UNIT II

What is xml

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

What is mark-up language

A **mark up language** is a modern system for highlight or underline a document.
Students often underline or highlight a passage to revise easily, same in the sense of modern mark up language highlighting or underlining is replaced by tags.

Why xml

Platform Independent and Language Independent: The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.
The main thing which makes XML truly powerful is its international acceptance. Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

Features and Advantages of XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.
With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.
XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.
This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.
Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.
XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources.
With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.
Here are some examples:

- **XHTML**
- **WSDL** for describing available web services
- **WAP** and **WML** as markup languages for handheld devices
- **RSS** languages for news feeds
- **RDF** and **OWL** for describing resources and ontology
- **SMIL** for describing multimedia for the web

XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".
Syntax:

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

Example of XML Document

XML documents uses a self-describing and simple syntax:

```
<?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>
```

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

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

And finally the last line defines the end of the root element.

```
</note>
```

XML: Books.xml

```
<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">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

XML: Emails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
<email>
  <to>Vimal</to>
  <from>Sonoo</from>
  <heading>Hello</heading>
  <body>Hello brother, how are you!</body>
</email>
<email>
  <to>Peter</to>
  <from>Jack</from>
  <heading>Birth day wish</heading>
  <body>Happy birth day Tom!</body>
</email>
<email>
  <to>James</to>
  <from>Jaclin</from>
  <heading>Morning walk</heading>
  <body>Please start morning walk to stay fit!</body>
```

```
</email>
<email>
  <to>Kartik</to>
  <from>Kumar</from>
  <heading>Health Tips</heading>
  <body>Smoking is injurious to health!</body>
</email>
</emails>
```

No.	Technology	Meaning	Description
1)	XHTML	Extensible html	It is a clearer and stricter version of XML. It belongs to the family of XML markup languages. It was developed to make html more extensible and increase inter-operability with other data.
2)	XML DOM	XML document object model	It is a standard document model that is used to access and manipulate XML. It defines the XML file in tree structure.
3)	XSL it contain three parts: i) XSLT (xsl transform) ii) XSL iii)XPath	Extensible style sheet language	i) It transforms XML into other formats, like html. ii) It is used for formatting XML to screen, paper etc. iii) It is a language to navigate XML documents.
4)	XQuery	XML query language	It is a XML based language which is used to query XML based data.
5)	DTD	Document type definition	It is an standard which is used to define the legal elements in an XML document.
6)	XSD	XML schema definition	It is an XML based alternative to dtd. It is used to describe the structure of an XML document.
7)	XLink	XML linking	xlink stands for XML linking language. This is a

		language	language for creating hyperlinks (external and internal links) in XML documents.
8)	XPointer	XML pointer language	It is a system for addressing components of XML based internet media. It allows the xlink hyperlinks to point to more specific parts in the XML document.
9)	SOAP	Simple object access protocol	It is an acronym stands simple object access protocol. It is XML based protocol to let applications exchange information over http. in simple words you can say that it is protocol used for accessing web services.
10)	WSDL	web services description languages	It is an XML based language to describe web services. It also describes the functionality offered by a web service.
11)	RDF	Resource description framework	RDF is an XML based language to describe web resources. It is a standard model for data interchange on the web. It is used to describe the title, author, content and copyright information of a web page.
12)	SVG	Scalable vector graphics	It is an XML based vector image format for two-dimensional images. It defines graphics in XML format. It also supports animation.
13)	RSS	Really simple syndication	RSS is a XML-based format to handle web content syndication. It is used for fast browsing for news and updates. It is generally used for news like sites.

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.
XML attributes enhance the properties of the elements.

```
<book publisher="Tata McGraw Hill"></book>
```

Here, book is the element and publisher is the attribute.

Metadata should be stored as attribute and data should be stored as element.

```
<book>
```

```
<book category="computer">
<author> A & B </author>
</book>
```

Difference between attribute and sub-element

1st way:

```
<book publisher="Tata McGraw Hill"> </book>
```

2nd way:

```
<book>
<publisher> Tata McGraw Hill </publisher>
</book>
```

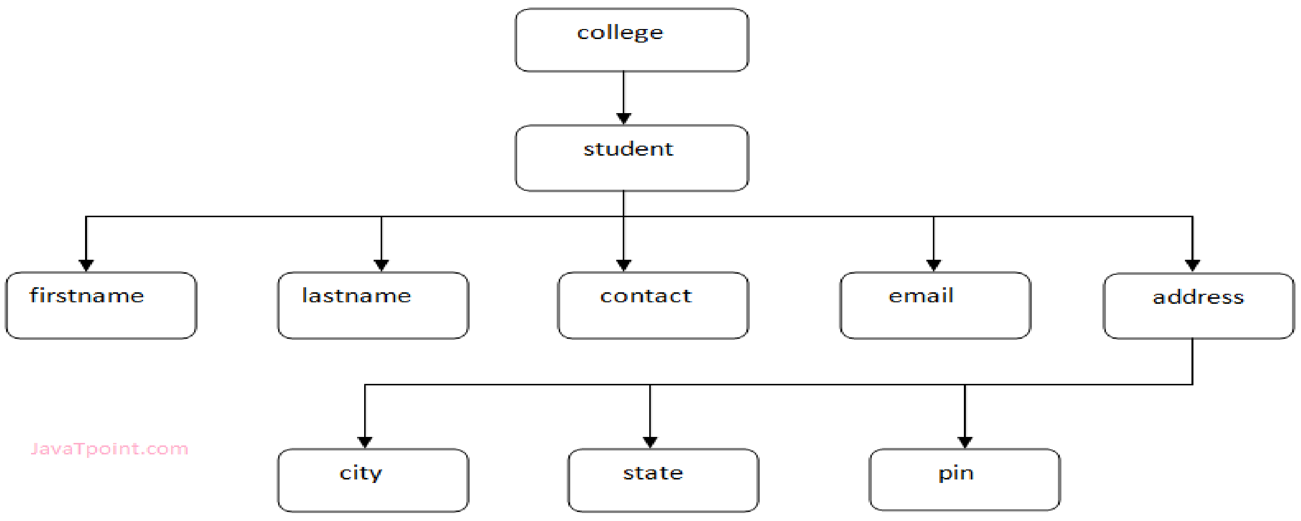
syntax ***An XML comment should be written as: <!-- Write your comment-->***

1. Don't use a comment before an XML declaration.
2. You can use a comment anywhere in XML document except within attribute value.
3. Don't nest a comment inside the other comment.

XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

```
<?xml version="1.0"?>
<college>
  <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
        <city>Ghaziabad</city>
        <state>Uttar Pradesh</state>
        <pin>201007</pin>
    </address>
  </student>
</college>
```



XML Validation

A well formed XML document can be validated against DTD or Schema.
A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)
It should be behave according to predefined DTD or XML schema

Rules for well formed XML

- It must begin with the XML declaration.
- It must have one unique root element.
- All start tags of XML documents must match end tags.
- XML tags are case sensitive.
- All elements must be closed.
- All elements must be properly nested.
- All attributes values must be quoted.
- XML entities must be used for special characters.

Document Type Definition (DTD)

What is a DTD?
A DTD is a Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document.
Why Use a DTD?
With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.
In a DTD, elements are declared with an ELEMENT declaration.
Declaring Elements
In a DTD, XML elements are declared with the following syntax:
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
Empty Elements
Empty elements are declared with the category keyword EMPTY:
<!ELEMENT element-name EMPTY>
Example:
<!ELEMENT br EMPTY>
XML example:
<br />
Elements with Parsed Character Data
Elements with only parsed character data are declared with #PCDATA inside parentheses:
<!ELEMENT element-name (#PCDATA)>
Example:
<!ELEMENT from (#PCDATA)>
Elements with any Contents
Elements declared with the category keyword ANY, can contain any combination of parsable data:
<!ELEMENT element-name ANY>
Example:
<!ELEMENT note ANY>
Elements with Children (sequences)
Elements with one or more children are declared with the name of the children elements inside parentheses:
<!ELEMENT element-name (child1)>
or
<!ELEMENT element-name (child1,child2,...)>
Example:
<!ELEMENT note (to,from,heading,body)>
Declaring Minimum One Occurrence of an Element
<!ELEMENT element-name (child-name+)>
Example:
<!ELEMENT note (message+)>
Declaring Zero or More Occurrences of an Element
<!ELEMENT element-name (child-name*)>
Example:
<!ELEMENT note (message*)>
Declaring Zero or One Occurrences of an Element
<!ELEMENT element-name (child-name?)>
Example:
<!ELEMENT note (message?)>
Declaring either/or Content

```
<!ELEMENT note (to,from,header,(message|body))>
```

Declaring Mixed Content

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

DTD - Attributes
In a DTD, attributes are declared with an ATTLIST declaration.

Declaring Attributes
An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type="check" />
```

The **attribute-type** can be one of the following:

Type	**Description**
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

The **attribute-value** can be one of the following:

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

A Default Attribute Value
DTD:
```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:
```
<square width="100" />
```
In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED
Syntax
```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```
Example
DTD:
```
<!ATTLIST person number CDATA #REQUIRED>
```

Valid XML:
```
<person number="5677" />
```

Invalid XML:
```
<person />
```
Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#IMPLIED
Syntax
```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```
Example
DTD:
```
<!ATTLIST contact fax CDATA #IMPLIED>
```

Valid XML:
```
<contact fax="555-667788" />
```

Valid XML:

```
<contact />
```

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#FIXED

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

Example

DTD:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

Valid XML:

```
<sender company="Microsoft" />
```

Invalid XML:

```
<sender company="W3Schools" />
```

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values

Syntax

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

Example

DTD:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check" />
```

or

```
<payment type="cash" />
```

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal DTD

```
<?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** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

```
<?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** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

```
<?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 here is the file "note.dtd", which contains the DTD:

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

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.
An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:<!ENTITY entity-name "entity-value">

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
  <!ELEMENT author (#PCDATA)>
  <!ENTITY sj "Sonoo Jaiswal">
]>
<author>&sj;</author>
```

XML Namespaces

XML **Namespace** is used *to avoid element name conflict* in XML document.

XML Namespace Declaration

An XML namespace is declared using the reserved XML attribute. This attribute name must be started with "xmlns".
Let's see the XML namespace syntax: **<element** xmlns:name = "URL">
Here, namespace starts with keyword **"xmlns"**. The word **name** is a namespace prefix.
The **URL** is a namespace identifier.

Let's see the example of XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<cont:contact xmlns:cont="http://sssit.org/contact-us">
<cont:name>Vimal Jaiswal</cont:name>
<cont:company>SSSIT.org</cont:company>
<cont:phone>(0120) 425-6464</cont:phone>
</cont:contact>
```

Namespace Prefix: cont
Namespace Identifier: http://sssit.org/contact-us
It specifies that the element name and attribute names with cont prefix belongs to http://sssit.org/contact-us name space.
In XML, elements name are defined by the developer so there is a chance to conflict in name of the elements. To avoid these types of confliction we use XML Namespaces. We can say that XML Namespaces provide a method to avoid element name conflict.
Generally these conflict occurs when we try to mix XML documents from different XML application.

Table1:

```
<table>
  <tr>
    <td>Aries</td>
    <td>Bingo</td>
  </tr>
</table>
```

Table2:

```
<table>
<tr>
  <name>Computer table</name>
  <width>80</width>
  </tr>
</table>
```

If you add these both XML fragments together, there would be a name conflict because both have <table< element. Although they have different name and meaning.

XML Schema

What is XML schema

XML schema is a language which is used for expressing constraint about XML documents. There are so many schema languages which are used now a days for example Relax- NG and XSD (XML schema definition).
An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure.
Why Learn XML Schema?
In the XML world, hundreds of standardized XML formats are in daily use.
Many of these XML standards are defined by XML Schemas.
XML Schema is an XML-based (and more powerful) alternative to DTD.
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 Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.
With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types

- Reference multiple schemas in the same document

The <schema> Element
The <schema> element is the root element of every XML Schema:

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

<xs:schema>
...
...
</xs:schema>
```

XSD Simple Elements

XML Schemas define the elements of your XML files.
A simple element is an XML element that contains only text. It cannot contain any other elements or attributes

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.
XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example
Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

And here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Default and Fixed Values for Simple Elements
Simple elements may have a default value OR a fixed value specified.
A default value is automatically assigned to the element when no other value is specified.
In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.
In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

XSD Attributes

All attributes are declared as simple types.

What is an Attribute?
Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.
How to Define an Attribute?
The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

where xxx is the name of the attribute and yyy specifies the data type of the attribute.
XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example
Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!
Examples of Complex Elements
A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description>
It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

How to Define a Complex Element
Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

We can define a complex element in an XML Schema two different ways:
1. The "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators in the XSD Indicators chapter.
2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Complex Empty Elements
An empty XML element:

```
<product prodid="1345" />
```

String Data Type
The string data type can contain characters, line feeds, carriage returns, and tab characters.
The following is an example of a string declaration in a schema:

```
<xs:element name="customer" type="xs:string"/>
```

Date Data Type
The date data type is used to specify a date.
The date is specified in the following form "YYYY-MM-DD" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day

Note: All components are required!
The following is an example of a date declaration in a schema:

```
<xs:element name="start" type="xs:date"/>
```

Decimal Data Type
The decimal data type is used to specify a numeric value.
The following is an example of a decimal declaration in a schema:

```
<xs:element name="prize" type="xs:decimal"/>
```

Boolean Data Type
The boolean data type is used to specify a true or false value.
The following is an example of a boolean declaration in a schema:

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

XML Schema Example

employee.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.abc.com"
xmlns="http://www.abc.com"
elementFormDefault="qualified">
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

employee.xml

```
<?xml version="1.0"?>
<employee
xmlns="http://www.abc.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.abc.com employee.xsd">
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Description of XML Schema

<xs:element name="employee"> : It defines the element name employee.
<xs:complexType> : It defines that the element 'employee' is complex type.
<xs:sequence> : It defines that the complex type is a sequence of elements.
<xs:element name="firstname" type="xs:string"/> : It defines that the element 'firstname' is of string/text type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

simpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty.

complexType

The complexType allows you to hold multiple attributes and elements. It can contain additional sub elements and can be left empty

DTD vs XSD

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
]]>
</employee>
```

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

XML CSS

Purpose of CSS in XML

CSS (Cascading Style Sheets) can be used to add style and display information to an XML document. It can format the whole XML document.

How to link XML file with CSS

To link XML files with CSS, you should use the following syntax:

```
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
```

XML CSS Example

cssemployee.css

```
employee
{
background-color: pink;
}
firstname,lastname,email
{
font-size:25px;
display:block;
color: blue;
margin-left: 50px;
}
```

employee.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Example 1.

In this example, the XML file is created that contains the information about five books and displaying the XML file using CSS.

Books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Rule.css"?>
<books>
        <heading>Welcome To Library </heading>
        <book>
                <title>Title -: Web Programming</title>
                <author>Author -: Chrisbates</author>
                <publisher>Publisher -: Wiley</publisher>
                <edition>Edition -: 3</edition>
                <price> Price -: 300</price>
        </book>
        <book>
                <title>Title -: Internet world-wide-web</title>
```

```
		<author>Author -: Ditel</author>
		<publisher>Publisher -: Pearson</publisher>
		<edition>Edition -: 3</edition>
		<price>Price -: 400</price>
	</book>
	<book>
		<title>Title -: Computer Networks</title>
		<author>Author -: Foruouzan</author>
		<publisher>Publisher -: Mc Graw Hill</publisher>
		<edition>Edition -: 5</edition>
		<price>Price -: 700</price>
	</book>
	<book>
		<title>Title -: DBMS Concepts</title>
		<author>Author -: Navath</author>
		<publisher>Publisher -: Oxford</publisher>
		<edition>Edition -: 5</edition>
		<price>Price -: 600</price>
	</book>
	<book>
		<title>Title -: Linux Programming</title>
		<author>Author -: Subhitab Das</author>
		<publisher>Publisher -: Oxford</publisher>
		<edition>Edition -: 8</edition>
		<price>Price -: 300</price>
	</book>
</books>
```

Rule.css

```
books {
	color: white;
	background-color : gray;
	width: 100%;
}
heading {
	color: green;
	font-size : 40px;
	background-color : powderblue;
}
heading, title, author, publisher, edition, price {
	display : block;
}
title {
	font-size : 25px;
	font-weight : bold;
}
```

in this example, the XML file is created that contains the information about various sections in Geeks for Geeks and the topics they contains and after that displaying the XML file using CSS .

Section.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Geeks.css"?>
<Geeks_for_Geeks>
        <title>Hello Everyone! Welcome to Library</title>
        <geeks_section>
                <name>Algo</name>
                <topic1>Greedy Algo</topic1>
                <topic2>Randomised Algo</topic2>
                <topic3>Searching Algo</topic3>
                <topic4>Sorting Algo</topic4>
        </geeks_section>
        <geeks_section>
                <name>Data Structures</name>
                <topic1>Array</topic1>
                <topic2>Stack</topic2>
                <topic3>Queue</topic3>
                <topic4>Linked List</topic4>
        </geeks_section>
        <geeks_section>
                <name>Web Technology</name>
                <topic1>HTML</topic1>
                <topic2>CSS</topic2>
                <topic3>Java Script</topic3>
                <topic4>Php</topic4>
        </geeks_section>
        <geeks_section>
                <name>Languages</name>
                <topic1>C/C++</topic1>
                <topic2>Java</topic2>
                <topic3>Python</topic3>
                <topic4>Ruby</topic4>
        </geeks_section>
        <geeks_section>
                <name>DBMS</name>
                <topic1>Basics</topic1>
                <topic2>ER Diagram</topic2>
                <topic3>Normalization</topic3>
                <topic4>Transaction Concepts</topic4>
        </geeks_section>
</Geeks_for_Geeks>
```

Geeks.css

```
Geeks_for_Geeks
                        {
                        font-size:80%;
                        margin:0.5em;
                        font-family: Verdana;
                        display:block;
                        }
geeks_section {
                        display:block;
                        border: 1px solid silver;
                        margin:0.5em;
                        padding:0.5em;
                        background-color:whitesmoke;
                        }
title {
        display:block;
        font-weight:bolder;
        text-align:center;
        font-size:30px;
        background-color: green;
        color: white;

        }
name, topic1, topic2, topic3, topic4
{
display: block;
text-align: center;
}
name {
        color: green;
        text-decoration: underline ;
        font-weight: bolder;
        font-size:20px;
        }
topic1 {
        color: green
        }
topic2 {
        color: brown
        }
topic3 {
        color: blue
        }
```

```
topic4 {
        color: orange }
```

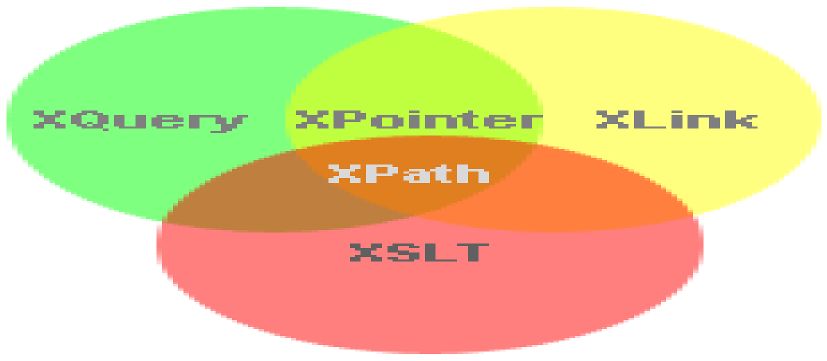
WHAT IS XPATH?

XPath is an official recommendation of the World Wide Web Consortium (W3C). It defines a language to find information in an XML file. It is used to traverse elements and attributes of an XML document. XPath provides various types of expressions which can be used to enquire relevant information from the XML document.

- **Structure Definitions** − XPath defines the parts of an XML document like element, attribute, text, namespace, processing-instruction, comment, and document nodes
- **Path Expressions** − XPath provides powerful path expressions select nodes or list of nodes in XML documents.
- **Standard Functions** − XPath provides a rich library of standard functions for manipulation of string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Major part of XSLT** − XPath is one of the major elements in XSLT standard and is must have knowledge in order to work with XSLT documents.
- **W3C recommendation** − XPath is an official recommendation of World Wide Web Consortium (W3C).

One should keep the following points in mind, while working with XPath −

- XPath is core component of XSLT standard.
- XSLT cannot work without XPath.
- XPath is basis of XQuery and XPointer.



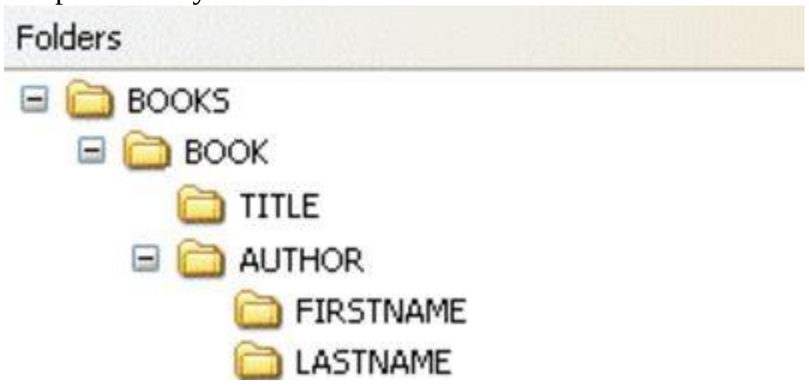
- Path stands for XML Path Language
- XPath uses "path like" syntax to identify and navigate nodes in an XML document
- XPath contains over 200 built-in functions

- XPath is a major element in the XSLT standard
- XPath is a W3C recommendation

Important features of XPath

- **XPath defines structure:** XPath is used to define the parts of an XML document i.e. element, attributes, text, namespace, processing-instruction, comment, and document nodes.
- **XPath provides path expression:** XPath provides powerful path expressions, select nodes, or list of nodes in XML documents.
- **XPath is a core component of XSLT:** XPath is a major element in XSLT standard and must be followed to work with XSLT documents.
- **XPath is a standard function:** XPath provides a rich library of standard functions to manipulate string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Path is 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 path expressions you use with traditional computer file systems:



XPath Standard Functions
XPath includes over 200 built-in functions.
There are functions for string values, numeric values, booleans, date and time comparison, node manipulation, sequence manipulation, and much more.
Today XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.
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 your XSLT knowledge.
XPath is a W3C Recommendation
XPath 1.0 became a W3C Recommendation on November 16, 1999.
XPath 2.0 became a W3C Recommendation on January 23, 2007.
XPath 3.0 became a W3C Recommendation on April 8, 2014.
XPath Terminology
Nodes

In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes.
XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.
Look at the following XML document:

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

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Example of nodes in the XML document above:
<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)
Atomic values
Atomic values are nodes with no children or parent.
Example of atomic values:
J K. Rowling

"en"
Items
Items are atomic values or nodes.
Relationship of Nodes
Parent
Each element and attribute has one parent.
In the following example; the book element is the parent of the title, author, year, and price:
Children
Element nodes may have zero, one or more children.
In the following example; the title, author, year, and price elements are all children of the book element
Siblings
Nodes that have the same parent.
In the following example; the title, author, year, and price elements are all siblings:
Ancestors
A node's parent, parent's parent, etc.
In the following example; the ancestors of the title element are the book element and the bookstore element:
Descendants
A node's children, children's children, etc.

In the following example; descendants of the bookstore element are the book, title, author, year, and price elements:
XPath uses path expressions to select nodes or node-sets in an XML document. The node is selected by following a path or steps.
The XML Example Document
We will use the following XML document in the examples below.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

Selecting Nodes
XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	**Description**
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

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

Path Expression	**Result**
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document

bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Predicates
Predicates are used to find a specific node or a node that contains a specific value.
Predicates are always embedded in square brackets.
In the table below we have listed some path expressions with predicates and the result of the expressions:

Path Expression	**Result**
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: *In JavaScript:* *xml*.setProperty("SelectionLanguage","XPath");
/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

Selecting Unknown Nodes
XPath wildcards can be used to select unknown XML nodes.

Wildcard	**Description**

*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

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

Path Expression	**Result**
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

Selecting Several Paths
By using the | operator in an XPath expression you can select several paths.
In the table below we have listed some path expressions and the result of the expressions:

Path Expression	**Result**
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

The XML Example Document
"books.xml":

```
<?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>
```

Loading the XML Document

Using an XMLHttpRequest object to load XML documents is supported in all modern browsers.

```
var xmlhttp = new XMLHttpRequest();
```

Code for older browsers (IE5 and IE6) can be found in the AJAX tutorial.

Selecting Nodes

Unfortunately, there are different ways of dealing with XPath in different browsers.
Chrome, Firefox, Edge, Opera, and Safari use the evaluate() method to select nodes:

```
xmlDoc.evaluate(xpath, xmlDoc, null, XPathResult.ANY_TYPE,null);
```

Internet Explorer uses the selectNodes() method to select node:

```
xmlDoc.selectNodes(xpath);
```

In our examples we have included code that should work with most major browsers.

Select all the titles

The following example selects all the title nodes:

Example

```
/bookstore/book/title
```

Select the title of the first book

The following example selects the title of the first book node under the bookstore element:

Example

```
/bookstore/book[1]/title
```

Select all the prices

The following example selects the text from all the price nodes:

Example

```
/bookstore/book/price[text()]
```

Select price nodes with price>35

The following example selects all the price nodes with a price higher than 35:

Example

```
/bookstore/book[price>35]/price
```

Select title nodes with price>35

The following example selects all the title nodes with a price higher than 35:

Example

```
/bookstore/book[price>35]/title
```

XSL

Before learning XSLT, we should first understand XSL which stands for E**X**tensible **S**tylesheet **L**anguage. It is similar to XML as CSS is to HTML.

Need for XSL

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

Following are the main parts of XSL −

- **XSLT** − used to transform XML document into various other types of document.
- **XPath** − used to navigate XML document.
- **XSL-FO** − used to format XML document.

What is XSLT

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

CSS = Style Sheets for HTML

HTML uses predefined tags. The meaning of, and how to display each tag is well understood.

CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

XML does not use predefined tags, and therefore the meaning of each tag is not well understood.

A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!

So, XSL describes how the XML elements should be displayed.

XSLT = XSL Transformations

XSLT is the most important part of XSL.

XSLT 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.

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.

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

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

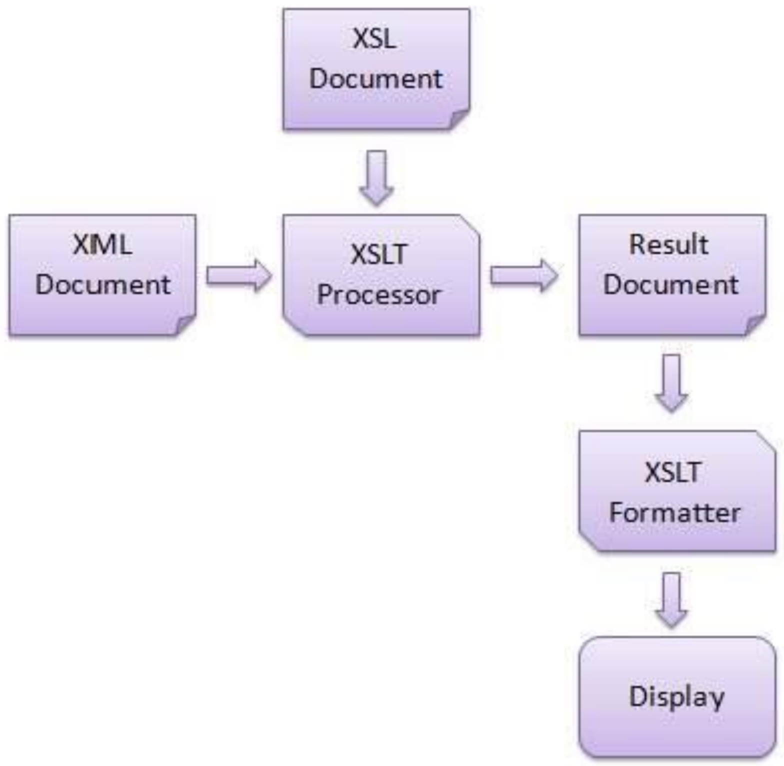
XSL - More Than a Style Sheet Language

XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

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.

Declaration

Following is the syntax declaration of **<xsl:template>** element.

```
<xsl:template
   name = Qname
   match = Pattern
   priority = number
   mode = QName >
</xsl:template>
```

Attributes

Sr.No	Name & Description
1	**name** Name of the element on which template is to be applied.
2	**match** Pattern which signifies the element(s) on which template is to be applied.
3	**priority** Priority number of a template. Matching template with low priority is not considered in from in front of high priority template.
4	**mode** Allows element to be processed multiple times to produce a different result each time.

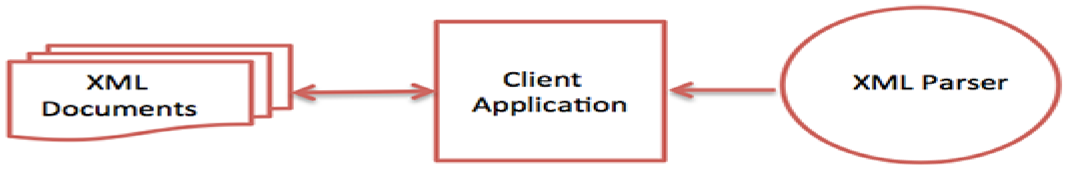
- Elements

Number of occurrences	Unlimited
Parent elements	xsl:stylesheet, xsl:transform
Child elements	xsl:apply-imports,xsl:apply-templates,xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:number, xsl:param, xsl:processing-instruction, xsl:text, xsl:value-of, xsl:variable, output elements

XML Parsers

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.
Let's understand the working of XML parser by the figure given below:



Types of XML Parsers

These are the two main types of XML Parsers:

Types of XML Parsers
These are the two main types of XML Parsers:

1. DOM
2. SAX

DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.

Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.
DOM Parser has a tree based structure.

Advantages

1) It supports both read and write operations and the API is very simple to use.
2) It is preferred when random access to widely separated parts of a document is required.

Disadvantages

1) It is memory inefficient. (consumes more memory because the whole XML document needs to loaded into memory).
2) It is comparatively slower than other parsers.

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.
Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.
It is an event based parser, it works like an event handler in Java.

Advantages

1) It is simple and memory efficient.
2) It is very fast and works for huge documents.

Disadvantages

1) It is event-based so its API is less intuitive.
2) Clients never know the full information because the data is broken into pieces.