UNIT – III
XML

Introduction:

XML stands for "Extensible Markup Language". XML is a text-based markup language. A markup language is a set of instructions often called as tags which can be added to text files. When the file is processed by a suitable application the tags are used to control the structure and presentation of data contained in the file. Most commonly tags are used by application when presenting data.

XML is more than HTML because, HTML only deals with presentation and carries no information about the data, whereas XML deals with the representation of data and carries information about the data. In XML, we create our own tags and also syntax for those tags that can let the document structure follow the data structure. As we are defining our own tags and there are no predefined tags, XML is defined to be "self-descriptive". Using scripting languages like javascript we can reach to various elements of an xml page and make use of data.

When compared with HTML, XML provides a way of structuring the data and store data with focus on "What data is", where HTML was designed to display data with focus on "How data looks".

XML is nothing special, it is just a plain text. Software that can handle plain text can also handle XML. XML is a software/hardware independent tool for carrying information.

Advantages of XML:

- **Readability:**

  XML document is plain text and human readable.

- **Hierarchal:**

  XML document has a tree like structure which is enough to express complex data as simple as possible.

- **Language Independent:**

  XML documents are language neutral. For example, a java program can generate an XML which can be parsed by a program written in C++ or perl.

- **Platform Independent:**

  XML files are operating system independent.

Uses of XML:

- XML is used to describe the contents of a document.
- XML is used in messaging, where applications exchange data between them.
- The data can be extracted from the database, can be preserved with original information and can be used in more than one application in different ways.

Important points about XML:

- XML is case sensitive.
- All the XML files are stored with .xml extension.
- Every XML document begin with <?xml version="1.0"?>

Example: XML to store customer information of a supermarket.

```
<?xml version="1.0"?>
<document>
    <customer>
        <name>
            <firstname>aaa</firstname>
            <lastname>bbb</lastname>
        </name>
        <date>01 jan 2017</date>
        <orders>
            <item>
                <product>Chocolates</product>
                <number>777</number>
                <price>250</price>
            </item>
            <item>
                <product>Sweets</product>
                <number>888</number>
                <price>450</price>
            </item>
```

```
        </orders>
    </customer>
</document>
```

Valid & Well-formed XML document:

An XML document is said to be valid only if it is associated with a **"Document Type Definition"** or **"XML schema"**.

An XML document is said to be well-formed only if it satisfies the below constraints:

- All tags must have corresponding end tag.

Well-Formed	Not Well-Formed
<personnel> <employee> <name>aaa</name> <id>504</id> <age>39</age> </employee> </personnel>	<personnel> <employee> <name>aaa</name> <id>504</id> <age>39 </employee> </personnel>

- Must have no overlapping of tags.

Well-Formed	Not Well-Formed
<personnel> <employee> <name>aaa</name> <id>504</id> <age>39</age> </employee> </personnel>	<personnel> <employee> <name>aaa</name> <id>504</id> <age>39 </personnel> </employee>

Most of the XML parsers require XML documents to be well-formed, but not necessarily valid.

Document Type Definition(DTD):

- Document Type Definition specifies syntax for XML document i.e., it specifies rules that apply to the data.
- The XML document contains data, whereas DTD contains rules that apply to the data.
- A DTD defines the document structure with a list of legal elements and attributes.

Syntax of DTD:

<!DOCTYPE ROOTELEMENT[

<!ELEMENT ELEMENT_NAME1(subelement1,subelement2….)>

<!ELEMENT ELEMENT_NAME2(subelement1,subelement2….)>

.

.

<!ELEMENT ELEMENT_NAMEn(subelement1,subelement2….)>

]>

There are two types of DTD:

1. Internal DTD.
2. External DTD.

Internal DTD is the one in which we specify the DTD within the XML document. The scope of Internal DTD is limited to one document only.

Example:

<?xml version=”1.0”?>

<!DOCTYPE document[

<!ELEMENT document (customer)*>

<!ELEMENT customer (name,date,orders)>

<!ELEMENT name (firstname,lastname)>

<!ELEMENT firstname(#PCDATA)>

<!ELEMENT lastname (#PCDATA)>

<!ELEMENT date (#PCDATA)>

<!ELEMENT orders (item)*>

<!ELEMENT item (product,number,price)>

<!ELEMENT product (#PCDATA)>

<!ELEMENT number (#PCDATA)>

<!ELEMENT price (#PCDATA)>

]>

```
<document>
    <customer>
        <name>
            <firstname>aaa</firstname>
            <lastname>bbb</lastname>
        </name>
        <date>01 jan 2017</date>
        <orders>
            <item>
                <product>Chocolates</product>
                <number>777</number>
                <price>250</price>
            </item>
            <item>
                <product>Sweets</product>
                <number>888</number>
                <price>450</price>
            </item>
        </orders>
    </customer>
</document>
```

External DTD is the one where DTD is specified as a separate file and is saved with .dtd extension and contains only set of rules. To import the DTD file into XML we include the below statement in XML file:

<!DOCTYPE ROOT_ELEMENT SYSTEM “filename.dtd”>

Example:

Extern.dtd:

```
<!ELEMENT document (customer)*>
<!ELEMENT customer (name,date,orders)>
<!ELEMENT name (firstname,lastname)>
```

```
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT orders (item)*>
<!ELEMENT item (product,number,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Cust.xml:

```
<?xml version=”1.0”?>
<!DOCTYPE document SYSTEM “extern.dtd”>
<document>
    <customer>
        <name>
            <firstname>aaa</firstname>
            <lastname>bbb</lastname>
        </name>
        <date>01 jan 2017</date>
        <orders>
            <item>
                <product>Chocolates</product>
                <number>777</number>
                <price>250</price>
            </item>
            <item>
                <product>Sweets</product>
                <number>888</number>
                <price>450</price>
            </item>
        </orders>
```

</customer>
</document>

Specifying attributes in a DTD:

We can specify the attributes of the elements in DTD by using <!ATTLIST> element. This element holds a list of attributes for an element. We can specify default values for the attribute and also can specify whether the attribute is necessary for an element or not. The format of defining an attribute in DTD is:

<!ATTLIST ELEMENT_NAME ATTRIBUTE_NAME TYPE DEFAULT_VALUE>

The value supplied for attribute can be:

Value – The default value of an attribute.

#REQUIRED – The attribute is required.

#IMPLIED – The attribute is not required.

#FIXEDVALUE – The attribute value is fixed.

The default type used for attributes is unparsed character data.

Example:

<?xml version=”1.0”?>
<!DOCTYPE document[
<!ELEMENT document (customer)*>
<!ELEMENT customer (name,date,orders)>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT orders (item)*>
<!ELEMENT item (product,number,price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST customer nationality CDATA “INDIAN”
Age CDATA #IMPLIED

```
                    Type        CDATA #REQUIRED>
]>
<document>
        <customer>
                <name>
                        <firstname>aaa</firstname>
                        <lastname>bbb</lastname>
                </name>
                <date>01 jan 2017</date>
                <orders>
                        <item>
                                <product>Chocolates</product>
                                <number>777</number>
                                <price>250</price>
                        </item>
                        <item>
                                <product>Sweets</product>
                                <number>888</number>
                                <price>450</price>
                        </item>
                </orders>
        </customer>
</document>
```

XML Namespaces:

XML document allows us to create our own XML elements with our own element names. This can result in naming collision i.e., different XML elements can have the same name in one XML document. For example:

```
<?xml version=”1.0”>
<college>
        <staff>
```

```
        <name>aaa</name>
        <dept>CSE</dept>
    </staff>
    <student>
        <name>bbb</name>
        <dept>CSE</dept>
    </student>
</college>
```

In the above example the XML elements <name> and <dept> of both staff and student have the common element names. This could provide naming collisions. To solve such problems in XML documents, we use the "XML Namespaces". The XML namespaces allows us to prevent collision by differentiating the elements with same names by using namespace prefixes.

The xmlns attribute:

XML namespaces use the keyword xmlns to create namespace prefixes and assign corresponding URI that uniquely identifies the namespace. For the above example we can define the namespace prefixes to avoid naming collisions.

```
<?xml version="1.0">
<college xmlns:staff="web technologies:staffInfo"
        xmlns:student="web technologies:studentInfo">
    <staff>
        < staff:name>aaa</ staff:name>
        < staff:dept>CSE</ staff:dept>
    </staff>
    <student>
        < student:name>bbb</ student:name>
        < student:dept>CSE</ student:dept>
    </student>
</college>
```

XML Schema:

An XML schema describes the structure of an XML document. XML schema is an XML-

based alternative to DTD.

Limitations of DTD:

1. DTD's do not have the angled bracket syntax.
2. We cannot use multiple DTD's to validate one XML document.
3. DTD's have very limited basic types.
4. DTD's are not object oriented.

An XML schema language is also referred to as "XML Schema Definition(XSD)". An XML schema begins by declaring the XML schema namespace as follows:

<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>

Defining an Element:

There are two types of elements defined in an XML schema. They are:

1. Simple elements.
2. Complex elements.

1. Simple Elements:

Simple elements can contain only data and they cannot have sub elements or attributes. The text they contain can be of various data types. The format for defining a simple element is:

<xsd:element name="element_name" type="xsd:type"/>

2. Complex Elements:

Complex elements can contain sub elements or attributes. The complex elements are made up of simple elements. The format of defining a complex element is:

```
<xsd:complexType name="element_name">
        <xsd:sequence>
                <xsd:element name="element_name" type="xsd:type"/>
                <xsd:element name="element_name" type="xsd:type"/>
        </xsd:sequence>
</xsd:complexType>
```

Defining an Attribute:

An attribute provides additional information used by an element besides the element's content. The format for defining an attribute in XML schema is:

```
<xsd:complexType name="element_name">
    <xsd:complexContent>
        <xsd:extension>
            <xsd:attribute name="attribute_name" type="xsd:type"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Example:

```
<xsd:complexType name="employee">
    <xsd:complexContent>
        <xsd:extension>
            <xsd:attribute name="gender" type="xsd:string"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

An XML schema is a file that we can create using any text editor. The file must be saved with .xsd extension and the contents of the file must confirm to the XML schema language.

Referencing an XML schema:

We link an XML schema by referencing the XML schema file in the <root> tag of the XML document. The format of <root> is:

<root xmlns:xsi=http://www.w3.org/2000.10.XMLSchema-instance xsi:schemalocation="URI">

Example of XML schema:

St.xsd:

```
<?xml version="1.0"?>
    <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
```

```
<xsd:complexType name=”student”>
    <xsd:sequence>
        <xsd:complexType name=”studentType”>
            <xsd:sequence>
            <xsd:element name=”rollnumber” type=”xsd:string”/>
            <xsd:element name=”name” type=”xsd:string”/>
            <xsd:element name=”marks” type=”xsd:integer”/>
            <xsd:element name=”result” type=”xsd:string”/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

studentInfo.xml:

```
<?xml version=”1.0”?>
<root xmlns:xsi=http://www.w3.org/2000.10.XMLSchema-instance xsi:schemalocation=”st.xsd”>
<student>
    <studentType>
        <rollno>95A0</rollno>
        <name>aaa</name>
        <marks>500</marks>
        <result>Firstclass</result>
    </studentType>
    <studentType>
        <rollno>95A1</rollno>
        <name>bbb</name>
        <marks>450</marks>
        <result>Firstclass</result>
    </studentType>
</student>
```

Presenting XML:

In order to present the data in XML documents we use web presentation technologies. Presenting data is again dependent to a specific technology. It is better to separate the data from the presentation technology and stored in XML document and the extensible stylesheet language (XSL) is used to present the data.

Extensible stylesheet language (XSL):

The extensible stylesheet is a language used to express stylesheets which are then used to present XML documents. XSL stylesheets are not same as HTML stylesheets. Rather than creating a style for a particular XML element or class of elements with XSL, a template is created. This template is used to format XML elements which match a specific pattern.

The XSL transformation language (XSLT) is used to transform one XML document from one form to another. The result of applying XSLT to an XML document could be another XML, HTML text or any other document.

Example:

People.xml:

```
<?xml version=”1.0”?>
<?xml-stylesheet type=”text/xsl” href=”people.xsl”?>
<people>
    <person born=”1993”>
        <name>
            <firstname>aaa</firstname>
            <lastname>bbb</lastname>
        </name>
        <profession>Software Engineer</profession>
    </person>
    <person born=”1982”>
        <name>
            <firstname>ccc</firstname>
```

```
                <lastname>ddd</lastname>
            </name>
            <profession>Project Manager</profession>
        </person>
</people>
```

People.xsl:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
<xsl:output method="html" omit-xml-declaration="no"/>
<xsl:template match="/">
<html>
    <head>
        <title>XSLT</title>
    </head>
    <body>
        <table border="2">
            <tr>
                <th>Born</th>
                <th>FirstName</th>
                <th>LastName</th>
                <th>Profession</th>
            </tr>
            <xsl:for-each select="people/person">
            <tr>
                <td><xsl:value-of select="@born"/></td>
                <td><xsl:value-of select="name/firstname"/></td>
                <td><xsl:value-of select="name/lastname"/></td>
                <td><xsl:value-of select="profession"/></td>
            </tr>
            </xsl:for-each>
```

```
            </table>
        </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

The XSL stylesheet begins with a tag called stylesheet which binds with namespace prefix xsl to the URI http://www.w3.org/1999/XSL/Transform which uniquely identifies the XSL namespace.

The <xsl:output> is used to write a XHTML document type declaration to the result tree. Attribute omit-xml-declaration specifies whether the transformation should write the XML declaration to the result tree. <xsl:template> element describes how to transform a particular node from the source tree into the result tree. The match is used to select the document root of the source document.

The element <xsl:for-each> is used to iterate through the source XML document and search for person element. The attribute select specifies the node on which the <xsl:for-each> operates. The element <xsl:value-of>is used to retrieve value of the attribute and elements. The @ symbol specifies that we need to retrieve an attributes value.

Document Object Model (DOM):

There are two models that are commonly used for XML parsers:

1. SAX (Simple API for XML).
2. DOM (Document Object Model).

SAX parser is mainly used when we are dealing with streams of data i.e., the data the XML is passing from one place to another with parser acting as an intermediate way point. Typically this model is used when passing XML data across a network between applications and is widely used by java programmers. SAX based parsers does not have to build any static models of the document in memory and are intended to run quickly.

The SAX parser is not suitable to use in websites where repeated querying and updating of XML document is required. Here it is sensible to build some sort of representation which can be held in memory for the duration of the use of application. In such cases the DOM based parser is a better choice.

DOM stands for Document Object Model and is an API for XML documents. Basically an API is a set of data items and operations which can be used by the developers of application programs. For example, the Microsoft windows environment has a very rich API which is used by the developers when creating windows programs.

The DOM API specifies the logical structure of XML documents and the ways in which they can be accessed and manipulated. The DOM API is just a specification. There isn't a single reference piece of software associated with it which everyone must use. DOM complaint applications include all of the functionality that is needed to handle XML documents. They can build static documents, navigate and search through them, add new elements, delete elements and modify the content of existing elements.

DOM views XML document as a tree, but this is very much logical view of the document. Each XML element is modelled as an object. This means that the node encompass both data and behaviour and that the whole document can be seen as a single complex object. Object oriented theory lets each object have a unique identity. If each node has a unique identity then the tree can be searched for individual nodes. DOM exposes the whole document to application so that the application can manipulate individual nodes.

Accessing XML data using DOM object:

School.xml:

```
<?xml version="1.0"?>
<school>
<class>
<title>XML</title>
<students>
<student>
<first_name>aaa</first_name>
<last_name>bbb</last_name>
</student>
<student>
<first_name>aaa</first_name>
<last_name>bbb</last_name>
```

```
</student>
</students>
</class>
</school>
```

School.html:

```
<html>
<head>
<title>Accessing XML data</title>
<script type="text/javascript">
function getStudentData()
{
var xmldoc;
xmldoc=new ActiveXObject("Microsoft.XMLDOM");
xmldoc.load("school.xml");
nodeSchool=xmldoc.documentElement;
nodeClass=nodeSchool.firstChild;
nodeStudents=nodeClass.lastChild;
nodeStudent=nodeStudents.lastChild;
nodeFirstname=nodeStudent.firstChild;
nodeLastname=nodeFirstname.nextSibling;
outputMessage="Name:"+nodeFirstname.firstChild.nodeValue+'
'+nodeLastname.firstChild.nodeValue;
message.innerHTML=outputMessage;
}
</script>
</head>
<body bgcolor="pink">
<center>
<h1>Accessing XML Data</h1>
<div id="message"></div>
```

```
<input type="button" value="GET DATA" onClick="getStudentData()">
</center>
</body>
</html>
```

Using XML processors: DOM & SAX:

Parsing is a process of reading and validating a program written in one format and converting it to the desired format. XML documents are organized as hierarchal structure similar to a tree. They are well-formed and valid documents. Thus if we have something equivalent to a compiler for XML that can read, validate and convert it. So we can make use of parser for XML documents.

The program that do the process of reading, validating and convert an XML document into the desired format is called as a parser or processor. The parser has to access the XML file and bring it into the memory. Then the parser converts the file into an object, where the object is accessed by an application program.

When an XML document is presented to a program as an object there are two possibilities:

a) Present the document in bits and pieces as we encounter a certain selection or portions of the document. This approach where it goes through the XML document item by item till the end of XML file is called “Simple API for XML (SAX)”.

b) To present the entire document tree at once where the program has access to the XML document as a single object. This approach of presenting entire document as a single object is done by “Document Object Model (DOM)”.