

XML

XML means extensible Markup Language, XML is a markup language which is used for storing and transporting data. XML doesn't depend on the platform and the software. You can write a program in any language on any platform to send, receive or store data using XML.

The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.

XML Properties

1. XML is a markup language that focuses on data rather than how the data looks.
2. XML is designed to send, store, receive and display data. In simple words you can say that XML is used for storing and transporting data.
3. XML became a W3C (W3C stands for World Wide Web Consortium, the main international standards organization for the World Wide Web) recommendation on February 10, 1998.
4. XML is different from HTML. XML focuses on data while HTML focuses on how the data looks.
5. XML does not depend on software and hardware, it is platform and programming language independent.
6. Unlike HTML where most of the tags are predefined, XML doesn't have predefined tags, rather you have to create your own tags.

Why we need XML?

Since there are systems with different-different operating systems having data in different formats. In order to transfer the data between these systems is a difficult task as the data needs to be converted in compatible formats before it can be used on other system. With XML, it is so easy to transfer data between such systems as XML doesn't depend on platform and the language.

XML is a simple document with the data, which can be used to store and transfer data between any systems irrespective of their hardware and software compatibilities.

Features of XML

1. XML focuses on data rather than how it looks
2. Easy and efficient data sharing
3. Compatibility with other markup language HTML

4. Supports platform transition
5. Allows XML validation
6. XML supports Unicode

Advantages of XML

1. XML is platform independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.
2. XML supports unicode. Unicode is an international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs. This feature allows XML to transmit any information written in any human language.
3. The data stored and transported using XML can be changed at any point of time without affecting the data presentation. Generally other markup language such as HTML is used for data presentation, HTML gets the data from XML and display it on the GUI (graphical user interface), once data is updated in XML, it does reflect in HTML without making any change in HTML GUI.
4. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.
5. XML simplifies data sharing between various systems because of its platform independent nature. XML data doesn't require any conversion when transferred between different systems.

Differences between HTML and XML

SNo.	HTML(Hyper Text Markup Language)	XML(eXtensible Markup Language)
1	HTML focuses on how the data looks	XML focuses on the data rather than how it looks
2	HTML is not a case sensitive language	XML is case sensitive language
3	HTML is mainly concerned with the presentation of data	XML is mainly used for storing and transporting the data
4	HTML is static	XML is dynamic

5	In HTML the closing tag is optional	In XML the closing tag is mandatory
6	HTML uses predefined tags such as , , etc.	XML uses the user-defined tags that we create while writing the XML document.
7	HTML does not preserve white space.	XML preserves white space.

XML Structure:

XML documents are formed as element trees.

An XML tree starts at a root element and branches from the root to child elements.

All elements can have sub elements

All elements can have text content and attributes.

XML documents uses a self-describing and simple syntax

Syntax:

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

An XML document is basically divided into two sections

1. Document prolog
2. Document Elements

1. Document prolog comes at the top of the document, before the root element this section contain two parts

- a. XML Declaration
- b. Document Type Defination

- a. XML Declaration:

XML declaration contains details that prepare an XML processor to parse the XML document. It is optional, but when used, it must appear in the first line of the XML document.

Syntax:

```
<?xml
```

```
version = "version_number"  
encoding = "encoding_declaration"  
standalone = "standalone_status"  
  
?>
```

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote.

Version – it specifies the version of the XML standard used. Always the value of the version is “**1.0**”.

Encoding -- It defines the character encoding used in the document. UTF-8 is the default encoding used. The values are UTF-8, UTF-16 , Etc...

Standalone -- It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to “**no**”.

Syntax Rules for XML Declaration

The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.

If document contains XML declaration, then it strictly needs to be the first statement of the XML document.

The XML declaration strictly needs be the first statement in the XML document.

An HTTP protocol can override the value of encoding that you put in the XML declaration.

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. The names of XML-elements are enclosed in triangular brackets < >

Example:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no"?>
```

2. Document Elements:

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose.

it can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Syntax:

```
<element-name attribute1 attribute2>
    ....content
</element-name>
```

here,

element-name is the name of the element. The name its case in the start and end tags must match.

attribute1, attribute2 are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters.

XML Elements Rules

- An element name can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements
- Attributes are part of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <name>Web Programming</name>
    <author>Chris Bates</author>
    <language>English</language>
    <genre>Web Applications</genre>
</book>
```

Here the first line is XML Declaration this is also called XML Prolog. It is optional, however when we include it in the XML document, it should always be the first line of the document.

The tag <book> is the root of this XML document. A XML document should always have a root element and at the end of the document this root element needs a closing tag, just like </book>

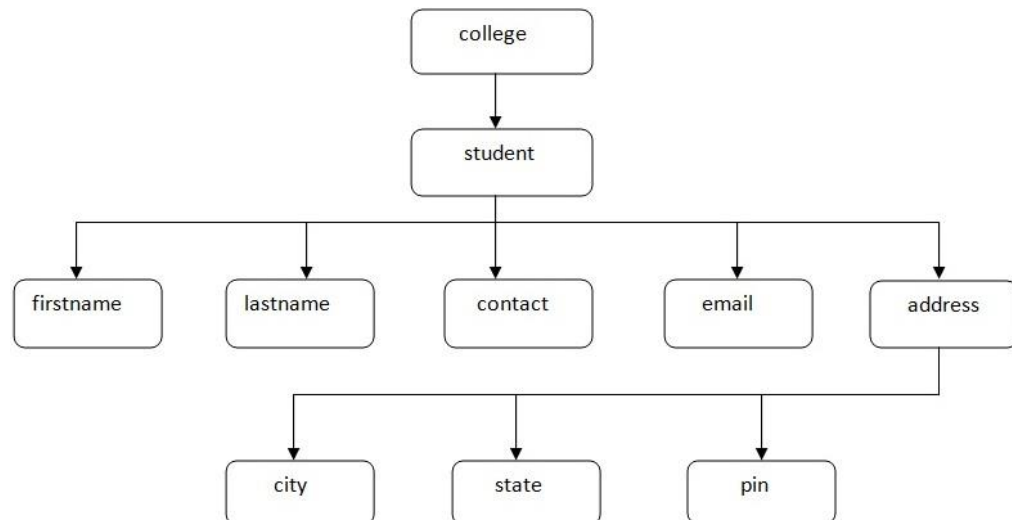
The tags <name>, <author>, <language> and <genre> are the child elements of the root element <book>.

- An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.
- A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

Example:

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

The Tree Structure of the above example is



In the above example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>.

<address> branch contains 3 sub-branches named <city>, <state> and <pin>.

An XML document is defined using user-defined tags, so if it is not valid then no data can be processed through it. The process of validate the XML document is called XML Validation. i.e XML Validator is used to validate the XML Document if it is Well Formatted XML Document or not. the Well Formatted Document is follow the following rules

Rules for well formed XML

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

A well formed XML document can be validated by using two methods those are

1. DTD
2. XML Schema

1. DTD (Document Type Definition)

The XML Document Type Declaration, commonly known as DTD. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then linked separately. Whenever DTD defined inside the XML Document is called internal DTD and defined as separate document then it is called External DTD

It can be classified into two categories

1. Internal DTD
2. External DTD

1. Internal DTD:

When a DTD is declared within the file it is called Internal DTD

Rules:

The elements that can appear in an XML document.

The order in which they can appear.

Optional and mandatory elements.

Element attributes and whether they are optional or mandatory.

Whether attributes can have default values.

To reference it as internal DTD, standalone attribute in XML declaration must be set to yes. This means the declaration works independent of external source.

Syntax

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

Here 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,branch,phone)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT branch (#PCDATA)>  
<!ELEMENT phone (#PCDATA)>  

```



```
<address>
  <name>Ramesh T</name>
  <branch>CSE</branch>
  <phone>1234567890</phone>
</address>
```

In the above example is to validate the XML document with internal DTD so in this process first we define the XML Declaration. This statement states which type of XML version and which type of character coding was used in the XML document.

After the XML declaration we define DTD declaration, it starts with DOCTYPE and it 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. The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations 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.

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

2. External DTD

It is declared in a separate file it is called External DTD.

They are accessed by specifying the system attributes which may be either the legal “.dtd” file or a valid URL.

To reference it as external DTD, standalone attribute in the XML declaration must be set as no.

Syntax:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

Here file-name is the file with .dtd extension.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Raju</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

And here is the file "note.dtd", which contains the DTD:

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

DTD declarations section you can define different elements, attributes, entity, notation rules. Well structured XML (include DTD) document follow the DTD specified rules.

Element declaration: it Specifies the name of the tag that you use to build XML document.

Attributes declaration: Specifies the attributes that you use inside element.

Entity declaration: Specifies the pieces of text that are represent as a specific entity name. XML ENTITY use to represent specific character that are difficult to write in standard keyboard. You can access defined entity in several times.

Notation declaration; Specifies the format type of non XML files like audio, image file.

DTD Element Declaration

DTD Element declaration in DTD specifies the name of the tag that use to build XML document. Every (General) XML element declare by following way,

```
<!ELEMENT element_name (inside_element)>
```

element_name specifies the general identifier and inside_element specifies what are content inside the element.

Elements with any Contents

Elements declared with the ANY keyword, Any keyword contain any combination of parse-able data.

```
<!ELEMENT element_name ANY>
```

```
<!ELEMENT div ANY>
```

EMPTY keyword specifies the empty tag. Inside no any element content.

Empty Element

```
<!ELEMENT element_name (EMPTY)>
```

```
<!ELEMENT br EMPTY>
```

EMPTY keyword specifies the empty tag. Inside no any element content.

Only Text Content Element

if your element content only text data you can declare.

```
<!ELEMENT element_name (#PCDATA)>
```

#PCDATA (parsed character data) keyword specifies parsed only character content.

Only One Child Element

If your element content only text data you can declare.

```
<!ELEMENT element_name (child_element)>
```

```
<!ELEMENT div (p)>
```

This declaration specifies <div>...</div> only have one child <p>...</p> element.

Multiple Child Element

Child elements specifies one or more separated by comma (,) sign.

```
<!ELEMENT div (p,a,span,h3)>
```

Multiple Child Element (with same name)

Child elements specifies one or more time along with same name. add + sign end of the child element name.

<!ELEMENT div (p+)>

Occur any number of child element (* sign) :

Child elements specifies any number of deep child of child element. add * (asterisk) sign end of the child element name.

<!ELEMENT div (p*)>

Optional child element(? sign) :

Child element optional if not use in XML document, you can say element_name contain empty.

<!ELEMENT element_name (child_element?)>

<!ELEMENT div (p?)>

Mixed Operator :

You can use mixed operation with together.

<!ELEMENT div (p+,a*,span?,h3)>

OR (|) Operator :

You can also specifies the choice of multiple child element using pipe sign (|) operator.

<!ELEMENT element_name (child_element | child_element | child_element)>

Suppose following specifies the child elements separated by pipe sign. You can use any of the specified child element.

<!ELEMENT div (p | a | span | h3)>

Group Elements :

You can also specifies the group of element with specific operator.

<!ELEMENT article (header,(p,a,span,)*, footer)>

<!ELEMENT article (header,(p,a,span)+, footer)>

<!ELEMENT article (header,(p,a,span)?, footer)>

<!ELEMENT article (header, (p | a | span)*, footer) >

DTD Attribute Declaration

DTD attribute declaration: If an element have attributes, you have to declare the name of the attributes in DTD.

```
<!ATTLIST      element_name      attr_name      attr_token_type
attr_declaration>
```

Description

element_name specifies the element name.

attr_name specifies the element attribute name.

attr_token_type specifies the structure/character string value.

attr_declaration specifies the default behavior of this attributes.

Attribute Declaration

Attribute declaration specifies the default behavior of the attribute.

#REQUIRED attribute must have value.

```
<!ATTLIST      element_name      attribute_name      CDATA
#REQUIRED>

<!ATTLIST employee id CDATA #REQUIRED>
```

#IMPLIED attribute value are optional. Not compulsory to have some value.

```
<!ATTLIST element_name attribute_name CDATA #IMPLIED>

<!ATTLIST      name      from      CDATA      #IMPLIED>

default_value attributes default value specifies.

<!ATTLIST element_name attribute_name CDATA "default_value">

<!ATTLIST email domain CDATA "personal">
```

#FIXED default_value attribute must have in element and also specifies the default value.

```
<!ATTLIST      element_name      attribute_name      CDATA      #FIXED      "value">

<!ATTLIST designation discipline CDATA #FIXED "Web developer">
```

2. XML Schema

Valid XML documents using DTDs. DTD has a characteristically simple syntax for functions and content definition. that DTD functions and definitions have limitations when it comes to using XML for a variety of complex purposes.

Traditionally, DTD has been the standard for XML schema definition; however, XML usage has expanded dramatically in core application systems, being tailored for a wide range of purposes for which DTD is not fully capable of supporting. Given this development, the W3C recommended "XML Schema" as a schema definition language to replace DTD. The recommendation of XML Schema has spurred its adoption as a standard schema definition language.

XML Schema Rules:

- Element definition can appear in a document
- Defined attributes can appear in the document
- Defines which element is a child, number and order also
- Defines whether the element is empty, or whether they may contain a text
- Data type definitions of elements and attributes
- Defining the elements and the default value of the property as well as a fixed value

XML Schema Advantages:

- It can be extended for future needs
- It is better and more powerful
- XML Schema-based XML authoring
- It Support Data Types, Name Spaces
- It can protect data communication
- XML Schema using the XML syntax

XML Schema Structure

<Schema> element is the root element of every XML Schema.

Syntax:

```
<?xml version="1.0"?>
<xs:schema>
    ...
    ...
</xs:schema>
```

The schema element is used as the root element, and the XML Schema "Namespace" is declared. Namespace is a specification used to avoid the duplication of attribute and element names defined under XML, and is normally designated using URL format. We use "xmlns:xs=<http://www.w3.org/2001/XMLSchema>". It is a Namespace declaration.

The "xs" designation is called the "Namespace Prefix," and can be used with an element and a child element. Generally, the "xs" prefix is used most often. The above statement written as

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ...
    ...
</xs:schema>
```

In XML schema an element belongs to two types.

1. Simple Type
2. Complex Type

1. Simple Type:

A singleType element can contain text, they do not contain other elements. When declaring an element, an ELEMENT keyword is used under DTD; however, under XML Schema, the element is used. The declaration method is different depending on whether the element has a child element or not. When no child element is present, the element name is designated with the name attribute, and the data type is designated using the type attribute.

Syntax:

```
<xs:element name="ele_name" type="Datatype"/>
```

here name refer to name of the element and type refer to datatype of element, under XML Schema, a variety of data types can be defined. Data types can be designated using pre-defined embedded simple type

Name	Explanation
xs:integer	Integers (infinite precision)
xs:float	Single-precision floating-point number (32-bit)
xs:double	Double-precision floating-point number (64-bit)

xs:string	Arbitrary text string
xs:time	Time of day
xs:dateTime	Date and time of day
xs:date	Date

2. Complex Type

Complex element contains other elements and / or attributes.

Syntax:

```
<xsd:complexType>
    Skeleton of Complex type
</xsd:complexType>
```

Example:

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

In XML Schema, we have two ways to define complex elements:

1. Name the element directly to the "employee" element declaration
2. "employee" element can use the type attribute, the role of this attribute is a reference to the name of the complex type to be used

There are four types of composite elements:

1. Empty elements

Empty the contents of the composite elements can not contain only contain attributes.

example:

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

here we define "product" no element content.

2. It contains other elements of

"Element containing only" complex type element is the only element that contains other elements.

3. Element contains only text
4. Element contains elements and text

Mixed composite type may contain attributes, elements and text.

XML Schema Indicator:

Indicators control the way how elements are to be organized in an XML document.

There are Three Types of Indicators

1. Order

Order indicators are used to define the order of the elements.

They are three types

a. all

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once

b. choice

The <choice> indicator specifies that either one child element or another can occur

c. sequence

The <sequence> indicator specifies that the child elements must appear in a specific order

2. Occurrence

Occurrence indicators are used to define how often an element can occur.

They are two types

a. maxOccurs -- It specifies the maximum number of times an element can occur

b. minOccurs -- It specifies the minimum number of times an element can occur

For all "Order" and "Group" indicators the default value for maxOccurs and minOccurs is 1.

3. Group

Group indicators are used to define related sets of elements.

They are two types

a. Group

<group> is used to group a related set of elements.

b. attributeGroup

<attributeGroup> is used to group a related set of attribute.

Example:

```
<?xml version="1.0"?>
<Employee_Info xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:noNamespaceSchemaLocation="employee.xs">
  <Employee Employee_Number="105">
    <Name>Masashi Okamura</Name>
    <Department>Design Department</Department>
    <Telephone>03-1452-4567</Telephone>
    <Email>okamura@xmltr.co.jp</Email>
  </Employee>
  <Employee Employee_Number="109">
    <Name>Aiko Tanaka</Name>
    <Department>Sales Department</Department>
    <Telephone>03-6459-98764</Telephone>
    <Email>tanaka@xmltr.co.jp</Email>
  </Employee>
</Employee_Info>
```

Xml Schema File

```
<?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
    <xs:element name="Employee_Info" type="EmployeeInfoType" />
    <xs:complexType name="EmployeeInfoType">
      <xs:sequence>
```

```

        <xs:element          ref="Employee"          minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:element name="Employee" type="EmployeeType" />
<xs:complexType name="EmployeeType">
    <xs:sequence >
        <xs:element ref="Name" />
        <xs:element ref="Department" />
        <xs:element ref="Telephone" />
        <xs:element ref="Email" />
    </xs:sequence>
    <xs:attribute          name="Employee_Number"          type="xs:int"
use="required"/>
</xs:complexType>
        <xs:element name="Name" type="xs:string" />
        <xs:element name="Department" type="xs:string" />
        <xs:element name="Telephone" type="xs:string" />
        <xs:element name="Email" type="xs:string" />
    </xs:schema>

```

Document Object Model:

DOM 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.

A DOM document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be tree based.

The XML DOM, on the other hand, also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.

DOM is a way to represent the webpage in the structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we

can easily access and manipulate tags, IDs, classes, Attributes or Elements using commands or methods provided by Document object.

Structure of DOM:

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

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed of like tag elements with attributes in HTML.

Methods of Document Object:

write("string"): writes the given string on the document.

getElementById(): returns the element having the given id value.

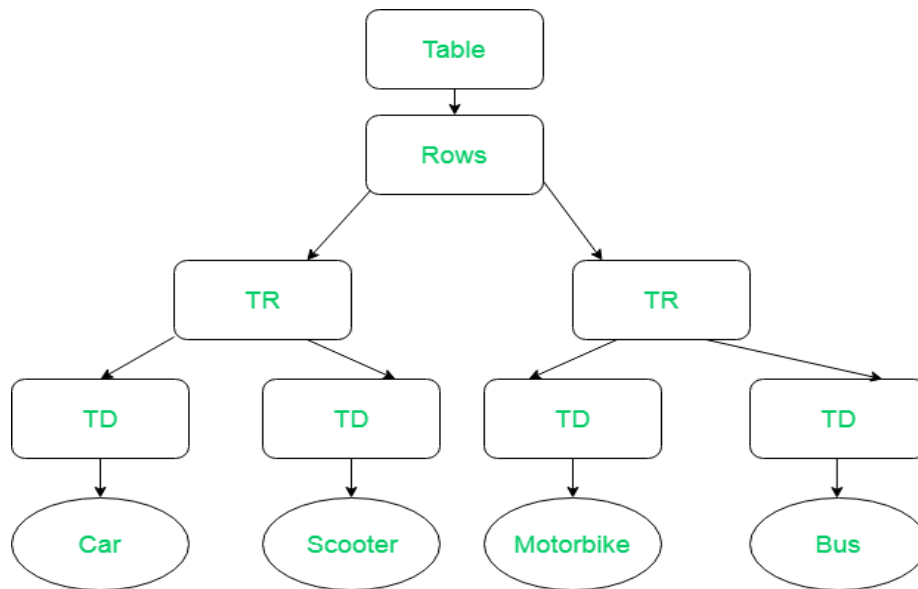
getElementsByName(): returns all the elements having the given name value.

getElementsByTagName(): returns all the elements having the given tag name.

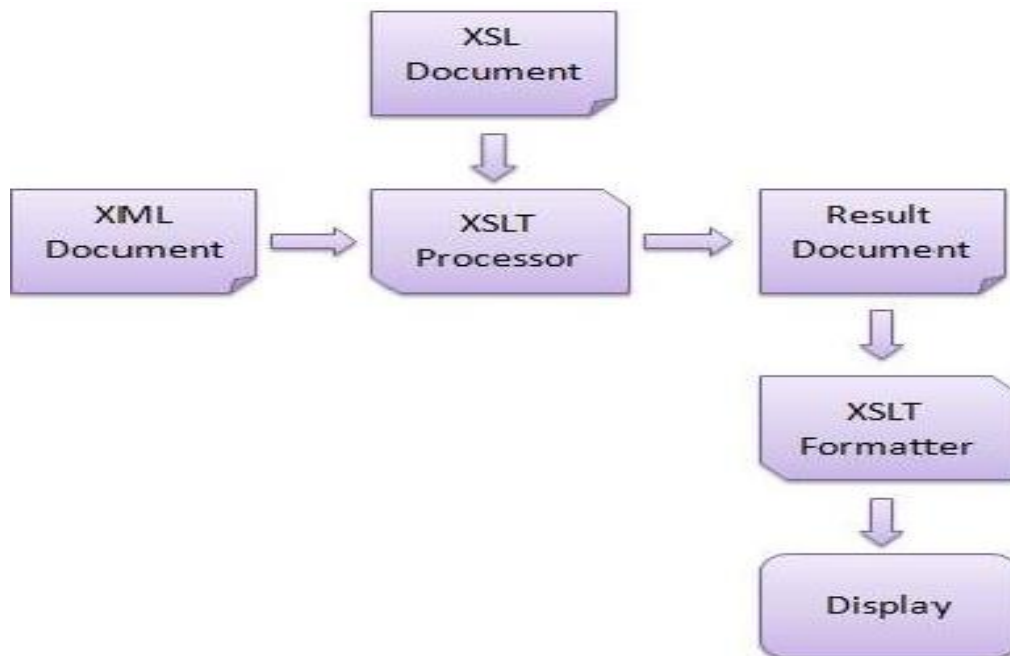
getElementsByClassName(): returns all the elements having the given class name.

Example:

```
<Table>
  <ROWS>
    <TR>
      <TD>Car</TD>
      <TD>Scooter</TD>
    </TR>
    <TR>
      <TD>MotorBike</TD>
      <TD>Bus</TD>
    </TR>
  </ROWS>
</Table>
```

**XSLT:**

XSLT stands for eXtensible Stylesheet Language Transformation. It is used to transform XML documents into other formats.



The XSLT stylesheet is written in XML format. It is used to define the transformation rules to be applied on the target XML document. The XSLT processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it

generates a formatted document in the form of XML, HTML, or text format. At the end it is used by XSLT formatter to generate the actual output and displayed on the end-user.

Advantage of XSLT

- XSLT provides an easy way to merge XML data into presentation because it applies user defined transformations to an XML document and the output can be HTML, XML, or any other structured document.
- XSLT provides Xpath to locate elements/attribute within an XML document. So it is more convenient way to traverse an XML document rather than a traditional way, by using scripting language.
- XSLT is template based. So it is more resilient to changes in documents than low level DOM and SAX.
- By using XML and XSLT, the application UI script will look clean and will be easier to maintain.
- XSLT templates are based on XPath pattern which is very powerful in terms of performance to process the XML document.
- XSLT can be used as a validation language as it uses tree-pattern-matching approach.

Example:

Let's suppose we take one sample xml file, student which is required to be transformed into a well-formatted HTML document.

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>
<class>
  <student rollno = "393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno = "493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
```

```

</student>
<student rollno = "593">
  <firstname>Jasvir</firstname>
  <lastname>Singh</lastname>
  <nickname>Jazz</nickname>
  <marks>90</marks>
</student>
</class>

```

XSLT style sheet document for the above XML document to meet the following criteria –

Page should have a title Students.

Page should have a table of student details.

Columns should have following headers: Roll No, First Name, Last Name, Nick Name, Marks

Table must contain details of the students accordingly.

Step 1: Create XSLT document

Create an XSLT document to meet the above requirements, name it as students.xsl and save it in the same location where students.xml lies.

```

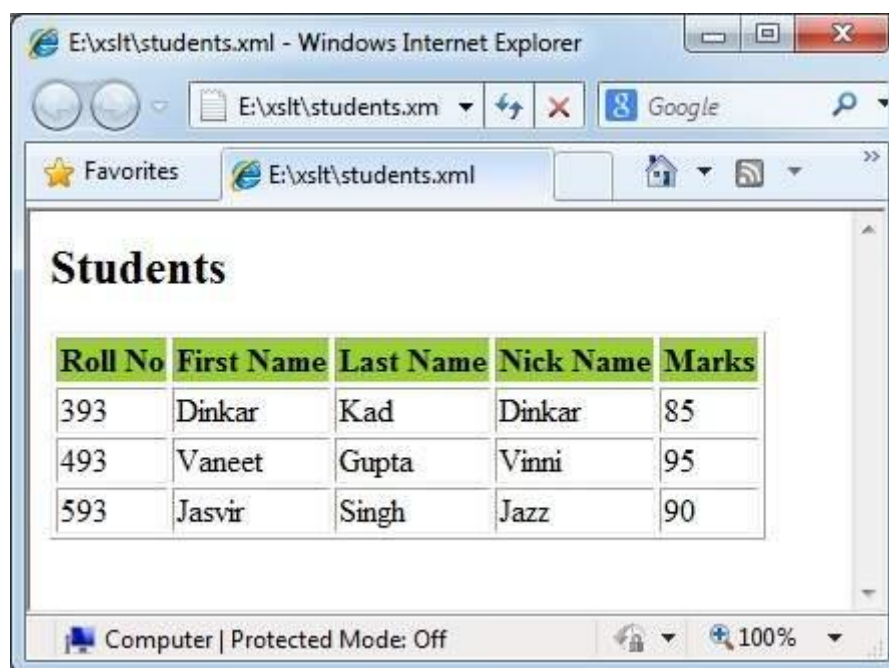
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0" xmlns:xsl =
"http://www.w3.org/1999/XSL/Transform">
  <xsl:template match = "/">
    <html>
      <body>
        <h2>Students</h2>
        <table border = "1">
          <tr bgcolor = "#9acd32">
            <th>Roll No</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Nick Name</th>
            <th>Marks</th>
          </tr>
          <xsl:for-each select="class/student">
            <tr>

```

```

        <td>
            <xsl:value-of select = "@rollno"/>
        </td>
        <td><xsl:value-of select = "firstname"/></td>
        <td><xsl:value-of select = "lastname"/></td>
        <td><xsl:value-of select = "nickname"/></td>
        <td><xsl:value-of select = "marks"/></td>
    </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Output:


Roll No	First Name	Last Name	Nick Name	Marks
393	Dinkar	Kad	Dinkar	85
493	Vaneet	Gupta	Vinni	95
593	Jasvir	Singh	Jazz	90

In above example we are using some XSLT elements

- <xsl:value-of> Element** -- The XSLT <xsl:value-of> element is used to extract the value of selected node. It puts the value of selected node as per XPath expression, as text.

Syntax:

```
<xsl:value-of    select = Expression    disable-output-escaping = "yes" | "no">
```


</xsl:value-of>

- 2. <xsl:for-each> Element** -- The XSLT <xsl:for-each> element is used to apply a template repeatedly for each node.

Syntax:

```
<xsl:for-each
  select = Expression>
</xsl:for-each>
```

- 3. <xsl:sort> Element** -- The XSLT <xsl:sort> element is used to specify a sort criteria on the nodes. It displays the output in sorted form.

The <xsl:sort> element is added inside the <xsl:for-each> element in the XSL file, to sort the output.

Syntax:

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" | QName }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } >
</xsl:sort>
```

- 4. <xsl:if> Element** -- The XSLT <xsl:if> element is used to specify a conditional test against the content of the XML file.

Syntax:

```
<xsl:if test="expression">
  ...some output if the expression is true...
</xsl:if>
```

- 5. <xsl:choose> Element** -- The XSLT <xsl:choose> element is used to specify a multiple conditional test against the content of nodes with the <xsl:otherwise> and <xsl:when> elements.

Syntax:

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
```

... some output

</xsl:otherwise>

</xsl:choose>

6. **<xsl:key> Element** -- The XSLT element is used to specify a named name-value pair assigned to a specific element in an XML document. This key is used with the key() function in XPath expressions to access the assigned elements in an XML document.

Syntax:

<xsl:key

name = QName

match = Pattern

use = Expression>

</xsl:key>

7. **<xsl:message> Element** -- The XSLT <xsl:message> element is used to display the error message and help to debug the XSLT processing. It is similar to JavaScript alerts. This element buffers a message to XSLT processor which terminates the processing and sends a message to the caller application to show an error message.

Syntax:

<xsl:message

terminate = "yes" | "no">

</xsl:message>

8. **<xsl:apply-template> Element** -- The XSLT <xsl:apply-template> element is used to tell XSLT processor to find the appropriate template to apply according to the type and context of each selected node.

Syntax:

<xsl:apply-template

select = Expression

mode = QName>

</xsl:apply-template>

9. **<xsl:import> Element** -- The XSLT <xsl:import> element is used to import the content of one stylesheet to another stylesheet. The importing stylesheet has higher precedence over imported stylesheet.

Syntax:

<xsl:import href = "uri">

</xsl:import>

XML Parsers

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

XML parser validates the document and check that the document is well formatted.

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

1. DOM
2. SAX

1. DOM:

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

Features of DOM Parser

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

DOM Parser has a tree based structure.

Advantages

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

Disadvantages

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

2. SAX:

SAX (Simple API for XML) A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.

Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.

It is an event based parser, it works like an event handler in Java.

Advantages

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

Disadvantages

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

Difference between DOM & SAX

DOM	SAX
Tree model parser (Object based) (Tree of nodes).	Event based parser (Sequence of events).
DOM loads the file into the memory and then parse- the file.	SAX parses the file as it reads it, i.e. parses node by node.
Has memory constraints since it loads the whole XML file before parsing.	No memory constraints as it does not store the XML content in the memory.
DOM is read and write (can insert or delete nodes).	SAX is read only i.e. can't insert or delete the node.
If the XML content is small, then prefer DOM parser.	Use SAX parser when memory content is large.
Backward and forward search is possible for searching the tags and evaluation of the information inside the tags. So this gives the ease of navigation.	SAX reads the XML file from top to bottom and backward navigation is not possible.
Slower at run time.	Faster at run time.

AJAX is about updating parts of a web page, without reloading the whole page.

What is AJAX?

Ajax is an acronym for Asynchronous Javascript and XML. It is used to communicate with the server without refreshing the web page and thus increasing the user experience and better performance.

AJAX is a web development technique for creating interactive web applications. If you know JavaScript, HTML, CSS, and XML, then you need to spend just one hour to start with AJAX.

AJAX = Asynchronous JavaScript and XML.

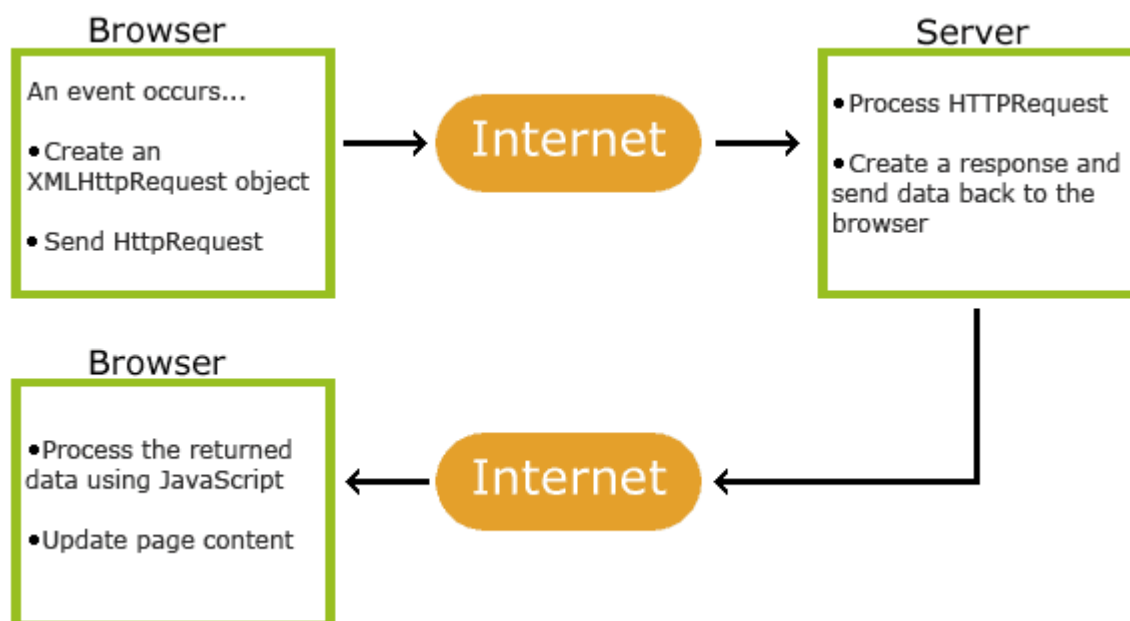
AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works



How does it work?

First, let us understand what does asynchronous actually mean. There are two types of requests synchronous as well as asynchronous. Synchronous requests are the one which follows sequentially i.e if one process is going on and in the same time another process wants to be executed, it will not be allowed that means the only one process at a time will be executed. This is not good because in this type most of the time CPU remains idle such as during I/O operation in the process which are the order of magnitude slower than the CPU processing the instructions. Thus to make the full utilization of the CPU and other resources use asynchronous calls. For more information visit this [link](#). Why the word javascript is present here. Actually, the requests are made through the use of javascript functions. Now the term XML which is used to create **XMLHttpRequest object**.

Thus the summary of the above explanation is that Ajax allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. Now discuss the important part and its implementation. For implementing Ajax, only be aware of XMLHttpRequest object. Now, what actually it is. It is an object used to exchange data with the server behind the scenes. Try to remember the paradigm of OOP which says that object communicates through calling methods (or in general sense message passing). The same case applied here as well. Usually, create this object and use it to call the methods which result in effective communication. All modern browsers support the XMLHttpRequest object.

Basic Syntax: The syntax of creating the object is given below

```
req = new XMLHttpRequest();
```

There are two types of methods open() and send(). Uses of these methods explained below.

```
req.open("GET", "abc.php", true);
```

```
req.send();
```

The above two lines described the two methods. req stands for the request, it is basically a reference variable. The GET parameter is as usual one of two types of methods to send the request. Use POST as well depending upon whether send the data through POST or GET method. The second parameter being the name of the file which actually handles the requests and processes them. The third parameter is true, it tells that whether the requests are processed asynchronously or synchronously. It is by default true which means that requests are asynchronous. The open() method prepares the request before sending it to the server. The send method is used to send the request to the server.

Sending the parameter through getting or POST request. The syntax is given below

```
req.open("GET", "abc.php?x=25", true);
```

```
req.send();
```

In the above lines of code, the specified query in the form of URL followed by ?which is further followed by the name of the variable then = and then the corresponding value. If sending two or more variables use ampersand(&) sign between the two variables. The above method as shown applies for GET request. Sending the data through the POST, then send it in the send method as shown below.

```
req.send("name=johndoe&marks=99");
```

Use of setRequestHeader() method as shown below.

```
req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

AJAX is based on Internet Standards

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent!

Google Suggest

AJAX was made popular in 2005 by Google, with Google Suggest.

[Google Suggest](#) is using AJAX to create a very dynamic web interface: When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

Start Using AJAX Today

In our PHP tutorial, we will demonstrate how AJAX can update parts of a web page, without reloading the whole page. The server script will be written in PHP.

If you want to learn more about AJAX, visit our [AJAX tutorial](#).

Advantages:

1. Speed is enhanced as there is no need to reload the page again.
2. AJAX make asynchronous calls to a web server, this means client browsers avoid waiting for all the data to arrive before starting of rendering.
3. Form validation can be done successfully through it.
4. Bandwidth utilization – It saves memory when the data is fetched from the same page.
5. More interactive.

Disadvantages:

1. Ajax is dependent on Javascript. If there is some Javascript problem with the browser or in the OS, Ajax will not support.
2. Ajax can be problematic in Search engines as it uses Javascript for most of its parts.
3. Source code written in AJAX is easily human readable. There will be some security issues in Ajax.
4. Debugging is difficult.
5. Problem with browser back button when using AJAX enabled pages.

PHP Ajax with Example

We will create a simple application that allows users to search for popular PHP MVC frameworks.

Our application will have a text box that users will type in the names of the framework.

We will then use mvc AJAX to search for a match then display the framework's complete name just below the search form.

Step 1) Creating the index page

Index.php

```
<html>
<head>
<title>PHP MVC Frameworks - Search Engine</title>
<script type="text/javascript" src="/auto_complete.js"></script>
</head>
<body>
<h2>PHP MVC Frameworks - Search Engine</h2>
<p><b>Type the first letter of the PHP MVC Framework</b></p>
<form method="POST" action="index.php">
<p><input type="text" size="40" id="txtHint"
onkeyup="showName(this.value)"></p>
</form>
<p>Matches: <span id="txtName"></span></p>
</body>
</html>
```

“onkeyup=“showName(this.value)”” executes the JavaScript function showName everytime a key is typed in the textbox.

This feature is called auto complete

Step 2) Creating the frameworks page

frameworks.php

```
<?php
$frameworks = array("CodeIgniter","ZendFramework","Cake PHP","Kohana");
$name = $_GET["name"];
if (strlen($name) > 0) {
```



```

$match = "";
for ($i = 0; $i < count($frameworks); $i++) {
if (strtolower($name) == strtolower(substr($frameworks[$i], 0, strlen($name)))) {
if ($match == "") {
    $match = $frameworks[$i];
    } else {
    $match = $match . " , " . $frameworks[$i];
    }
    }
}
echo ($match == "") ? 'no match found' : $match;
?>

```

Step 3) Creating the JS script

auto_complete.js

```

functionshowName(str){
if (str.length == 0){ //exit function if nothing has been typed in the textbox
document.getElementById("txtName").innerHTML=""; //clear previous results
return;
}
if (window.XMLHttpRequest) { // code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
    } else { // code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState == 4 &&xmlhttp.status == 200){
document.getElementById("txtName").innerHTML=xmlhttp.responseText;
    }
}
xmlhttp.open("GET","frameworks.php?name="+str,true);
xmlhttp.send();
}

```

“if (str.length == 0)” check the length of the string. If it is 0, then the rest of the script is not executed.

“if (window.XMLHttpRequest)...” Internet Explorer versions 5 and 6 use ActiveXObject for AJAX implementation. Other versions and browsers such as Chrome, FireFox use XMLHttpRequest. This code will ensure that our application works in both IE 5 & 6 and other high versions of IE and browsers.

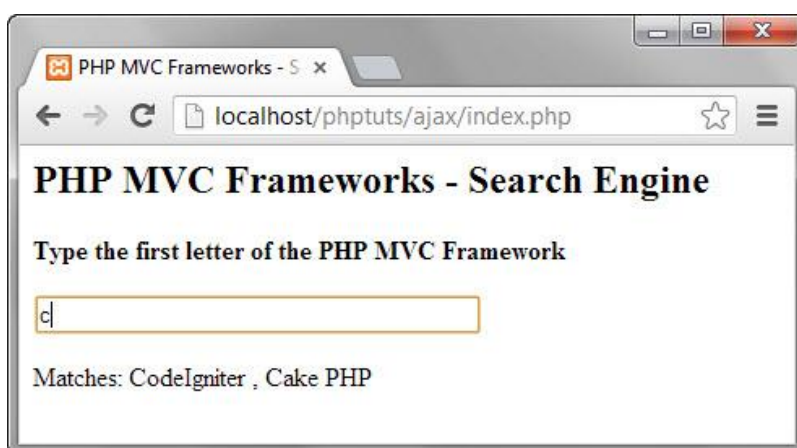
“xmlhttp.onreadystatechange=function...” checks if the AJAX interaction is complete and the status is 200 then updates the txtName span with the returned results.

Step 4) Testing our PHP Ajax application

Assuming you have saved the file index.php In phututs/ajax, browse to the URL <http://localhost/phptuts/ajax/index.php>



Type the letter C in the text box You will get the following results.



The above example demonstrates the concept of AJAX and how it can help us create rich interaction applications.