BSc CsIt SEM - V
Web technology

Unit 5: Ajax and XML

Syllabus:

Unit 5: AJAX and XML (7 Hrs.)
Basics of AJAX; Introduction to XML and its Application; Syntax Rules for creating XML document; XML Elements; XML Attributes; XML Tree; XML Namespace; XML schema languages: Document Type Definition(DTD), XML Schema Definition (XSD); XSD Simple Types, XSD Attributes; XSD Complex Types; XML Style Sheets (XSLT), XQuery

AJAX

Ajax, which stands for Asynchronous JavaScript and XML, is a set of techniques for creating highly interactive web sites and web applications. The idea is to make what's on the Web appear to be local by giving you a rich user experience, offering you features that usually only appear in desktop applications.

The emphasis in Ajax applications is to update the web page, using data fetched from the Internet, without refreshing the web page in the browser. You saw an example of that with Google Suggest, where a drop-down list appears in the browser without a page refresh.

The term "Ajax" was created by Jesse James Garrett, president of Adaptive Path, in a February 18, 2005 article collecting the technologies that already existed, and which make up Ajax, under one umbrella term.

Ajax incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the Document Object Model;
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- and JavaScript binding everything together.

Ajax is an umbrella term for techniques you use to make web applications look like desktop applications. Here's how it works: In the browser, code written in a scripting language—most frequently, JavaScript, watches what information the user wants, such as what term they're searching for in Google Suggest. When, or even before, the user needs that information, the JavaScript code communicates with the web server behind the scenes to fetch that information without causing a page refresh in the browser.

That is, the way Ajax fetches data from the server is invisible to the user. The JavaScript code uses a special object built into the browser—an XMLHttpRequest object—to open a connection to the server and download data from the server. That data is often in XML format, but it can be just plain text.

When the data that the user needs has been downloaded behind the scenes, the JavaScript code uses that data to update the display in the browser. For example, in the

Google, JavaScript was responsible for fetching, behind the scenes, the suggestions Google made and then displaying those suggestions in the drop-down list box after they were downloaded.

In other words, …

- 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.

Hence,

- AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.
- it uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- 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.

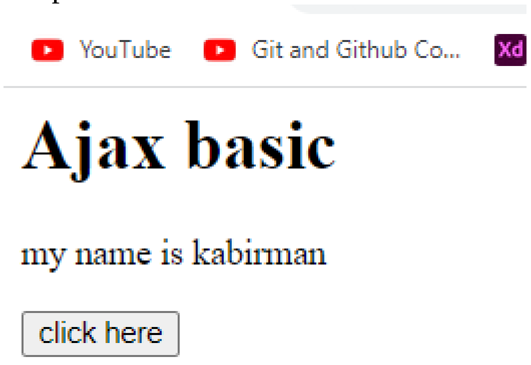
Example:

Employee.html

```
<html>
<head>
<title>Ajax example</title>
   <script type="text/javascript">
   function empdetail()
      {

         var myobj=new XMLHttpRequest();
         myobj.onreadystatechange=function()
         {
            if(this.readyState==4 && this.status==200)
               {
                  document.getElementById("pid").innerHTML=this.responseText;
               }
         };
         myobj.open("GET","employee.txt",true);
         myobj.send();


      }
```

```
    </script>

</head>
    <body>
    <h1>Ajax basic</h1>
    <p id="pid">this is the field , will change by ajax</p>
    <input type="button" id="mubutton" onclick="empdetail()" value="click here">
    </body>


</html>
```

Output:



XML

XML (Extensible Markup Language) is a markup language similar to HTML, but without predefined tags to use. Instead, you define your own tags designed specifically for your needs. This is a powerful way to store data in a format that can be stored, searched, and shared. Most importantly, since the fundamental format of XML is standardized, if you share or transmit XML across systems or platforms, either locally or over the internet, the recipient can still parse the data due to the standardized XML syntax.

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

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

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

Use of XML

XML is one of the most widely-used formats for sharing structured information between programs, between people, between computers and people, both locally and across networks.

Hence…

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

Structure of an XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <to>Students</to>
  <from>Teacher</from>
  <subject>Regarding assignment submission</subject>
  <text>All students will have to submit assignment by tomorrow.</text>
</message>
```

Below is the explanation of each point.

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

- This line is called XML Prolog or XML declaration.
- This line is optional i.e, it can be either used or not in an XML document. However, it should be the very first line if used.
- The version="1.0″ is the version of the XML currently used.
- The encoding=" UTF-8″ specifies the character encoding used while writing an XML document, for example, êèé is for French and so on. Its default value is "UTF-8".
- This declaration is case sensitive for example "xml" should must be in lower case .

<message>

- This is the root element and all the remaining elements <to>, <from> etc are the child element and they reside within the root element.
- Every XML file should have one or more Root elements to avoid error and It is case sensitive.

<to>, <from>, <subject>, <text>

- These elements are the child element and they reside within the root element <message>.

Syntax Rules for creating XML document

1. Root Element is mandatory in XML

XML document must have a root element. A root element can have child elements and sub-child elements.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
  <to>Steve</to>
  <from>Paul</from>
  <subject>Message from teacher to Student</subject>
  <text>You have an exam tomorrow at 9:00 AM</text>
</message>
```

In The above XML document, <message> is the root element and <to>, <from>, <subject> and <text> are child elements.

2. XML is case sensitive

XML is a case sensitive language.

For example:

This is valid :<from>Paul</from>
This is invalid: <from>Paul</From>

3. XML Prolog

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

This line is called the XML Prolog. It is an optional line, however it should be the first line when you mention it. It specifies the XML version and the encoding used in the XML document.

4. Elements should not overlap

All the elements in XML should be properly nested and they should not overlap.

<class><teacher>Rick</class></teacher> -->Wrong (Not nested properly)
<class><teacher>Rick</teacher></class> -->Correct (Correctly nested)

5. Attributes in XML

An opening tag in XML can have attributes, these attributes are name & value pairs. Attribute names are case sensitive and should not be in quotation marks. Attribute values should be in single or double quotation.

<text category = "message">You have an exam tomorrow at 9:00 AM</text>
Here category is the attribute name and message is the attribute value.

6. XML elements must have a closing tag

All XML documents must have a closing tag.

<text category = message>hello</text> -->correct

<text category = message>hello -->wrong

7. Comments in XML

This is how a comment should look like in XML document.
<!-- This is just a comment -->

8. White-spaces are preserved in XML

Unlike HTML that doesn't preserve white space, the XML document preserves white spaces.

XML Elements and Attributes

XML Elements

The XML elements are the basic building block of the XML document. It is used as a container to store text elements, attributes, media objects etc. Every XML documents contain at least one element whose scopes are delimited by start and end tags or in case of empty elements it is delimited by an empty tag.

Syntax:

<element-name attributes> Contents...</element-name>

- **element-name**: It is the name of element.
- **attributes:** The attributes are used to define the XML element property and these attributes are separated by white space. It associates the name with a value, which is a string of characters.

Example:

```
<?xml version = “1.0”?>
<contactinfo>
   <address category = “collage”>
      <name>Amar</name>
      <College>Nccs</College>
      <mobile>123454567890</mobile>
   </address>
</contactinfo>
```

Above is the example of an XML document describing the address of a college student using XML elements.

Rules to define XML elements:

- An element an contain alphanumeric values or characters. But only three special characters are required in the names these are hyphen, underscore and period.
- Names are case sensitive. It means lower case letters have different meaning and upper case characters have different meaning. For example address, Address, aDDress are different names.
- Both start and end tags for elements need to be same.
- An element, which is a container, can contain text or elements

XML attribute

The XML attribute is a part of an XML element. The addition of attribute in XML element gives more precise properties of the element i.e, it enhances the properties of the XML element.

Syntax:

<element_name attribute1 attribute2 ... > Contents... </element_name>

In the above syntax element_name is the name of an element which can be any name. The attribute1, attribute2, … is XML attribute having unique attribute name. Then in the content section, any message can be written and at the end, the element name is ended.

Example:

<address category = "collage">

In the above example, XML element is **address,** the **category** is the attribute name and collage are the attribute value, Attribute name and its value always appear in pair. The attribute name is used without any quotation but attribute value is used in single ('') or double quotation ("").

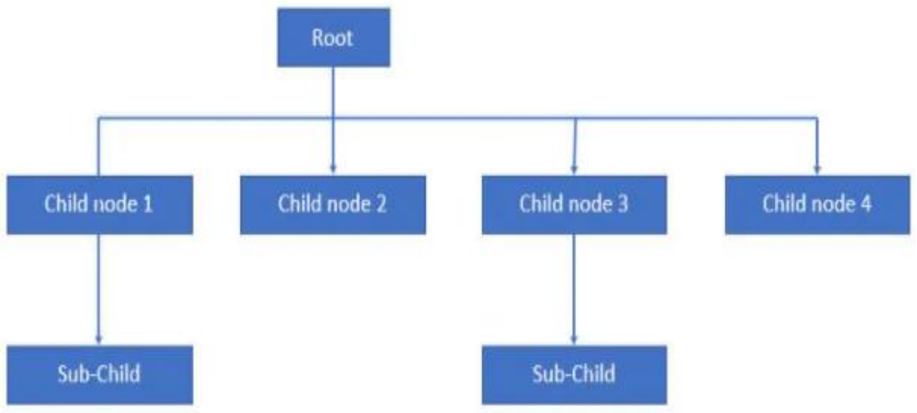
XML Tree

XML tree structure is also called as tree model or hierarchical model. An XML document has a self-descriptive structure. The tree structure is often referred to as XML Tree and plays an important role to describe any XML document easily.

The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root. The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes.

Syntax

The basic Syntax structure is drawn below:



How to design a tree in XML?

At first, we shall see an XML document, and based on this; we can design a tree structure as the tree structure is the best supportive of XML language, which has the capability of deriving positions, powerfully combining hierarchical formats. We have some rules for designing a structure and a relationship between the two elements in XML Document. The rules are stated as follows:

- **Rule 1 – Descendants:** If the XML element 'X' is contained by another element 'Y', then X is Y's descendant.
- **Rule 2 – Ancestors:** If the XML element 'X' has element 'Y', then Y is the ancestors of X.
- **Rule 3** – The name of the label in the section data should be relative.

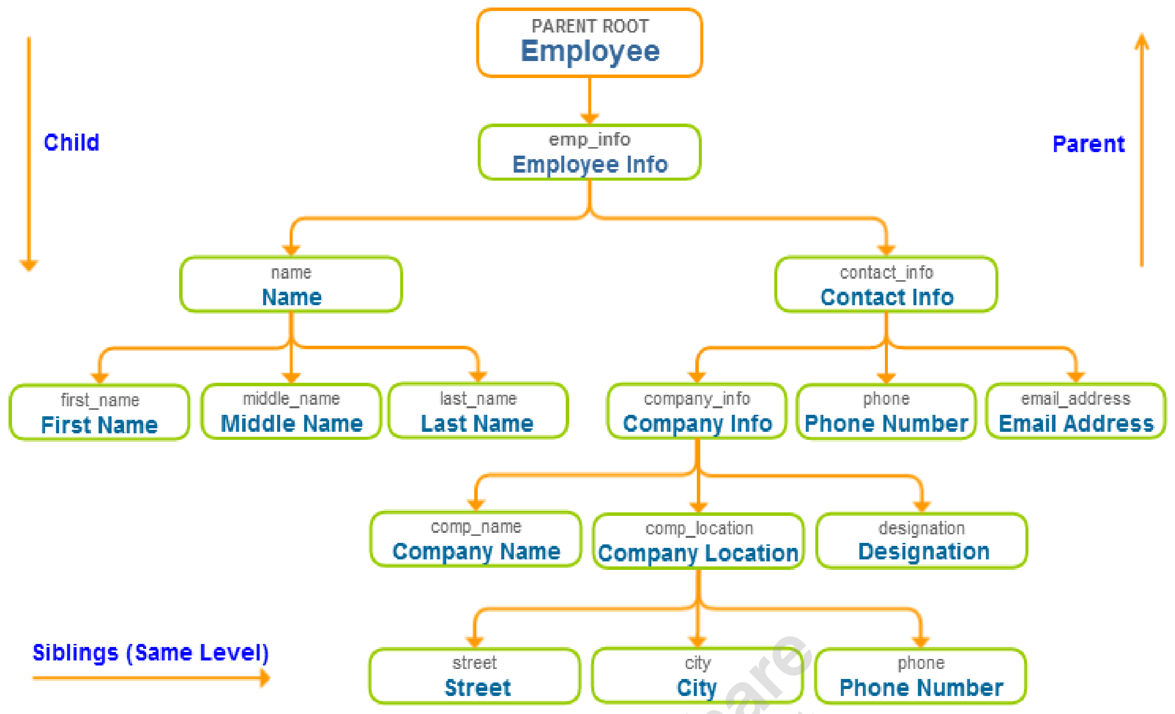
<u>Examples of XML Document</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
<emp_info id="1">
   <name>
     <first_name>Teksan</first_name>
     <middle_name>Gharti</middle_name>
     <last_name>magar</last_name>
   </name>
   <contact_info>
     <company_info>
       <comp_name>Capgemini</comp_name>
       <comp_location>
         <street>Tower-1, Infocity</street>
         <city>mumbai</city>
         <phone>000-478-1414</phone>
       </comp_location>
       <designation>Software Developer</designation>
     </company_info>
   <phone>000-987-4745</phone>
   <email>email@myemail.com</email>
   </contact_info>
  </emp_info>
</employee>
```

In this example, the first line states version of the XML document as "1.0". So, here <employee> is the root element followed by many child elements.

<u>XML Tree</u>

The Tree structure for the above xml document is shown below:



XML Namespace

In XML, elements name is defined by the developer so there is a chance to conflict in name of the elements. To avoid these types of conflictions we use XML Namespaces. We can say that XML Namespaces provide a method to avoid element name conflict. Generally, this conflict occurs when we try to mix XML documents from different XML application.

So,

An XML namespace is a collection of names that can be used as element or attribute names in an XML document. The namespace qualifies element names uniquely on the Web in order to avoid conflicts between elements with the same name. The namespace is identified by some Uniform Resource Identifier (URI), either a Uniform Resource Locator (URL), or a Uniform Resource Name (URN), but it doesn't matter what, if anything, it points to. URIs are used simply because they are globally unique across the Internet.

Namespace Declaration

Declaration of the namespace in the XML document is very important which are done by making use of a family of reserved attributes. And these attributes can be put on the way directly or default in the following syntax:

Syntax

<element xmlns:name = "URL">

Here,

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

Example1

Following is a simple example of XML Namespace …

```
<?xml version = "1.0" encoding = "UTF-8"?>
<cont:contact xmlns:cont = "www.xyz.com/profile">
  <cont:name>Ram </cont:name>
  <cont:company>Reliance</cont:company>
  <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

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

Let's take an example with two tables:

Table1:

```
<table>
  <tr>
    <td>Apple</td>
    <td>Banana</td>
  </tr>
</table>
```

Table2:

This table carries information about a computer table.

```
<table>
  <name>Computer table</name>
  <width>80</width>
  <length>120</length>
</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.

Hence to get rid of name conflict we use XML namespace.

1) By Using a Prefix

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

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


<f:table>
  <f:name> Computer 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.

2) By Using xmlns Attribute

You can use xmlns attribute to define namespace with the following syntax:

<element xmlns:name = "URL">

Example

```
<root>
<h:table xmlns:h="http://www.abc.com/TR/html4/">
  <h:tr>
    <h:td>Apple</h:td>
    <h:td>Banana</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.xyz.com/furniture">
  <f:name>Computer table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

In the above example, the <table> element defines a namespace and when a namespace is defined for an element, the child elements with the same prefixes are associated with the same namespace.

Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<purchaseOrder xmlns="http://www.somecompany.com/order/PO>
    <firstname>Fred</firstname>
    <surname>Bloggs</surname>
    <address xmlns="http://www.somecomapany.com/order/ADDR">
        <addressLine1>2 South Road</address1>
        <addressLine2/>
        <town>Colchester</town>
        <county>Essex</county>
        <postcode>CO8 9SR</postcode>
    </address>
    <telephone>01334 234567</telephone>
</purchaseOrder>
```

The above example the first namespaces is defined within the purchaseOrder element and the other is defined within the address element. The final part of the URI (/PO and /ADDR in the example) completes the URL to create a unique identifier for each namespace. All elements descending the purchaseOrder element will inherit the namespace. This also applies to all elements descending address element.

Namespaces can be declared either **explicitly or by default**. With an explicit declaration, you define a **shorthand, or prefix,** to substitute for the full name of the namespace. You use this prefix to qualify elements belonging to that namespace. A **default declaration** declares a namespace to be used for all elements within its scope, and a prefix is not used.

The Default Namespace

The default namespace is used in the XML document to save you from using prefixes in all the child elements.

The only difference between default namespace and a simple namespace is that: There is no need to use a prefix in default namespace.

You can also use multiple namespaces within the same document just define a namespace against a child node.

Example of Default Namespace:

```
<mynotes xmlns="http://www.xyz.com/web-note">
 <note>
   <title>web-tutorial</title>
   <author>xyz</author>
 </note>
 ...
</ mynotes >
```

Using default namespaces can be complicated when many are declared within the XML tree. It is recommended to use prefixed namespaces. A qualified name assigns a prefix to an element that maps to a declared namespace.

It is not required to qualify all elements within the XML. You can rely on default namespace to determine the namespace for all elements not explicitly prefixed. This is shown in the example below:

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<purchaseOrder xmlns="http://www.somecompany.com/order/PO
```

```
   xmlns:addr="http://www.somecomapany.com/order/ADDR>
   <firstname>Fred</firstname>
   <surname>Bloggs</surname>
   <addr:address">
      <addr:addressLine1>2 South Road</addr:address1>
      <addr:addressLine2/>
      <addr:town>Colchester</addr:town>
      <addr:county>Essex</addr:county>
      <addr:postcode>CO8 9SR</addr:postcode>
   </addr:address>
   <telephone>01334 234567</po:telephone>
</purchaseOrder>
```

In the example above, the namespace declaration is moved to the purchaseOrder element. This is optional and can be declared within the address element. Adding the namespace gives the address namespace scope throughout the whole of the document and is not restricted to the address element.

Namespaces declared explicitly

An element using a prefix is defined with two parts, a prefix and a local name separated with a **":"** e.g addr:address. All descendants of the address element should also be prefixed as shown in the example below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<po xmlns:po="http://www.somecompany.com/order/PO
   xmlns:addr="http://www.somecomapany.com/order/ADDR>
   <po:firstname>Fred</po:firstname>
   <po:surname>Bloggs</po:surname>
   <addr:address">
      <addr:addressLine1>2 South Road</addr:address1>
      <addr:addressLine2/>
      <addr:town>Colchester</addr:town>
      <addr:county>Essex</addr:county>
      <addr:postcode>CO8 9SR</addr:postcode>
   </addr:address>
   <po:telephone>01334 234567</po:telephone>
</po:purchaseOrder>
```

XML schema languages: Document Type Definition(DTD), XML Schema Definition (XSD)

An XML schema is the structural layout of an XML document, expressed in terms of constraints and contents of the document. Constraints are expressed using a combination of the following:

- Grammatical rules governing the order of elements
- Data types governing an element and content attribute
- Boolean predicates that the content has to satisfy
- Specialized rules including uniqueness and referential integrity constraints

Document Type Definition (DTD)

The XML DTD (Document Type Declaration) is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

In short…

- DTD stands for Document Type Definition.
- It is used to validate the XML documents.
- XML provides facility to create your own DTDs for XML document.
- DTD specifies the structure of the XML document.
- DTD is part of the file or separate document file.

Hence, the purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in your XML document, or as an external reference.

Syntax:

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

Here,

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

Types of DTD

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

There are two types of DTDs:
a) Internal DTD.
b) External DTD.

Internal DTD

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

Syntax

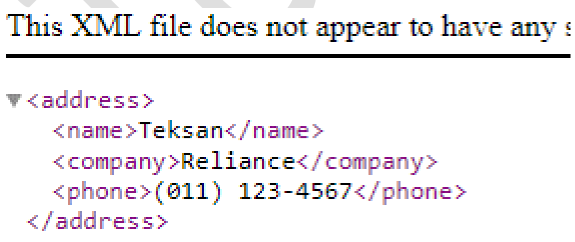
<!DOCTYPE root-element [element-declarations]>

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

Example

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

Output:



```
This XML file does not appear to have any s
▼<address>
   <name>Teksan</name>
   <company>Reliance</company>
   <phone>(011) 123-4567</phone>
 </address>
```

Code Explanation:

Start Declaration:
Begin the XML declaration with the following statement.

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

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

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

DTD Body:
The DOCTYPE declaration is followed by DTD Body, where you declare elements, attributes, entities, and notations.

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

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

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

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

External DTD

This type of DTD is declared outside the XML file with a separate file. External DTD is used in multiple XML documents, the updation done in this file affects all the XML document.

In external DTD the 'standalone' keyword is set to "no". The external content is specified using a keyword 'PUBLIC' and 'SYSTEM'. The public keyword is used outside the XML document followed by a URL (specifies the path).

Note: Multiple DTDs are allowed in which both external and internal DTDs are combined.

<!DOCTYPE root-name SYSTEM " XML file-name">

There are two types of External DTD: Private and public.

Example :1 – External DTD

Here the DTD file is created external and saved as stck.dtd and the corresponding element name is declared in the separate XML file.

stck.dtd

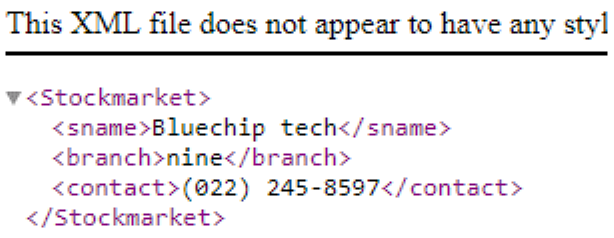
```
<?xml version="1.0"?>
<!ELEMENT Stockmarket (sname,branch,contact)>
<!ELEMENT sname (#PCDATA)>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT contact (#PCDATA)>
```

XML file

vision.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE Stockmarket SYSTEM "stck.dtd">
<Stockmarket>
<sname>Bluechip tech</sname>
<branch>nine</branch>
<contact>(022) 245-8597</contact>
</Stockmarket>
```

Output:



```
This XML file does not appear to have any styl

▼<Stockmarket>
   <sname>Bluechip tech</sname>
   <branch>nine</branch>
   <contact>(022) 245-8597</contact>
 </Stockmarket>
```

XML Schema Definition (XSD)

An XML schema, commonly known as an XML Schema Definition (XSD), is the current standard schema language for all XML documents and data. It is used to describe and validate the structure and the content of XML data. It is a framework document that defines the rules and constraints for XML documents.

An XSD defines the structure of an XML document. It specifies the elements and attributes that can appear in an XML document and the type of data these elements and attributes can contain. This information is used to verify that each element or attribute in an XML document adheres to its description.

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

<u>Advantages of XSD over DTD</u>

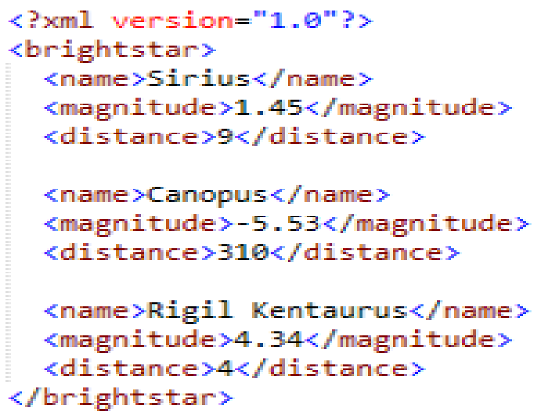
Following are the advantages of XSD over Document Type Definition (DTD):

1. XSD is extensible. You can derive new elements from the existing elements. DTD is not extensible.
2. XSD is defined in XML. It does not require intermediate processing by a parser. DTD is not defined in XML. You need separate parsers for DTD and XML.
3. XSD supports data types. You can restrict the content of an element. DTD does not support data types. Therefore, you cannot restrict the content of an element.
4. XSD supports default values. You can specify default values of the elements. You cannot specify the default values of elements in DTD.
5. XSD supports references to external XML schemas. You can include or import more than one XML schema within an XML schema. You cannot include other DTDs within a DTD.

Example:

XML Without XSD

The following is a simple XML file that contains some information about the three brightest stars at night.



```
<?xml version="1.0"?>
<brightstar>
  <name>Sirius</name>
  <magnitude>1.45</magnitude>
  <distance>9</distance>

  <name>Canopus</name>
  <magnitude>-5.53</magnitude>
  <distance>310</distance>

  <name>Rigil Kentaurus</name>
  <magnitude>4.34</magnitude>
  <distance>4</distance>
</brightstar>
```

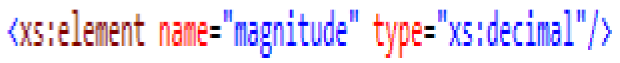
As you can see, we want to share the name, magnitude, and distance of these stars. However, each of these pieces of information is rather special:

1. The first one is text
2. The second one is a decimal
3. The third one is an integer

So, it's much better to make sure over our data.

XML with XSD

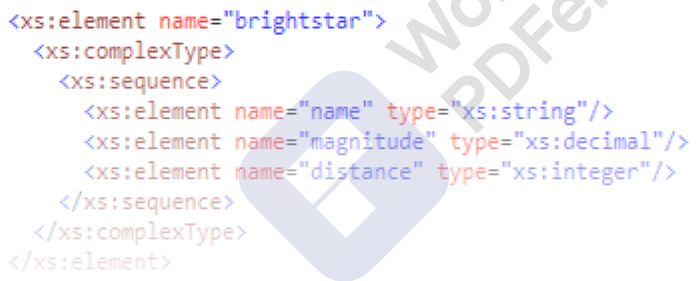
XSD files are made up of tags, just like an XML file. In fact, XSD files actually are XML files. Like an XML file, an XSD file has elements, and each element must have a name and a type. There are simple, complex, and custom (user-defined) types. A simple element looks like this:



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

As you can see, all we need is a name for the element (to distinguish it) and a type to indicate what kind of element we can expect. Now when a user accesses the 'magnitude' piece of data, it will always be checked to make sure it's a decimal (not a string or an integer). There are many predefined types, such as string, integer, and decimal.

We could just define our "brightstar" schema with all simple elements, since all we need are predefined types. Here is an example:



```
<xs:element name="brightstar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="magnitude" type="xs:decimal"/>
      <xs:element name="distance" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD Elements and Attributes

An XML schema defines elements and their structures. It also defines the attributes and their data types. The elements' structures can be of simpleType or complexType, depending on whether the element is a leaf element or a parent element.

XSD Elements

XSD elements can be of type simpleType, complexType, or anyType. An element of type simpleType contains only text. It cannot have attributes and elements. An element of type complexType can contain text, elements, and attributes. An element of type complexType is parent to all the elements and attributes contained within it. An any element in an XSD specifies that any well-formed XML is allowed in its place in XML instance.

Elements can be local or global. An element declared as the direct child of an XML schema is called a global element and can be used throughout the schema. Elements declared within a complex type definition are local elements. So, you cannot use these elements anywhere else in the schema.

XSD Attributes

XSD attributes are always of type simpleType. The syntax for defining an attribute is:

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

In this example, myattribute is an attribute of data type string. An attribute can have a fixed or default value.

An attribute group defines an association between a name and a set of attribute declarations. You can reuse the attribute groups in complexType definitions. The syntax for defining an attribute group is:

```
<xsd:attributeGroup name="Address">
<xsd:attribute name="Street1" type="xs:string"/>
<xsd:attribute name="Street2" type="xs:string"/>
<xsd:attribute name="City" type="xs:string"/>
<xsd:attribute name="State" type="xs:string"/>
</xsd:attributeGroup>
```

In this example, Address is the name of the attribute group that contains attributes such as Street1, Street2, City, and State. InterConnect resolves the attribute group references by copying the attributes definition to the referencing complexType.

anyAttribute

Like the any element, anyAttribute specifies that any attribute can be used in its place in the XML instance. InterConnect stores anyAttribute as an attribute with its name in the format _anyAttribute_. The type of this attribute is stored as String.

Nillable

Sometimes, you need to represent unknown information, or inapplicable information, explicitly with an element. For example, you might need to represent a null value being sent to or from a relational database with an element. In such cases, you can declare the nillable attribute of an element as true in its XML schema definition. InterConnect stores the nillable attribute value in the repository while importing the XSD.

XSD Type Definitions

XSD Type definitions are used to create new simpleType data type or complexType data type.

simpleType

XML schema has several built-in data types. A simpleType type is derived from an XML schema built-in data type. For example, to create a new simpleType type called myInt that has range of values between 10000 and 99999, you can base the definition of myInt on the built-in data type Integer.

Each XML schema built-in data type is mapped to an InterConnect data type. When you import an XSD that has an element of type simpleType, InterConnect identifies to which InterConnect data type this element should be mapped. InterConnect also stores the original XSD data type of an element.

For example, if you import an XSD that contains an element of XSD data type dateTime, then it will be stored as InterConnect data type Date. In addition, InterConnect also stores the dateTime data type, which is the XSD data type of the element, and its format in the repository. This pattern is used at run time for converting the Date object to date strings.

Table shows how various XSD data types are mapped to InterConnect data types in InterConnect.

Table : Mapping Between XSD Data Types and InterConnect Data Types

XSD Data Type	**InterConnect Data Type**
xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:positiveInteger	Integer
xsd:double, xsd:decimal	Double
xsd:float	Float
xsd:string, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:ID, xsd:IDREF, xsd:ENTITY, xsd:anyURI, xsd:QName, xsd:NOTATION, xsd:NMTOKENS, xsd:IDREFS, xsd:ENTITIES, xsd:duration, xsd:time, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth	String
xsd:binary and xsd:base64binary	Binary
xsd:date, xsd:datetime	Date
xsd:boolean	Boolean

complexType

A complexType type definition contains a set of element declarations, element references, and attribute declarations.You first define a complexType type in schema and then, you define the elements and attributes of this complexType type.

InterConnect stores a complexType type as an object. A complexType type can be either named or anonymous. When an imported XML schema has anonymous complexType type, then InterConnect generates its object name based on the name of the element. The type of this object is anonymous complexType. For example, when you import an XSD that contains an element MyElement of type anonymous complexType, InterConnect generates and stores the object for the element with a typename such as MyElement_CT.

anyType

All the complexTypes and simpleTypes data types are derived from anyType. So, anyType is the base type of all simpleTypes and complexTypes data types. You can create an element of type anyType by not specifying its data type in the definition. You cannot restrict the content of an anyType element. When you import an XSD that contains an element of type anyType, then InterConnect stores this anyType as String. An element of type anyType can be mapped only to an element of type anyType while doing transformations.

anySimpleType

All the simpleTypes data types of XML schema are derived from anySimpleType. So, anySimpleType is the base type of all simpleTypes data types. While importing an XSD, if an attribute does not have its type defined, then InterConnect treats the attribute type as anySimpleType. At run time, an anySimpleType element is treated as String. An anySimpleType element is mapped only to an element or attribute of type anySimpleType during transformation.

Example

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

XML Style Sheets (XSLT)

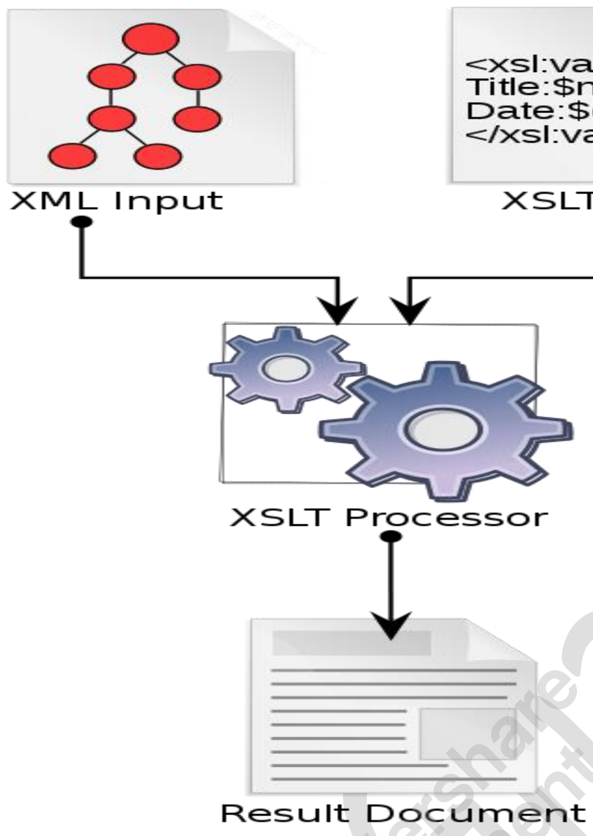
Extensible Stylesheet Language Transformations (XSLT) is a language for transforming Extensible Markup Language (XML) documents into other structured documents such as XML, HTML, plain text etc, as similar to the way that CSS files are used to transform HTML documents. This is done by using a style sheet defining template rules for transforming a given input XML document into an appropriate output document with the help of an XSL processor.

XSLT transformations can take place either at the client or server side. The XSLT processing model consists of one or more source XML documents, one or more XSL style sheets, an XSL processor and one or more structured output documents.

How XSLT Works

An XSLT document is a well-formed XML document that contains template rules. Each template rule has a pattern and a template. An XSLT processor is a software that applies a given XSLT stylesheet to a given XML document and builds the output document.

An XSLT processor works as follows. It visits the XML document tree using a pre-order traversal and compares each encountered tree node to the patterns of the template rules in the stylesheet. If a match is found, the processor writes the template of the matched rule into the output document. When all nodes have been visited, 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.



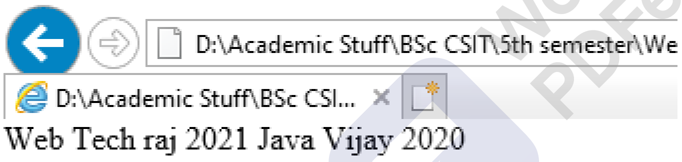
Example:
Library.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/XSL" href="xslstylesheet.xsl" ?>
<library>
  <book>
    <title>Web Tech</title>
    <author>raj</author>
    <yearpublished>2021</yearpublished>
  </book>
  <book>
    <title>Java</title>
    <author>Vijay</author>
    <yearpublished>2020</yearpublished>
  </book>
</library>
```

xslstylesheet.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/library">
  <html>
  <body>
  <h2>Book Collection</h2>
  <table border="3" align= "center">
   <tr>
     <th>Title</th>
     <th>Author</th>
     <th>Year</th>
   </tr>
   <xsl:for-each select="library/book">
   <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="author"/></td>
     <td><xsl:value-of select=" yearpublished "/></td>
   </tr>
   </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Output:



```
Web Tech raj 2021 Java Vijay 2020
```

D:\Academic Stuff\BSc CSIT\5th semester\We
D:\Academic Stuff\BSc CSI...

XQuery

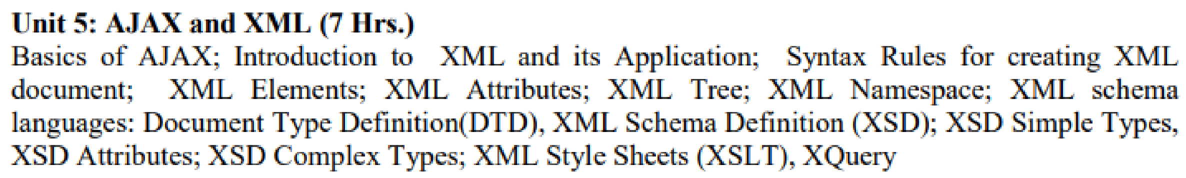
XQuery is a functional query language used to retrieve information stored in XML format. It is same as for XML what SQL is for databases. It was designed to query XML data. XQuery is built on XPath expressions. It is a W3C recommendation which is supported by all major databases.

XQuery was devised primarily as a query language for data stored in XML form. So its main role is to get information out of XML databases — this includes relational databases that store XML data, or that present an XML view of the data they hold.

Some people are also using XQuery for manipulating free-standing XML documents, for example, for transforming messages passing between applications. In that role XQuery competes directly with XSLT, and which language you choose is largely a matter of personal preference.

BSc CsIt SEM - V
Web technology

====================== Unit- 5 End=============================



Unit 5: AJAX and XML (7 Hrs.)
Basics of AJAX; Introduction to XML and its Application; Syntax Rules for creating XML document; XML Elements; XML Attributes; XML Tree; XML Namespace; XML schema languages: Document Type Definition(DTD), XML Schema Definition (XSD); XSD Simple Types, XSD Attributes; XSD Complex Types; XML Style Sheets (XSLT), XQuery

---

Unit 5: AJAX and XML (7 Hrs.) Basics of AJAX; Introduction to XML and its Application; Syntax Rules for creating XML document; XML Elements; XML Attributes; XML Tree;

XML Namespace; XML schema languages: Document Type Definition (DTD), XML Schema Definition (XSD); XSD Simple Types, XSD Attributes; XSD Complex Types;