

Document Prolog Section

The **document prolog** comes at the top of the document, before the root element. This section contains:

- XML declaration
- Document type declaration

Document Elements Section

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

XML Document Type Declaration

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code:

Start Declaration- Begin the XML declaration with following statement

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

DTD- Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE:

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

XML Namespace:

XML Namespaces provide a method to avoid element name conflicts. A Namespace is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to group. The Namespace is identified by URI(Uniform Resource Identifiers).

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
```

```
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

Namespace Declaration

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

A Namspace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below:

```
<element xmlns:name="URL">
```

Syntax

- The Namespace starts with the keyword **xmlns**.
- The word **name** is the Namespace prefix.
- The **URL** is the Namespace identifier.

Example

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace. Following is a simple example of XML Namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="www.xyz.com/profile">
  <cont:name>Tanmay Patil</cont:name>
  <cont:company>TutorialsPoint</cont:company>
```

```
  <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is **cont**, and the Namespace identifier (URI) as *www.xyz.com/profile*. This means, the element names and attribute names with the **cont** prefix (including the contact element), all belong to the *www.xyz.com/profile* namespace.

Namespaces can also be declared in the XML root element:

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Note: The namespace URI is not used by the parser to look up information.

The purpose of using an URI is to give the namespace a unique name.

However, companies often use the namespace as a pointer to a web page containing namespace information.
Uniform Resource Identifier (URI)

A Uniform Resource Identifier (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the Uniform Resource Locator (URL) which identifies an Internet domain address. Another, not so common type of URI is the Universal Resource Name (URN).
Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

```
xmlns="namespaceURI"
```

This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a piece of furniture:

```
<table xmlns="http://www.w3schools.com/furniture">
```

```
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Namespaces in Real Use

XSLT is a language that can be used to transform XML documents into other formats.

The XML document below, is a document used to transform XML into HTML.

The namespace "http://www.w3.org/1999/XSL/Transform" identifies XSLT elements inside an HTML document:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr>
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XML Schema

An XML Schema describes the structure of an XML document, just like a DTD.
An XML document with correct syntax is called "Well Formed".
An XML document validated against an XML Schema is both "Well Formed" and "Valid".
XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="company" type="xs:string" />
      <xs:element name="phone" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

XML Schemas are More Powerful than DTD

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

Why Use an XML Schema?

With XML Schema, your XML files can carry a description of its own format.
With XML Schema, independent groups of people can agree on a standard for interchanging data.
With XML Schema, you can verify data.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types:

- It is easier to describe document content
- It is easier to define restrictions on data
- It is easier to validate the correctness of data
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT

Definition Types

You can define XML schema elements in following ways:

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

```
<xs:element name="phone_number" type="xs:int" />
```

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
          <xs:element name="company" type="xs:string" />
      <xs:element name="phone" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string" />
               <xs:element name="company" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Now let us use this type in our example as below:

```
<xs:element name="Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="address" type="AddressType" />
               <xs:element name="phone1" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="Address2">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="address" type="AddressType" />
               <xs:element name="phone2" type="xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:

```
<xs:attribute name="x" type="y"/>
```

Display RAW XML Documents

Raw XML files can be viewed in all major browsers. Don't expect XML files to be displayed as HTML pages.

An XML enabled browser or any other system that can deal with XML documents cannot possibly know how to format the tags defined in the doc. Without a style sheet that defines presentation styles for the doc tags the XML doc cannot be displayed in a formatted manner.

Some browsers like FX2 have default style sheets that are used when style sheets are not defined.

Eg of planes.xml document.

Viewing XML Files

```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
```

```
    <body>Don't forget me this weekend!</body>
  </note>
```

Look at the XML file above in your browser: note.xml


```
<?xml version="1.0" encoding="UTF-8" ?>
- <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
```

Most browsers will display an XML document with color-coded elements.
Often a plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure.
To view raw XML source, try to select "View Page Source" or "View Source" from the browser menu.

Why Does XML Display Like This?

XML documents do not carry information about how to display the data.
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, the browsers can just display the XML document as it is.

Displaying XML Documents with CSS

Style sheet information can be provided to the browser for an xml document in two ways.
• First, a CSS file that has style information for the elements in the XML document can be developed.
• Second the XSLT style sheet technology can be used..
Using CSS is effective, XSLT provides far more power over the appearance of the documents display.
A CSS style sheet for an XML document is just a list of its tags and associated styles
The connection of an XML document and its style sheet is made through an xmlstylesheet processing instruction
Display– used to specify whether an element is to be displayed inline or in a separate block.
<?xml-stylesheet type = "text/css“ href = “planes.css"?>
For example: planes.css

```
<!-- planes.css - a style sheet for the planes.xml document -->
ad { display: block; margin-top: 15px; color: blue;}
year, make, model { color: red; font-size: 16pt;}
color {display: block; margin-left: 20px; font-size: 12pt;}
description {display: block; margin-left: 20px; font-size: 12pt;}
seller { display: block; margin-left: 15px; font-size: 14pt;}
location {display: block; margin-left: 40px; }
city {font-size: 12pt;}
state {font-size: 12pt;}
```

```
<?xml version = "1.0" encoding = "utf-8"?>
<?xml-stylesheet type="text/css" href="planes.css"?>
<!-- planes.xml - A document that lists ads for used airplanes -->
<planes_for_sale>
<ad>
```

<year> 1977 </year>
<make> Cessana </make>
<model> Skyhawk </model>
<color> Light blue and white </color>
<description> New interior
</description>
<seller phone = "555-222-3333">
Skyway Aircraft </seller>
<location>
<city> Rapid City, </city>
<state> South Dakota </state>
</location>
</ad>
</planes_for_sale>

With planes.css the display of planes.xml as following:

1977 Cessana Skyhawk

Light blue and white

New interior

Skyway Aircraft

Rapid City, South Dakota

XSLT Style Sheet

With XSLT you can transform an XML document into HTML. XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.

XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

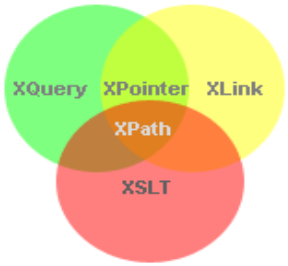
XSLT uses XPath to find information in an XML document.

What is XPath?

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.

- XPath is a syntax for defining parts of an XML document



- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery
- XPath is a W3C recommendation

XPath Path Expressions

- XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.
- XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

- Following is the list of useful paths and expression to select any node/ list of nodes from an XML document.

S.No.	Expression & Description
1	**node-name** Select all nodes with the given name "nodename"
2	/ Selection starts from the root node
3	// Selection starts from the current node that match the selection
4	. Selects the current node
5	.. Selects the parent of the current node
6	@ Selects attributes
7	**student** Example – Selects all nodes with the name "student"
8	**class/student** Example – Selects all student elements that are children of class
9	**//student** Selects all student elements no matter where they are in the document

XPath is Used in XSLT

- XPath is a major element in the XSLT standard.
- With XPath knowledge you will be able to take great advantage of XSL.

XPath Example

We will use the following XML document:

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

<bookstore>

<book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
```

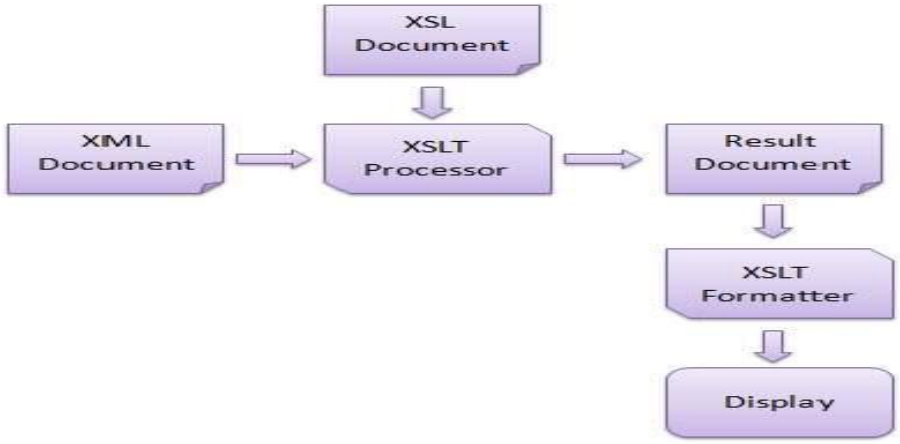
```
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

In the table below we have listed some XPath expressions and the result of the expressions:

XPath Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

How XSLT Works

An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



Advantages

Here are the advantages of using XSLT −

- Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
- Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

Displaying XML with XSLT

XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
XSLT uses XPath to find information in an XML document.

XSLT Example

We will use the following XML document:

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

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
<calories>900</calories>
</food>
```

```
<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
<calories>950</calories>
</food>
</breakfast_menu>
```

Use XSLT to transform XML into HTML, before it is displayed in a browser:

Example XSLT Stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;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"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

Belgian Waffles - $5.95

Two of our famous Belgian Waffles with plenty of real maple syrup *(650 calories per serving)*

Strawberry Belgian Waffles - $7.95

Light Belgian waffles covered with strawberries and whipped cream *(900 calories per serving)*

Berry-Berry Belgian Waffles - $8.95

Light Belgian waffles covered with an assortment of fresh berries and whipped cream *(900 calories per serving)*

French Toast - $4.50

Thick slices made from our homemade sourdough bread *(600 calories per serving)*

Homestyle Breakfast - $6.95

Two eggs, bacon or sausage, toast, and our ever-popular hash browns *(950 calories per serving)*

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Output:

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler

Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart

XML Processor

When a software program reads an XML document and takes actions accordingly, this is called *processing* the XML. Any program that can read and process XML documents is known as an *XML processor*. An XML processor reads the XML file and turns it into in-memory structures that the rest of the program can access.
The most fundamental XML processor reads an XML document and converts it into an internal representation for other programs or subroutines to use. This is called a *parser*, and it is an important component of every XML processing program.

Types

XML processors are classified as **validating** or **non-validating** types, depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue with normal processing.
A few validating parsers are:xml4c (IBM, in C++), xml4j (IBM, in Java), MSXML (Microsoft, in Java), TclXML (TCL), xmlproc (Python), XML::Parser (Perl), Java Project X (Sun, in Java).
A few non-validating parsers are: OpenXML (Java), Lark (Java), xp (Java), AElfred (Java), expat (C), XParse (JavaScript), xmllib (Python).
XML Parser:
All major browsers have a built-in XML parser to access and manipulate XML. The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.
However, before an XML document can be accessed, it must be loaded into an XML DOM object.
All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

Parsing a Text String

This example parses a text string into an XML DOM object, and extracts the info from it with JavaScript:
Example

```
<html>
<body>
<p id="demo"></p>
<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

Example Explained

A text string is defined:

```
text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";
```

An XML DOM parser is created:

```
parser = new DOMParser();
```

The parser creates a new XML DOM object using the text string:

```
xmlDoc = parser.parseFromString(text,"text/xml");
```

Output: Everyday Italian (in browser)