Unit – 2

XML AND AJAX

XML-Documents and Vocabularies-Versions and Declaration –Namespaces JavaScript and XML: Ajax-DOM based XML processing Event-Transforming XML Documents-Selecting XML Data: XPATH-Template based Transformations: XSLT-Displaying XML Documents in Browsers -Evolution of AJAX -Web applications with AJAX -AJAX Framework.

What is XML

- XML stands for E**X**tensible **M**arkup **L**anguage, was designed to **describe data**.
- XML allows the user to define own tags instead of using the predefined tags, which is known as self – describing.
- XML uses a DTD (**Document Type Definition**) to formally describe the data, a meta-markup language.

Difference between XML and HTML

- XML was designed to **describe data** and to focus on **what data is, where as H**TML was designed to **display data** and to focus on **how data looks**.

- HTML is about **displaying** information, XML is about **describing** information.

- The author of HTML documents can only use tags that are defined **in the HTML standard,** XML allows the author to **define his own tags** and his own document structure.

- XML will be used to structure and describe the Web data, while HTML will be used to format and display the same data.

- XML is used for all **data transmission and data manipulation** over the Web.

XML Vs HTML

XML (Extensible Markup Language)	HTML (Hypertext Markup Language)
It stores and transports data.	It displays data.
It uses user-defined tags.	It uses predefined tags.
It contains structural data.	It doesn't contain any structural data.
It can distinguish uppercase and lowercase letters (case sensitive).	It can't distinguish uppercase and lowercase letters (case insensitive).
It maintains spacing, tabs, newlines, and any other whitespace formatting.	It doesn't maintain whitespace.
It needs to have an end-tag.	It doesn't need an end-tag.
It needs structure or nesting.	It doesn't need structure.

Disadvantages in XML

- Represents redundant and similar syntax for binary data
- Does not support all data types
- Requires a processing application
- Requires XML specific browsers

Example

- This is a note to John from Sam, stored as XML:
- **<note>**
  <to>John</to>
  <from>Sam</from>
  <heading>Reminder</heading>
  <body>Conference to Attend</body>
  </note>
- The XML above is quite self-descriptive:
  - It has sender information.
  - It has receiver information
  - It has a heading
  - It has a message body.

XML - Documents

- An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package.
- An XML *document* can contains wide variety of data.
- For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document structure

- An XML document consists of three parts, in the order given:
  - An XML declaration (which is technically optional, but recommended in most normal cases)
  - A document type declaration that refers to a DTD (which is optional, but required if you want validation)
  - A body or document instance (which is required)
- Collectively, the XML declaration and the document type declaration are called the XML prolog.

XML Declaration

- The XML declaration is a piece of markup that identifies this as an XML document.

<?xml version="1.1" ?>

- DTD is a set of rules that describes the structure of an XML document.
- XML is case-sensitive, so it's important to use lowercase for xml and version.
- If present it must be the *very first line* of the document and must not have leading white space.

XML Declaration

- The following declaration means that there is an external DTD on which this document depends.

  <?xml version="1.0" standalone="no" ?>

- If your XML document has no associated DTD, the correct XML declaration is:

  <?xml version="1.0" standalone="yes" ?>

employee.xml

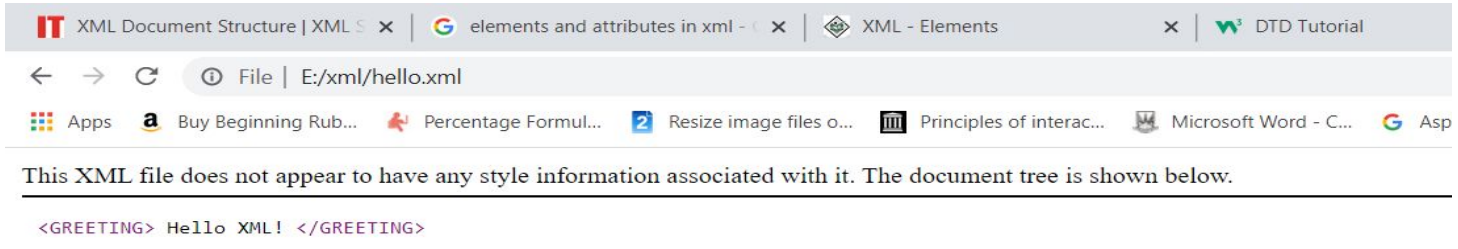
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
  <employee>
  <firstname>Maha</firstname>
  <lastname>Lakshmi</lastname>
  <email>Mahalakshmi.j@christuniversity.in</email>
</employee>
```

cssemployee.css

```
employee { display: block }
firstname { display: block; font-size: 16pt; font-weight: bold }
lastname { display: block; margin-bottom: 10px }
email { display: block; margin-bottom: 10px }
```

Example (without CSS)

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING>
```



Document Body

- The document body, or instance, is the bulk of the information content of the document.
- Across multiple instances of a document of a given type the XML prolog will remain constant, the document body changes with each document instance.
- This is because the prolog defines (either directly or indirectly) the overall structure while the body contains the real instance-specific data.
- Because the document type declaration specifies the root element, this must be the first element the parser encounters.
- If any other element but the one identified by the DOCTYPE line appears first, the document is immediately invalid.

Well-formed XML Document

- An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD).
- Validation is dealt in two ways by the XML parser. They are:
  - Well-formed XML document
  - Valid XML document
- An XML document is said to be **well-formed** if it adheres to the following rules :
  - It must begin with the XML declaration
  - It must have one unique root element
  - Start-tags must have matching end-tags
  - Elements are case sensitive
  - All elements must be closed
  - All elements must be properly nested
  - All attribute values must be quoted
  - Entities must be used for special characters

Comments in XML

- **<!-- This is a comment -->**
- <!-- This is -- an invalid comment -->

(Two dashes in the middle of a comment are not allowed)

- White-space is Preserved in XML

XML Elements

An element can contain:
– text
– attributes
– other elements
– or a mix of the above
• EMPTY Element
<element> </element> (or) <element />

Empty elements can have attributes.

XML Naming Rules

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces
- Any name can be used, no words are reserved (except xml).

XML Attributes

- Attribute values must always be quoted. Either single or double quotes can be used. Eg: <title lang="English">
- Attributes cannot contain multiple values (elements can)
- Attributes cannot contain tree structures (elements can)
- Attributes are not easily expandable (for future changes)

XML Schema Definition (XSD)

- XML Schema Definition or XSD is a recommendation by the World Wide Web Consortium to describe and validate the structure and content of an XML document.
- It is primarily used to define the elements, attributes and data types the document can contain.
- The information in the XSD is used to verify if each element, attribute or data type in the document matches its description.
- An XSD is similar to Document Type Definition (DTD), but it is a more powerful alternative as it provides greater control over the XML structure.

XML Schema Definition (XSD)

- An XML schema represents the relationships between the attributes and elements of an XML object.
- XSD is similar to a database schema that describes the data within a database.
- It defines the building blocks of an XML document, including:
  - its attributes and elements;
  - the number and order of child elements;
  - the corresponding data types of multiple elements and attributes; and
  - the default and fixed values for elements and attributes.

XSD elements

- An element is the building block of an XML document
  - Simple
  - Complex
  - Global
- **Attribute**
  - Attribute is itself a type and is used in Complex Element.
- **Restriction**
  - Restriction defines the acceptable values of an XML element.

- The simpleType element contains only text and cannot have attributes.
- It can not contain any other element.
- Examples include:
  - xs:integer
  - xs:Boolean
  - xs:string
  - xs:date

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example (XSD)

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

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
```

```
<xs:element name="item" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
         <xs:element name="title" type="xs:string"/>
         <xs:element name="note" type="xs:string" minOccurs="0"/>
         <xs:element name="quantity" type="xs:positiveInteger"/>
         <xs:element name="price" type="xs:decimal"/>
       </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="orderid" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```

XML Document

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

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Example

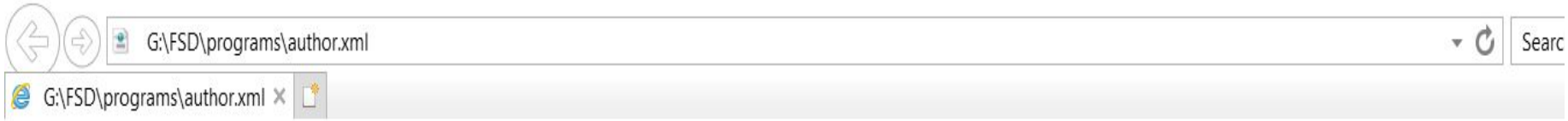
```
<?xml version="1.0"?>
<book xmlns="http://www.w3schools.com"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
"
        xsi:schemaLocation="author.xsd">
<authorname>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</authorname>
<bookname>Programming in C</bookname>
</book>
```

author.xml

author.xsd

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

Output




```
<?xml version="1.0"?>
- <book xsi:schemaLocation="author.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.w3schools.com">
  - <authorname>
      <firstname>John</firstname>
      <lastname>Smith</lastname>
    </authorname>
    <bookname>Programming in C</bookname>
</book>
```

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.
- For Eg: This XML carries HTML table information and another XML carries information about a table (a piece of furniture):

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

```
<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.

XML Namespaces - The xmlns Attribute

- A prefix can be added to avoid the conflict.
- When using prefixes in XML, a **namespace** for the prefix must be defined.
- The namespace can be defined by an **xmlns** attribute in the start tag of an element.
- The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".
- For example: 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.

The xmlns Attribute

```
<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>
```

XML Namespaces…

Namespaces can also be declared in the XML root element:

```
<root xmlns:h="http://www.abc.com/TR/html4/"
xmlns:f="http://www.xyz.com/furniture">
<h:table>
  <h:tr>
    <h:td>Aries</h:td>
    <h:td>Bingo</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>
</root>
```

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

XSL Style Sheet Documents

- An XSLT accepts as input a tree represented as an XML document and produces output.
- XSLT can be used to transform XML format to any other text format.
- The root element for an XSLT file is the <xsl:stylesheet> element.
- XSLT uses the xsl namespace, which is specified via:

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

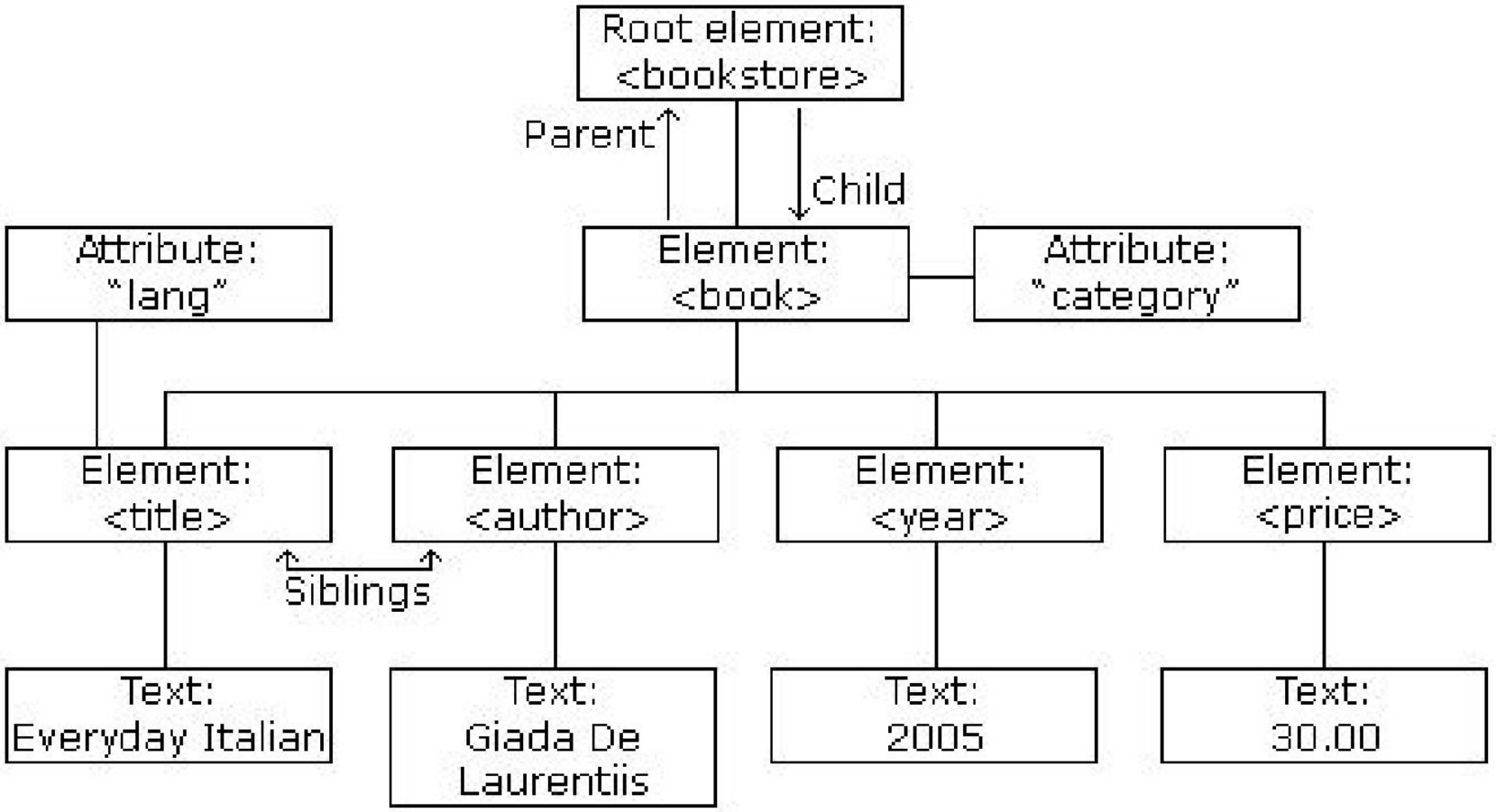
XML DOM

- Every XML DOM contains the information in hierarchical units called *Nodes* and the DOM describes these nodes and the relationship between them.
- The most common types of nodes in XML are −
  - **Document Node** − Complete XML document structure is a *document node*.
  - **Element Node** − Every XML element is an *element node*.
  - **Attribute Node** − Each attribute is considered an *attribute node*. It is not actually considered to be children of the element.
  - **Text Node** − The document texts are considered as *text node*.

XML for bookstore

```
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

XML Tree for bookstore



The Components of XSLT

- There are a number of common XSLT elements that specify how the XML data should be transformed:
  - <xsl:template>
  - <xsl:for-each>
  - <xsl:value-of>

Examining <xsl:template>

- An XSLT document can contain an arbitrary number of <xsl:template> elements.

- Each <xsl:template> element must include a **match** attribute, whose value is an XPath expression that indicates what XML nodes the <xsl:template> tag will work with.

<xsl:apply-templates> Element

- The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes.
- If we add a "select" attribute to the <xsl:apply-templates> element, it will process only the child elements that matches the value of the attribute.
- We can use the "select" attribute to specify in which order the child nodes are to be processed.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
<xsl:template match="cd">
  <p>
    <xsl:apply-templates select="title"/>
    <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>
<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
</xsl:stylesheet>
```

My CD Collection
Title: Empire Burlesque
Artist: Bob Dylan

Title: Hide your heart
Artist: Bonnie Tyler

<xsl:sort> Element

To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file

```
<xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
</xsl:for-each>
```

<xsl:if> Element

The <xsl:if> element is used to put a conditional test against the content of the XML file.

```
<table border="1">
    <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
    <th>Price</th>
  </tr>
  <xsl:for-each select="catalog/cd">
  <xsl:if test="price>10">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
      <td><xsl:value-of select="price"/></td>
    </tr>
  </xsl:if>
  </xsl:for-each>
</table>
```

<xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

```
<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
    <xsl:when test="price > 10">
      <td bgcolor="#ff00ff">
      <xsl:value-of select="artist"/>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td><xsl:value-of select="artist"/></td>
    </xsl:otherwise>
    </xsl:choose>
  </tr>
</xsl:for-each>
```

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="books1.xsl"?>
 <books
xmlns:xsi="http://www.w3.org/2001/XMLSchem
a-                              instance"
        xsi:schemaLocation="books1.xsd">

  <book>
        <title>Python</title>
        <price>107</price>
        <publisher>Addison-Wesley</publisher>
        <year>2004</year>
  </book>
  <book>
        <title>C++</title>
        <price>116</price>
        <publisher>Academic Press</publisher>
        <year>2002</year>
  </book>
```

books1.xml

```
<book>
        <title>Java</title>
        <price>120</price>
        <publisher>Academic Press</publisher>
        <year>2001</year>
   </book>
  <book>
        <title>XML</title>
        <price>126</price>
        <publisher>A K Peters</publisher>
        <year>1999</year>
  </book>
</books>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="UTF-8"/>
<xsl:template match="/">
<html>
<head><title>Books</title>
</head>
<body>
<table width="50%" border="1">
    <TR>
      <TD width="10%"
bgcolor="red"><B>Title</B></TD>
      <TD width="15%"
bgcolor="red"><B>price</B></TD>
      <TD width="10%"
bgcolor="red"><B>Publisher</B></TD>
      <TD width="10%"
bgcolor="red"><B>Year</B></TD>
    </TR>
```

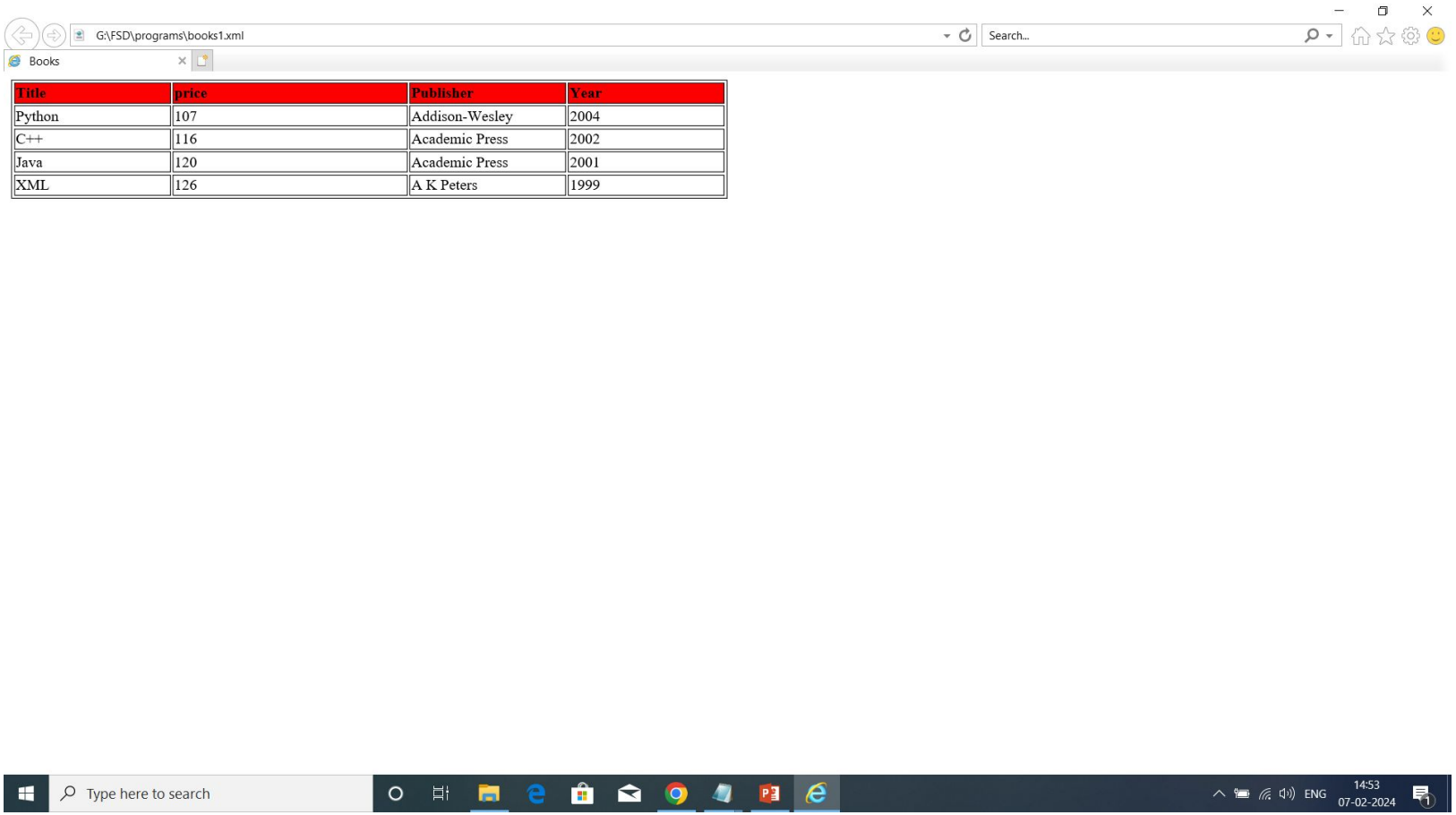
books1.xsl

```
 <xsl:for-each select="books/book">
       <TR>
          <TD width="10%"><xsl:value-of select="title"
                                                    /><
/TD>
          <TD width="15%"><xsl:value-of select="price"
                                                    /><
/TD>
          <TD width="10%"><xsl:value-of select="publisher"
                                                    /><
/TD>
          <TD width="10%"><xsl:value-of select="year"
                                                    /><
/TD>
        </TR>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

books1.xsd

```
<xs:element name="books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="book">
      <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="price" type="xs:string"/>
       <xs:element name="publisher" type="xs:string"/>
        <xs:element name="year" type="xs:string"/>
      </xs:sequence>
      </xs:complexType>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Output



Title	price	Publisher	Year
Python	107	Addison-Wesley	2004
C++	116	Academic Press	2002
Java	120	Academic Press	2001
XML	126	A K Peters	1999

Why use the DTD?

- XML provides an application independent way of sharing data
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data
- Your application can use a standard DTD to verify that data that you receive from the outside world is valid
- You can also use a DTD to verify your own data

Example
BookStore.dtd

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title,author,genre?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (first-name, last-name)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ATTLIST book price CDATA #REQUIRED>
<!ATTLIST book publicationdate CDATA>
<!ATTLIST book IBSN CDATA>
```

Internal DTD

- Using an internal DTD, the code is placed between the DOCTYPE tags
  (eg, <!DOCTYPE tutorials [and]>

- It's internal because the DTD is included in the target XML document.

Example Internal DTD

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE tutorials [
<!ELEMENT tutorials (tutorial)+>
<!ELEMENT tutorial (name,url)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ATTLIST tutorials type CDATA #REQUIRED>
]>
<tutorials type="web">
  <tutorial>
    <name>XML Tutorial</name>
    <url>https://www.quackit.com/xml/tutorial</url>
  </tutorial>
  <tutorial>
    <name>HTML Tutorial</name>
    <url>https://www.quackit.com/html/tutorial</url>
  </tutorial>
</tutorials>
```

External DTD

- An external DTD is one that resides in a separate document.
- To use the external DTD, you need to link to it from your XML document by providing the URI of the DTD file.
- This URI is typically in the form of a URL.
- The URL can point to a local file or a remote one

External DTD

- External DTD are better because of:
  - possibility of sharing definitions between XML documents
  - The documents that share the same DTD are more uniform and easier to retrieve

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM “note.dtd">
<note>
    <to>Ken Anderson</to>
    <from>Lukasz Kurgan</from>
    <text>Ok! We can see some progress</text>
</note>
```

Declaring Elements in DTD

An element declaration has the following syntax:

<!ELEMENT element-name (element-content)>

- Elements name cannot include < > characters and must start with letter or underscore
- Elements name can include namespace declaration: Namespace:element-name

Cardinality '?', '*', '+'

- The number of times a child XML Element appears within the XML Document can be controlled using the cardinality operators.

Operator	Cardinality	Description
	1-1	By default a child element is required (it must appear once within the XML document).
?	0-1	Optional operator, the element does not have to appear within the XML Document
*	0-n	Zero to Many operator, the element can appear 0 or more times
+	1-n	One to Many operator, the element can appear 1 or more times.

Built-in entities

- All XML parsers supports built-in entities.
- There are five built-in entities that play their role in well-formed XML, they are
  - ampersand: &
  - Single quote: '
  - Greater than: >
  - Less than: <
  - Double quote: "

“Bibliography” Example

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE MLBibliographies SYSTEM "default.dtd">
<MLBibliographies>
      <PageTitle>Machine Learning Bibliographies</PageTitle>
      <PageSubTitle>Maintained by Lukasz Kurgan</PageSubTitle>
      <Category href="FeatureSelection.xml"> 1. Feature Selection</Category>
      <Category href="RuleInduction.xml"> 2. Rule Induction</Category>
      <Category href=""> 3. Discretization</Category>
      <Category href=""> 4. Learning Ensemble of Classifiers</Category>
      <LastUpdate> 09/11/01</LastUpdate>
</MLBibliographies>
```

DTD document

```
<!ELEMENT MLBibliographies (PageTitle, PageSubTitle, Category*, LastUpdate, Publication*)>
<!ELEMENT PageTitle (#PCDATA)>
<!ELEMENT PageSubTitle (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT LastUpdate (#PCDATA)>
<!ATTLIST Category href CDATA #IMPLIED>
```

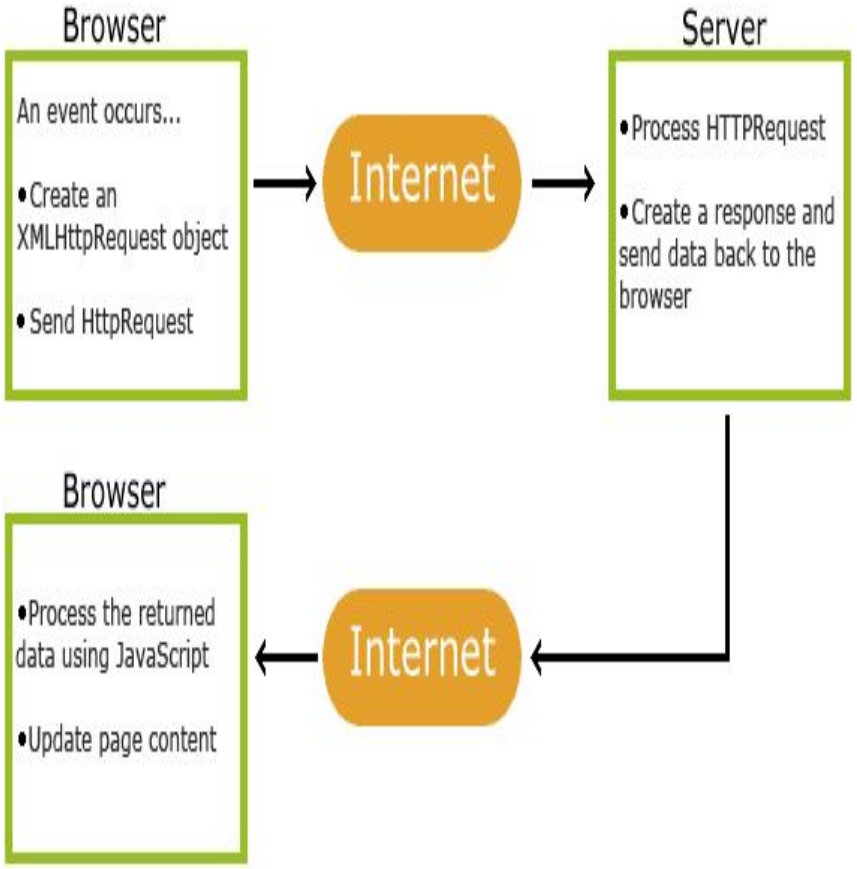
AJAX

- AJAX is an acronym for **Asynchronous JavaScript and XML**.
- It is a group of inter-related technologies like JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest etc.

- AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.
- AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

- AJAX is not a programming language.
- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)

Working of Ajax



1. An event occurs in a web page (the page is loaded, a button is clicked)

2.An XMLHttpRequest object is created by JavaScript

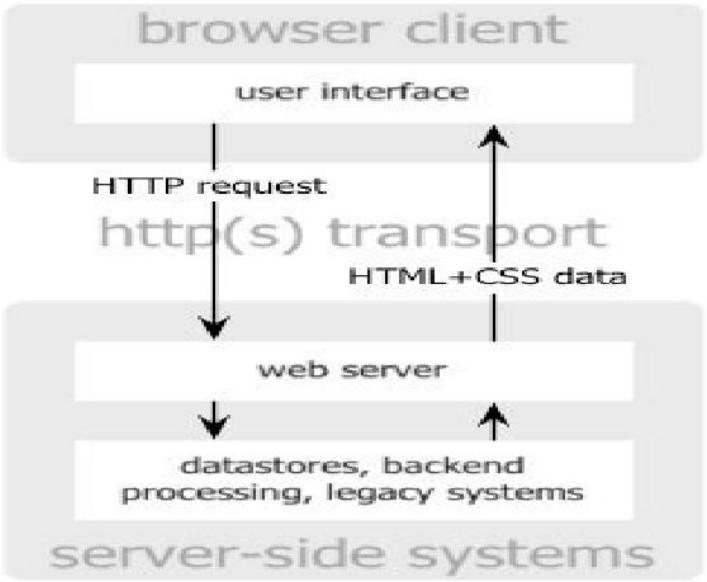
3. The XMLHttpRequest object sends a request to a web server

4. The server processes the request

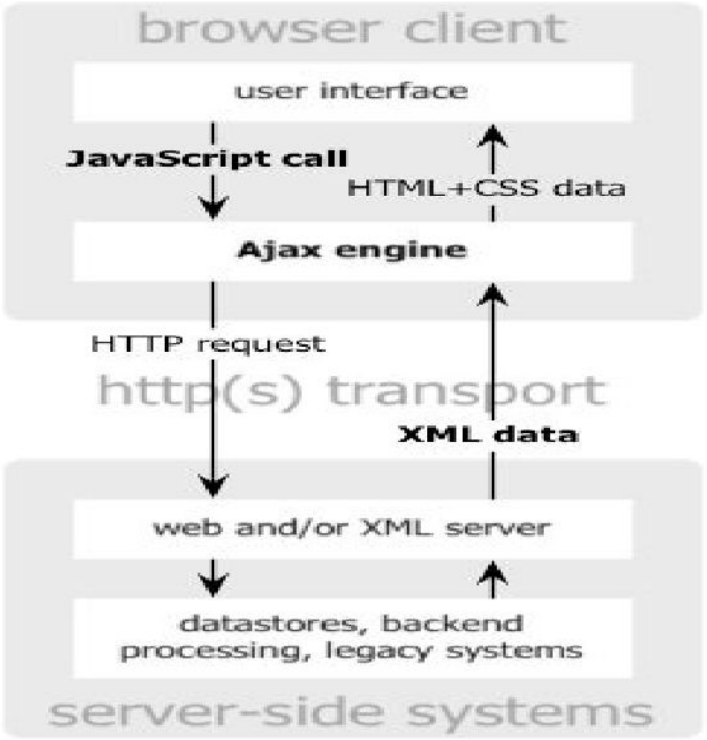
5. The server sends a response back to the web page

6. The response is read by JavaScript

7. Proper action (like page update) is performed by JavaScript



classic
web application model



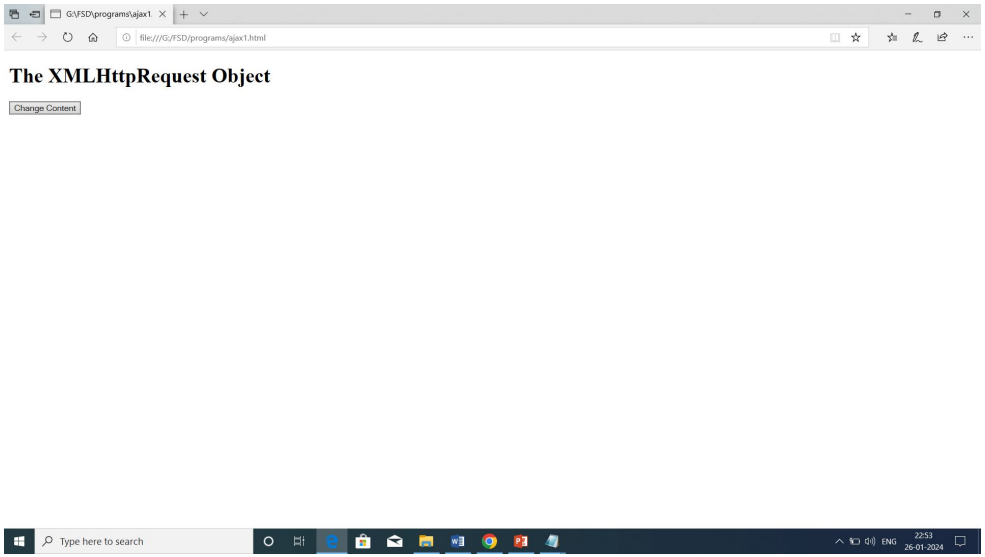
Ajax
web application model

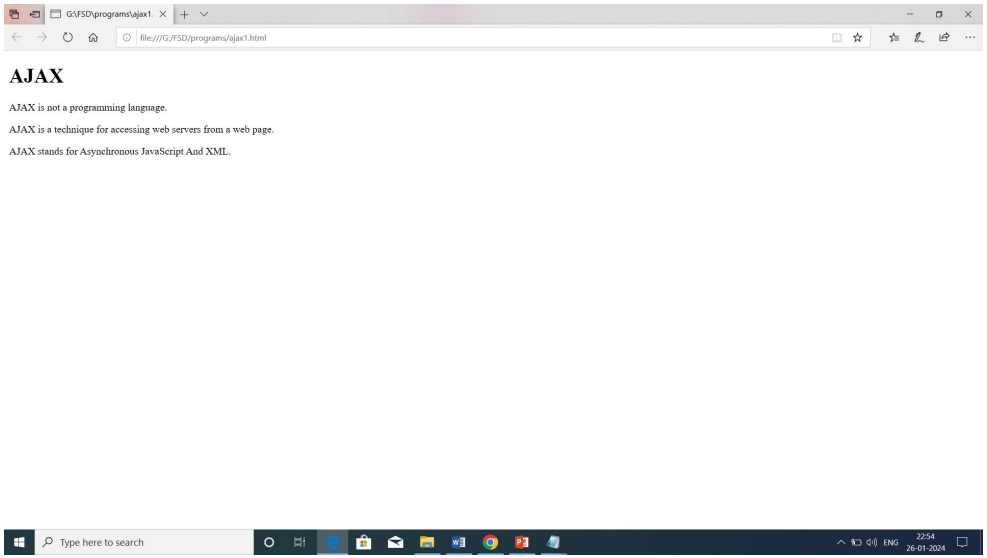
The XMLHttpRequest Object

- Base object for AJAX
  - Used to make connections, send data, receive data, etc.
- Allows your javascript code to talk back and forth with the server all it wants to, without the user really knowing what is going on.
- Available in most browsers
  - But called different things

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers
from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And
XML.</p>
```


G:\FSD\programs\ajax1.
file:///G:/FSD/programs/ajax1.html
The XMLHttpRequest Object
Change Content
Type here to search
ENG
22:53
26-01-2024


G:\FSD\programs\ajax1.
file:///G:/FSD/programs/ajax1.html
AJAX
AJAX is not a programming language.
AJAX is a technique for accessing web servers from a web page.
AJAX stands for Asynchronous JavaScript And XML.
Type here to search
ENG
22:54
26-01-2024

HTTP Ready States

- 0: request not initialized
  - Before calling open()
- 1: server connection established
  - Before calling send()
- 2: request received
  - Sometimes you can get content headers now
- 3: processing request
  - The server hasn't finished with its response
- 4: request finished and response is ready

Status

- 200: "OK“
- 403: "Forbidden“
- 404: "Page not found"

The XMLHttpRequest Object

- Methods
  - abort()
    - cancel current request
  - getAllResponseHeaders()
    - Returns the complete set of http headers as a string
  - getResponseHeader("headername")
    - Return the value of the specified header
  - open("method", "URL", async, "uname", "passwd")
    - Sets up the call
  - setRequestHeader("label", "value")
  - send(content)
    - Actually sends the request

The XMLHttpRequest Object

- Properties
  - onreadystatechange
    - Event handler for an event that fires at every state change
  - readyState
    - Returns the state of the object
  - responseText
    - Returns the response as a string
  - responseXML
    - Returns the response as XML - use W3C DOM methods
  - status
    - Returns the status as a number - ie. 404 for “Not Found”
  - statusText
    - Returns the status as a string - ie. “Not Found”

XPath

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery

XPath Path Expressions

- XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

- XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath - Expression

- An XPath expression generally defines a pattern in order to select a set of nodes. These patterns are used by XSLT to perform transformations or by XPointer for addressing purpose.
- XPath specification specifies seven types of nodes which can be the output of execution of the XPath expression.
  - Root
  - Element
  - Text
  - Attribute
  - Comment
  - Processing Instruction
  - Namespace

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

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

XPath Example

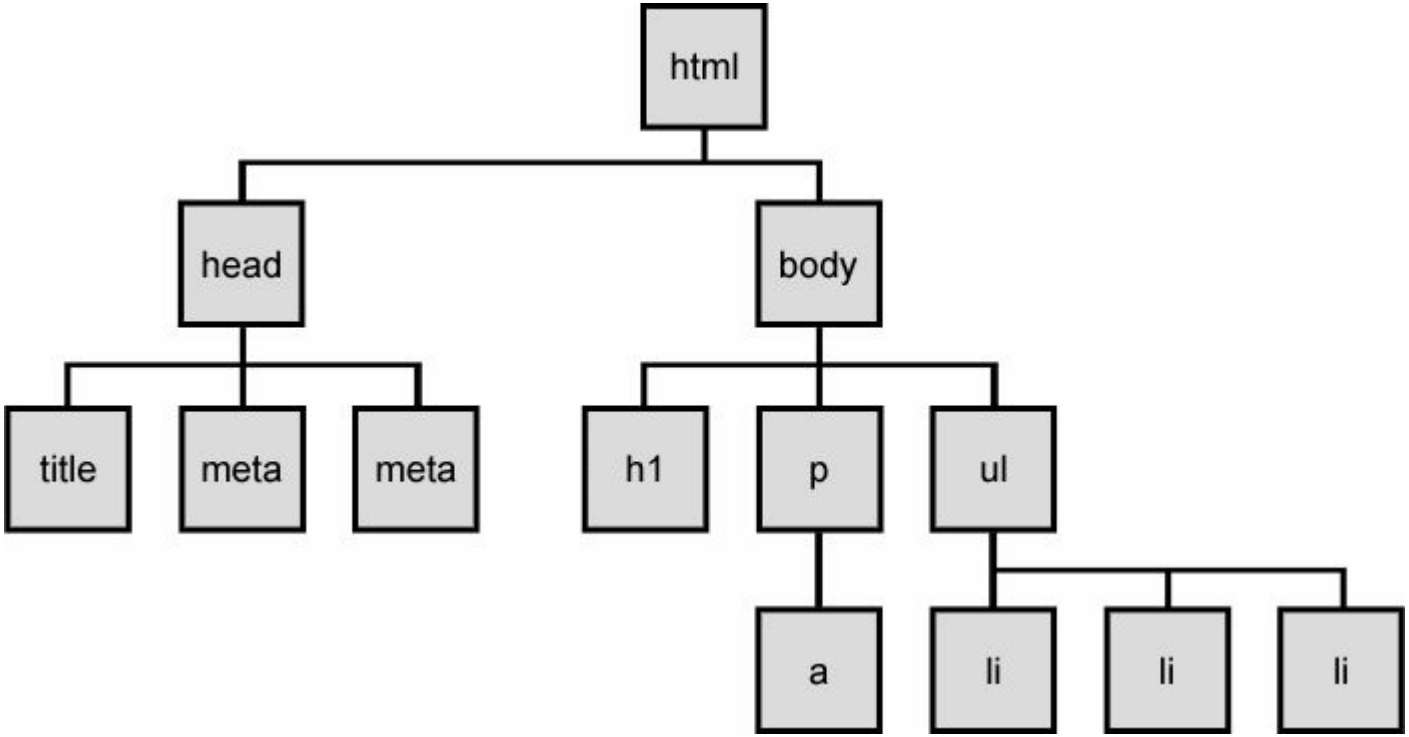
```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

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

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

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

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

</bookstore>
```

jQuery

- jQuery is a small, fast and lightweight JavaScript library.
- jQuery is platform-independent.
- jQuery means "write less do more".
- jQuery simplifies AJAX call and DOM manipulation.

The DOM tree (jQuery)




THANK
YOU