UNIT-III

XML: Document type Definition, XML schemas, Document object model, XSLT, DOM and SAX Approaches. **AJAX A New Approach:** Introduction to AJAX, Integrating PHP and AJAX.

Introduction to XML:

- XML stands for Extensible mark-up Language.
- XML is a mark-up language much like HTML.
- XML was designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is designed to be self-descriptive.
- XML was designed to transport and store data where HTML was designed to display data
- XML is a meta mark-up language i.e. is a language used to develop some other mark-up languages
- XML is a subset of SGML added with some additional services to simplify the language development

Advantages of XML:

1. XML document is human readable and we can edit in any XML document in any text editors. The XML document is language neutral.
2. Every XML document has a tree structure, hence complex data can be arranged systematically and can be understood in simple manner.
3. XML files are independent of an operating system.
4. XML is useful in exchanging data between the applications.
5. XML document is used to extract data from data base.

Disadvantages of XML:

1. XML schema is complex to design and hard to learn
2. XML document cannot be present if the corresponding schema is not found
3. Maintaining the schema for large and complex operations causes less execution speed

DIFFERENCES BETWEEN XML AND HTML

XML	**HTML**
XML is an extensible mark-up language	HTML is Hypertext mark-up language
It has user defined tags	It has only pre – defined tags
User has the control on tags	No control on tags
Case sensitive	Case insensitive
Root element is user defined and only one root element is allowed	Root element is <html>
One can create new mark-up language using XML	There is no such possibility
Self-Description of data can be possible	Self – description of data is not possible

XML DOCUMENT

An XML document is a serialised segment of text which follows the XML standard. An XML document may contain the following items:

- XML Declaration
- Document Type Definition (DTD)
- Elements and its Attributes
- Text
- Processing Instructions
- Comments <!---------->
- Entity References

The following image contains an example:

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

SYNTAX RULES FOR XML DOCUMENT

- XML Documents Must Have a Root Element
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted

XML documents that conform to the syntax rules are said to be "Well Formed" XML documents.

A Well Formed XML document that conforms to the DTD is said to be Valid XML document.

XML DECLARATION

The XML declaration appears as the first line of an XML document. Its use is optional. An example declaration appears as follows:

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

- Version indicates the XML version.
- Encoding indicates how the individual bits correspond to a character set.
- **standalone** indicates whether an external type definitions must be consulted in order to correctly process the document

DOCUMENT TYPE DEFINITION (DTD)

The Document Type Definition (DTD) defines the structure of an XML document. A DTD is a set of rules called declarations, which specify a set of elements and attributes that can appear in a document.

ELEMENTS

Elements describe the data

ATTRIBUTES

Attributes give extra meaning to the data described by an element

COMMENTS

In XML, comments are specified as follows:

<!- - Comments are written to provide readability to readers of the document -->

ENTITY REFERENCES

An entity reference is a group of characters used in text as a substitute for a single specific character. For example, if an attribute must contain a left angle bracket (<), you can substitute the entity reference "<".

Entity references always begin with an ampersand (&) and end with a semicolon (;). You can also substitute a numeric or hexadecimal reference.

The entities predefined in XML are identified in the following table.

XML ENTITY REFERENCES			
Character	**Entity reference**	**Numeric reference**	**Hexadecimal reference**
&	&amp;	&#38;	&#x26;
<	&lt;	&#60;	&#x3C;
>	&gt;	&#62;	&#x3E;
"	&quot;	&#34;	&#x22;
`	&apos;	&#39;	&#x27;

DOCUMENT TYPE DEFINITION

The Document Type Definition (DTD) defines the structure of an XML document. A DTD is a set of rules called declarations, which specify a set of elements and attributes that can appear in a document, as well as the attributes of each element.

An XML document can be tested against DTD to determine whether it conforms to the rules of DTD which describes it.

Its use is optional, and it appears either at the top of the document. DTDs can be defined either internally or in an externally referenced location (a file with .dtd extension).

- A DTD can be embedded in the XML Document, which is called Internal DTD
- A DTD can be stored in a separate file, which is called External DTD. It is allowed to use with more than one XML Document.

SYNTAX

Basic syntax of a DTD is as follows –

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

An example DTD:

```
<!DOCTYPE book [
        <!ELEMENT book(title, author*, version?)>
                <!ELEMENT title (#PCDATA)>
                <!ELEMENT author (#PCDATA)>
                        <!ATTLIST author email CDATA #REQUIRED>
                        <!ATTLIST author office CDATA #REQUIRED>
                        <!ATTLIST author type CDATA "lecturer">
                <!ELEMENT version (number)>
                <!ELEMENT number (#PCDATA)>
]>
```

ELEMENTS

XML elements can be defined as building blocks of an XML. Elements are containers to hold text, elements, attributes, media objects or all of these. And an element can also be an empty element with no content. Each XML document contains one or more elements.

Syntax

Following is the syntax to write an XML element −

<!ELEMENT element-name (list of child elements)>

- Each element can have child elements and they are specified with in brackets ( ) followed by element name.
- Each child element name in the brackets can optionally be followed by a symbol each with a different meaning:

- '+' Element must have one or more child elements.
- '*': Element may have zero or more child elements.
- '?': The existence of child element is optional

The content of an element is of type PCDATA:

PCDATA is the Parsable Character Data which is a string of printable characters except "lessthan", "greaterthan" and "ampersand" and this type of content is handled by XML Parser.

- Content of an element can also be specified as **EMPTY or ANY:**
  - EMPTY specifies that element has not content
  - ANY specifies that element may contain literally any content.

Example:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE garden [
  <!ELEMENT garden (plants*)>
  <!ELEMENT plants (#PCDATA)>
  <!ATTLIST plants category CDATA #REQUIRED>
]>

<garden>
  <plants category = "flowers">Jasmine</plants>
  <plants category = "shrubs"> Strawberry </plants>
</garden>
```

ATTRIBUTES

Attributes define properties of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. An XML attribute is always a name-value pair.
An attribute declaration includes attribute name, its type and a default option.

The General form of declaring attribute to elements is as follows:

```
<!ATTLIST element-name
                Attribute-name_1  attribute-type  default-option_1
                Attribute-name_2  attribute-type  default-option_2
                Attribute-name_3   attribute-type default-option_3

                Attribute-name_n   attribute-type default-option_n
>
```

- ***Attribute_type*** can be either CDATA or Entity
  CDATA is any string of characters except <, >, &
  Entity is a container that holds Complex data
- The default option of an attribute can be either an actual value or a requirement for the value of an attribute in an XML document.

Possible default options for attributes are:

1. A Value: The quoted value, which is used if nothing is specified in an element
2. #FIXED Value: The quoted value, which every element will have and cannot be changed
3. #REQUIRED: No default Value is given, but every instance of the element must have a value
4. #IMPLIED: No default is given, but the value may or may not be specified.

Example:

<!ATTLIST airplane places CDATA "4">
<!ATTLIST airplane engine_type CDATA #REQUIRED>
<!ATTLIST airplane price CDATA #IMPLIED "$1000000000">
<!ATTLIST airplane manufacturer CDATA #FIXED "Cessna">

ENTITIES

Entity is a container that holds Complex data.

- Entities can be defined so that they can be referenced anywhere in the content of the XML Document, such entities are called General Entities.
- Entities can be defined so that they can be referenced only in the DTDs, such entities are called Parameter Entities.

Following is the general form to declare an entity:

<!ENTITY [%] ***entity_name "entity_value"***>

When the optional percent sign % is used in the entity declaration, then it is parameter entity rather than general entity.

Example:

<!ENTITY POS "Point of Sale">

- An entity may also be an external entity, it is used when the entity definition is outside the DTD. and its declaration is as follows:

<!ENTITY ***entity_name*** SYSTEM "File_Location" >

The keyword SYSTEM specifies that the definition of the entity is in different file.

Example:

<!ENTITY JPGPhoto SYSTEM "myEntities/Photo.jpg">

INTERNAL AND EXTERNAL DTDs

Internal DTD

In an Internal DTD the rules for the structure of the document are written with in the XML document itself.

Example:Student.xml

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

<!-- <!DOCTYPE student SYSTEM "student.dtd"> -->

<!DOCTYPE student [<!ELEMENT student (name,address,marks)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT address (#PCDATA)>

<!ELEMENT marks (#PCDATA)>]>

<student>

<name>Yellaswamy</name>

<address>Medchal</address>

<marks>70 percent</marks>

</student>

External DTD

In an External DTD the rules for the structure of the document are written in a separate file and saved with the extension of ".dtd".

How to write DTD include in External File

Syntax

<! DOCTYPE recipes SYSTEM recipe.dtd>

This declaration tells the parser that the XML file uses a DTD which is stored in a file called **recipe.dtd,** and **recipes** is the root element of the XML Document. The keyword SYSTEM indicates that the DTD is created by user which owns on the local system. Internationally agreed DTDs are denoted by the use of keyword PUBLIC.

DATA TYPES OF DTD:

1. CDATA (character data): This type allows all the characters including numbers and space character
2. NMTOKEN: It is same as CDATA but does not accept space character
3. NMTOKENS: It accepts one or more tokens (where one token is a sequence of characters without space character) and in this case space is taken as separator between tokens
4. ID: The value of ID type attribute should be unique it should not start with number but it contain number
5. IDREF: It allows one of the ID type attribute value
6. IDREFS: It can take one or more ID type attribute values where space is the separator
7. ENUM: In this case while declaring attribute we will specify the list of values and it allows using any one of the specified value.
8. ENTITY: It allows one entity name where this entity should be unparsed entity
9. ENTITIES: It allows one or more entity names where space is the separator

XML NAMESPACES

XML Namespaces provide a method to avoid element 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.

XML Namespaces - *xmlns* Attribute:

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.

<element_name xmlns:[*prefix*]= "*URL*" >

Here, prefix is optional, if it is not included, then the namespace is the default namespace.

A prefix is used for two reasons:

1. URLs are too long to be typed on every occurrence of the document
2. URLs include characters that are invalid in XML document

Example:

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

In the example above:

- The ***xmlns*** attribute in the first <table> element gives the h: prefix a qualified namespace.
- The ***xmlns*** attribute in the second <table> element gives the f: prefix a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared in the XML root element:

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

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.

XML SCHEMA

XML Schema is an alternative to "DTD". The XML schemas are used to represent the structure of the XML document. XML schema language is known as XML schema Definition Language(XSD). XML Schema are Written in XML.

The advantage of using XML schema is it supports different data types when compared with DTD and it support name spaces. XML Schema is mainly used to represent the interface for an application which is not possible with DTD.

Most of the XML tools in present days uses XML schema to validate XML document.

XML schema defines elements, attributes, elements having child elements, order of child elements, fixed and default values of elements and attributes. The following is an example of XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="student">
        <xsd:complexType>
        <xsd:sequence>
                <xsd:element name="name"></xsd:element>
                <xsd:element name="address"></xsd:element>
                <xsd:element name="marks"></xsd:element>
        </xsd:sequence>
        </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

XML Data types:

XML Schema defines 44 data types, 19 of which are primitive and 25 of which are derived.

- Primitive types include `String, float, time` and `anyURI`. The predefined data types include `byte, long, decimal, unsignedInt, positiveInt,` and `NMTOKEN`.

- User – defined types are also defined by imposing some restrictions on existing types (base type), and the restrictions are called **facets** of the base type. For example the `integer` data type has eight possible facets: `totalDigits, maxInclusive, maxExclusive, minInclusive, minExclusive, pattern, enumeration` and `whitespace.`

There are two categories of data types: Simple and Complex

SIMPLE TYPES:

- A simple data type is a data type whose content is restricted to strings. A simple type cannot have attributes or does not include child elements
- Simple data types are defines using the **element** tag. To specify the name of the element **name attribute** is used and to specify the data type of the element **type attribute** is used.

Examples:

1. **String Data Type:** The string data type is used to define the element containing characters.

For example

student.xsd

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

student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<student xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="student.xsd">Naresh </student>
```

2. **Date Data Type**: Date data type is used to specify the date.
   The format of this date is yyyy-mm-dd denotes year, month and date. For example

student.xsd

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

student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dob xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="student.xsd">2001-09-01
</dob>
```

3. Numeric Data Type: To represent decimal or integer values Numeric Data Type is used.

For example

student.xsd

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

student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<salary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi: schemaLocation="student.xsd">9876.54
</salary>
```

4. Boolean Data Type: for specifying true or false Boolean data type is used. For example

student.xsd

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

student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<flag xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="student.xsd"> true
</flag>
```

COMPLEX TYPES:

- A complex type can have attributes and include child elements, these are called element – only elements.
- Complex types are defined with **complexType** tag**.** The elements that are the content of an element – only element must be contained in an ordered group or an unordered group.
- An ordered group of elements are defined using the **sequence** tag.

Example:

```
<xsd:complexType    name= “sports_car”>
        <xsd:sequence>
                <xsd:element name = “make” type = “xsd:string” />
                <xsd:element name = “model” type = “xsd:string” />
                <xsd:element name = “engine” type = “xsd:string” />
                <xsd:element name = “year” type = “xsd:decimal” />
        </xsd:sequence>
</xsd:complexType>
```

Differences between XML Schema and DTD:

XML Schema	DTD
1. XML schema’s are suitable for large applications	1. DTD’s are suitable for small applications
2. XML schema’s support for defining the type of data.	2. DTD’s does not support
3. XML schema’s support namespace	3. DTD’s does not support namespace.
4. XML schema has large number of built in and derived data types.	4. DTD’s has limited number of data types

DOCUMENT OBJECT MODEL

DOM is an acronym stands for Document Object Model. It defines a standard way to access and manipulate documents. The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

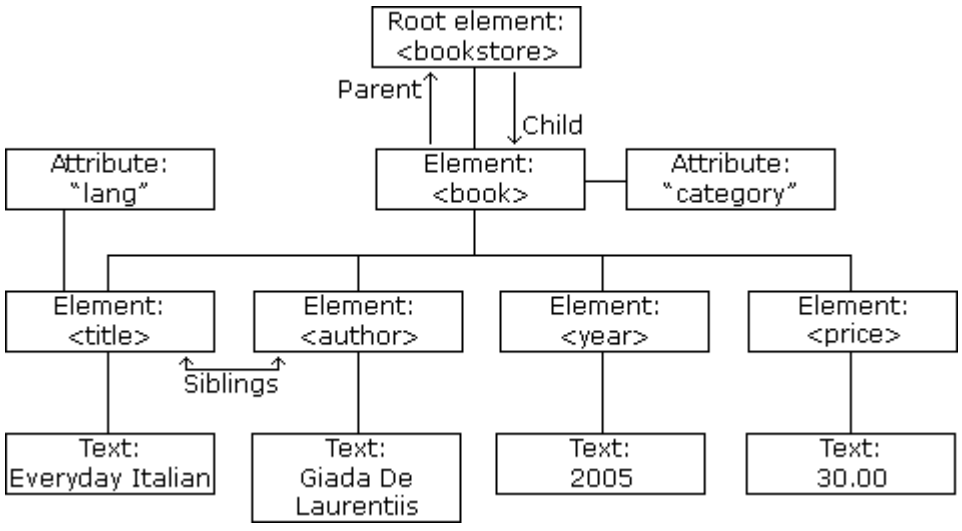
As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The Document Object Model can be used with any programming language.

XML DOM defines a standard way to access and manipulate XML documents. The XML DOM makes a tree-structure view for an XML document. We can access all elements through the DOM tree. We can modify or delete their content and also create new elements. The elements, their content (text and attributes) are all known as nodes.

Consider the below 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>
</bookstore>
```

Document Object Model represents this XML Document as follows:



The below example loads a text string into an XML DOM object, and extracts the info from it with JavaScript:

```
<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");
var txt= xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
document.getElementById("demo").innerHTML = txt;
</script>
</body>
</html>
```

Explanation:

- xmlDoc - the XML DOM object created by the parser.
- getElementsByTagName("title")[0] - get the first <title> element
- childNodes[0] - the first child of the <title> element (the text node)
- nodeValue - the value of the node (the text itself)

XML DOM Properties

These are some typical DOM properties:

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

In the list above, x is a node object.

XML DOM Methods

- x.getElementsByTagName(*name*) - get all elements with a specified tag name
- x.appendChild(*node*) - insert a child node to x
- x.removeChild(*node*) - remove a child node from x

In the list above, x is a node object.

XSLT (eXtensible Stylesheet Language Transformation)

XSL stands for EXtensible Stylesheet Language. It is a styling language for XML just like CSS is a styling language for HTML.

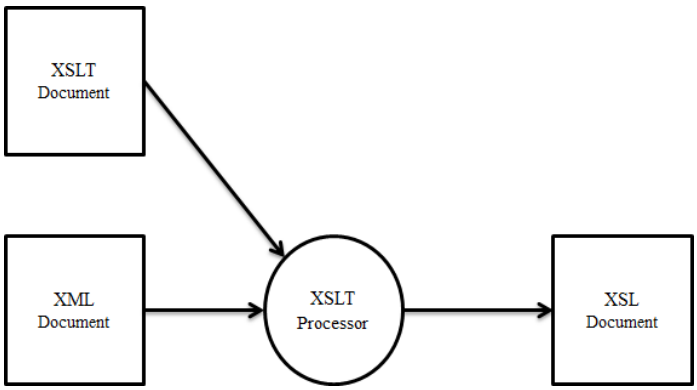
XSLT stands for XSL Transformation. It is used to transform XML documents into other formats (like transforming XML into HTML).

In **HTML** documents tags are predefined, but, in **XML** documents, tags are not predefined. World Wide Web Consortium (W3C) developed XSL to understand and style an XML document, which can act as XML based Stylesheet Language.

MAIN PARTS OF XSL DOCUMENT

- **XSLT**: It is a language for transforming XML documents into various other types of documents like HTML, Plaintext, etc.
- **XPath**: It is a language used to identify parts of the XML documents.
- **XQuery**: It is a language for querying XML documents.
- **XSL-FO**: It is a language for formatting XML documents.

HOW XSLT Processor WORKS



- XSLT processor takes both an XML document and an XSLT document as input.
- The XSLT document is the program to be executed. XML document is the input to this program.
- Parts of the XML document are selected, possibly modified and merged with the parts of the XSLT document to form a new document called as XSL Document.
- This XSL document is also an XML document, and it can be stored for future use for any applications or immediately displayed by the browser.

Example: **XML Document:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet  type="text/xsl"   href="cdscss.xsl" ?>
<catalog>
        <cd>
```

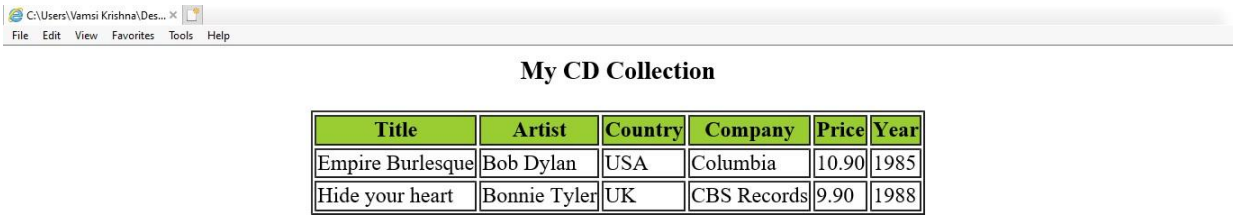
```
        </cd>
        <cd>



        </cd>
 </catalog>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>



<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
```

XSLT Document:

```
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <html><body><h3 align="center">My CD Collection</h3>
        <table border="1" align="center">
                <tr bgcolor="#9acd32">
                <th>Title</th>
                <th>Artist</th>
                <th>Country</th>
                <th>Company</th>
                <th>Price</th>
                <th>Year</th>
                </tr>
                <xsl:for-each select="catalog/cd">
                <tr>
                <td><xsl:value-of select="title"/></td>
                <td><xsl:value-of select="artist"/></td>
                <td><xsl:value-of select="country"/></td>
                <td><xsl:value-of select="company"/></td>
                <td><xsl:value-of select="price"/></td>
                <td><xsl:value-of select="year"/></td>
                </tr></xsl:for-each> </table>
 </body>
 </html>
```

</xsl:template>
</xsl:stylesheet>

Outcome:



DOM AND SAX APPROACHES

Purpose of XML Processor

1. Processor check the basic syntax of document for the well – formedness
2. Processor replace all the references to the entities in an XML Document
3. Attributes in DTD and elements in XML Schemas can specify their values in an XML document have default values, which must be copied into XML document during processing
4. Validating Parser which is the part of the processor conforms the structure of the XML document with either XML Schema or DTD

Two popular approaches for XML processor are: SAX and DOM

SAX Approach

The Simple API for XML (SAX) standard, which was released in May 1998, was developed by an XML users group, XML – DEV. SAX has been widely accepted standard and is now widely supported XML processors.

The SAX approach to processing called *Event Processing*. The processor scans the XML document from beginning to end. Every time a syntactic structure such as opening tag, closing tag, attributes and text of the document is recognized, the processor signals an event to the application by calling an event handler for the particular structure that was found. The interfaces that describe the event handlers form the SAX API.

DOM Approach

The natural alternative to the SAX approach to XML document parsing is to build a hierarchical structure to the document. Such a hierarchical structure is built using DOM approach so that the created documents are dynamic. The parser of the XML processor constructs this DOM tree. Each element of XML document is considered as an object and it can be accessed and modified by the application. When parsing is complete, the DOM representation of the XML document is in memory and can be accessed in number of different ways.

Advantages of DOM over SAX

1. It is possible to access the random parts of the document.
2. Any part of the document can be accessed more than once.
3. It is also possible to rearrange the elements of the document.

Advantages of SAX over DOM

1. The DOM structure is stored entirely in memory, so that large sized XML documents require a great deal of memory.
2. Since there is no limit on the size of the XML document, some documents cannot be parsed with the DOM method.

The process of building the DOM structure of an XML document requires some syntactic analysis of the document, which is done by SAX parser. In most DOM parsers include a SAX parser.

AJAX

WHY AJAX?

AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating

better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script. AJAX is a process of using asynchronous requests from the web browser to the server to fetch data which is used to update a part of the browser displayed document.

A traditional (non – Ajax) session of web use begins with the user requesting an initial document. At that point the browser is blocked from activity while it waits for the server to provide the requested document. When the document arrives, the browser replaces the former display with a rendering of new document. And the user interactions with the displayed document may require that only relatively small parts of the document be modified or updated. In a non – Ajax web application, even the smallest change in the displayed document requires the same process that produced the initial display. The request must go to the server, the server must construct and send back the document and the whole display must be rendered. During this time, browser is blocked and goes into wait state. If the web application requires many such interactions, it took a lot of time.

Ajax is meant to increase the speed of user interactions with web applications. For those requests that update only a small part of the document, Ajax technology shortens the required time for both transmitting and rendering the document. It does this by having the server provide only a relatively small part of the document – the part that must change. This shortens the transmission time because the document that is being transmitted is much smaller and the rendering time is also shortened.

Another key feature of Ajax is that the requests from the browser to the server are asynchronous. This means that when the browser requests only part of the document from the document, it doesn‟t need to lock while it waits for response. Both the user and browser can continue with some other task during the time it takes to fetch and update the part of the document.

INTRODUCTION TO AJAX:

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.

- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

RICH INTERNET APPLICATION TECHNOLOGY

AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

AJAX is based on the following open standards −

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen dynamically.

AJAX TECHNOLOGIES

AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages.

JavaScript

- Loosely typed scripting language.
- JavaScript function is called when an event occurs in a page.
- Glue for the whole AJAX operation.

DOM

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents.

CSS

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

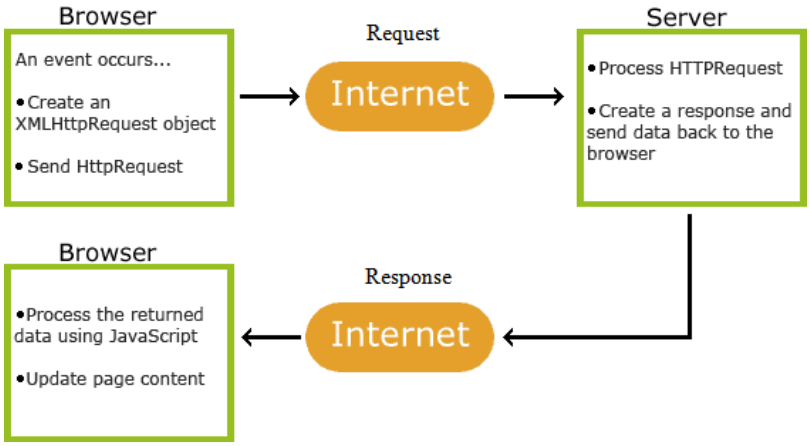
XMLHttpRequest

- JavaScript object that performs asynchronous interaction with the server.

HOW AJAX WORKS?

When user makes a request, the browser creates an object for the HTTP request and a request is made to the server over an internet. The server processes this request and sends the required data to the browser. At the browser side the returned data is processed using java script and the web document gets updated accordingly.

Following figure illustrates this working:



The keystone of AJAX is the **XMLHttpRequest** object. Every modern browser supports **XMLHttpRequest** object. But Old versions of Internet Explorer (IE5 and IE6) use an **ActiveX** object instead of the **XMLHttpRequest** object. All modern browsers (Chrome, Firefox, IE7+, Edge, Safari Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable_name = new XMLHttpRequest();

Example:

var xhttp = new XMLHttpRequest();

Syntax for creating an ActiveX object:

Variable_name= new ActiveXObject("Microsoft.XMLHTTP");

Example:

var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");

XMLHttpRequest Object Methods

Method	Description
XMLHttpRequest( )	Creates a new XMLHttpRequest object
abort ( )	Cancels the current request
getAllResponseHeaders( )	Returns header information
open(method, url, async, user, psw)	Specifies the request **method:** The request type GET or POST **url:** The file location **async:** True (asynchronous) or false (synchronous) **user:** optional user name **psw**: optional password
send( )	Sends the request to the server. Used for GET requests
send (string)	Sends the request to the server. Used for POST requests

setRequestHeader( )	Adds a label/value pair to the header to be sent
getResponseHeader( )	Returns specific header information

XMLHttpRequest Object Properties

Property	**Description**
onReadyStateChange	Defines a function to be called when the **readyState** property changes
readyState	Holds the status of the XMLHttpRequest 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" *For a complete list go to the Http Messages Reference*
statusText	Returns the status-text (e.g. "OK" or "Not Found")

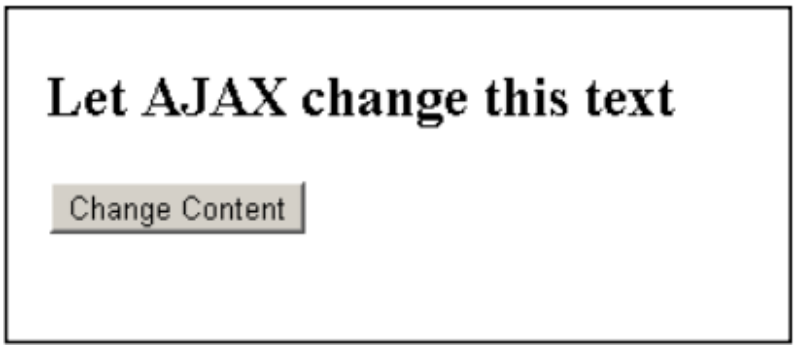
Example:

```
<html>
<head>
<script>
	function myfun()
	{
		var req;
		if (window.XMLHttpRequest)
		{
			req=new XMLHttpRequest();
		}
		else
		{
			req=new ActiveXObject("Microsoft.XMLHTTP");
		}
		req.onreadystatechange=function()
		{
			if (req.readyState==4 && req.status==200)
			{
```

```
                        document.getElementById("myDiv").innerHTML=req.responseText;
                    }
                }
                req.open("GET","ajax_info.txt",true);
                req.send();
            }
</script>
</head>
<body>
```

```
    <div id="myDiv"><h2>Let AJAX change this text</h2></div>
    <button type="button" onclick="myfun()">Change Content</button>
</body>
</html>
```

On load: Document looks as follows:



On Click: Document changes dynamically

ADVANTAGES AND DISADVANTAGES OF AJAX:

Ajax is a set of web development techniques using many web technologies on the client-side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behaviour of the existing page.

Any other technology Ajax also has its own pros and cons. Let's look at some of those.

Pros −

- Allows applications to render without data and fill data as the application gets it from the server.
- Gives platform independence to application developers
- Faster page renders
- More responsive applications
- No re-rendering of whole pages is needed to update only a single area.

Cons−

- Any user whose browser does not support JavaScript or XMLHttpRequest, or has this functionality disabled, will not be able to properly use pages that depend on Ajax.
- Multiple server requests need more data consumed at the client-side.
- Failure of any one request can fail the load of the whole page.
- Browsers with JS disabled will not be able to use pages using ajax

INTEGRATING PHP AND AJAX

We can use Ajax along with XML and PHP. In the following example we will discuss how, HTML file along with java script communicates with XML and PHP. It will work as follows:

1. HTML displays the form that contains a drop down list. User can select his/her friend`s name from the dropdown list.

2. When user selects some name, a function named **showNames** will be triggered. This function is defined in java script file.

3. This function in java script file will send the name as a query string to some PHP file. The name of the PHP file is considered as URL

4. The PHP file makes use of DOM. It will load XML file using DOM. Using DOM object we go through each node of XML file and retrieve the contents.

5. These contents are then returned to the HTML using the **innerHTML**. Hence on browser we can get the details of the friend whose name we have selected.

Step1: Create an HTML document for displaying the form.

AJAXDemo.html

```
<html>
<head>
<script src=”testing.js”></script>
</head>
<body>
<form>
Select a Name:
        <select name=”names”onchange=”showNames(this.value)”>
        <option value=”chitra”>chitra</option>
        <option value=”priyanka”>priyanka</option>
        <option value=”Raj”>Raj</option>
        </select>
</form>
        <p>
                <div id=”txtHint”><b>Friend Details:</b></div>
```

```
        </p>
</body>
</html>
```

Step2: The java script will be as follows. It would contain function **showNames**.

Testing.js

```
var xmlHttp;
function showNames(str)
{
        xmlHttp=GetxmlHttpobject();
        if(xmlHttp==null)
        {
                alert("Browser does not support HTTP Request");
                return;
        }
        var url="getInfo.php";
        url=url+"?q="+str;
        url=url+"&sid="Math.random();
        xmlHttp.onreadystatechange=statechanged;
        xmlHttp.open("GET",url,true);
        xml.Http.send(null);
}
function statechanged()
{
        if(xmlHttp.readystate==4||xmlHttp.status==200)
        {
        document.getElementBYId("txtHint").innerHTML=xmlHttp.responseText;
        }
}
function GetxmlHttpobject()
{
        var xmlHttp=null;
        try
        {
        //Firefox,opera 8.0+,safari
        xmlHttp=new XMLHttpRequest();
        }
        catch(e)
        {
        //Internet Explorer
                try
                {
                xmlHttp=new ActiveXobject("Msxml2.XMLHTTP");
```

```
}
catch(e)
{
```

```
                xmlHttp=new ActiveXobject("Microsoft.XMLHTTP");
                }
        }
return xmlHttp;
}
```

Step3: The PHP script that normally runs on the server side is as given below. It will make use of DOM to load and handle the XML file.

getinfo.php

```
<?php
        $q=$_GET["q"];
        $xmlDoc=new DOMDocument();
        $xmlDoc->load("FriendNames.xml");
        $a=$xmlDoc->getElementsByTagName(„name");
        for($i=0;$i<=$a->length-1;$i++)
        {
        //going through elements
                if($a->item[$i]->nodeType==1)
                {
                        if($a->item[$i]->childNodes->item[0]->nodevalue==$q)
                        {
                                $b=($a->item[$i]->parentNode);
                        }
                }
         }
$friend=($b->childNodes);
for($i=0;$i<$friend->length;$i++)
{
//going through other elements
        if($friend->item[$i]->nodeType==1)
        {
                echo("<b>".$friend->item[$i]->nodeName.":</b>");
                echo($friend->item[$i]->childNodes->item[0]->nodevalue);
                echo("<br/>");
        }
}
?>
```

Step4: The XML file which is handled by the PHP in the above step is:

FriendNames.xml

```
<?xml version=”1.0”?>
<friend>
        <info>
                <name>chitra</name>
                <phone>1111111111</phone>
                <email>chitra_abc@gmail.com</email>
                <hobby>singing</hobby>
        </info>
        <info>
                <name>priyanka</name>
                <phone>2222222222</phone>
                <email>pr123@rediffmail.com</email>
                <hobby>reading</hobby>
        </info>
        <info>
                <name>raj</name>
                <phone>333333333</phone>
                <email>raj_2008@hotmail.com>/email>
                <hobby>photography</hobby>
        <info>
</friend>
```

Step5: for getting the output we will open the HTML file (Created in step 1) in browser window