AJAX

AJAX stands for Asynchronous JavaScript and XML. AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the server using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

Rich Internet Application Technology

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

AJAX is Based on Open Standards

AJAX is based on the following open standards:

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

AJAX - Technologies

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

JavaScript

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

DOM

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

CSS

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

XMLHttpRequest

- JavaScript object that performs asynchronous interaction with the server.

AJAX – Examples: The web applications using of AJAX are :

Google Maps : A user can drag an entire map by using the mouse, rather than clicking on a button.
Ex : http://maps.google.com/

Google Suggest : As you type, Google will offer suggestions. Use the arrow keys to navigate the results.

Ex : http://www.google.com/webhp?complete=1&hl=en

Gmail : Gmail is a webmail, built on the idea that email can be more intuitive, efficient and useful.
Ex : http://gmail.com/

Yahoo Maps (new) : Now it's even easier and more fun to get where you're going!
Ex : http://maps.yahoo.com/

Difference between AJAX and Conventional Server Side Program

When we click a submit button in AJAX, then the page will not be is submitted and we get the response very quickly, but when we use server side programming language, thenwe have to wait for the response and your page also gets refreshed/submitted.

Ensuring Browser Support for AJAX :

Some browsers cannot support AJAX. The following browsers support AJAX.

- Mozilla Firefox 1.0 and above.
- Netscape version 7.1 and above.
- Apple Safari 1.2 and above.
- Microsoft Internet Explorer 5 and above.
- Opera 7.6 and above.

NOTE: When we say that a browser does not support AJAX, it simply means that the browser does not support creation of Javascript object XMLHttpRequest object.

Writing Browser Specific Code

The Simplest way to make your source code compatible with a browser is to use try...catch blocks in your JavaScript.

```
<html>
<body>
  <script language="javascript" type="text/javascript">
  <!--
  //Browser Support Code
  function ajaxFunction(){
    var ajaxRequest;  // The variable that makes Ajax possible!

    try{
      // Opera 8.0+, Firefox, Safari
      ajaxRequest = new XMLHttpRequest();
      alert("hi");
    }catch (e){

      // Internet Explorer Browsers
      try{
        ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
        alert("hey");
      }catch (e) {
        try{
          ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
          alert("hello");
        }catch (e){

          // Something went wrong
          alert("Your browser broke!");
          return false;
```

```
        }
      }
    }
  }
  //-->
  </script>

  <form name='myForm'>
    Name: <input type='text' name='username' /> <br />
    Time: <input type='text' name='time' />
     <input type="submit" value="Submit" onClick="return ajaxFunction()" />
  </form>

</body>
</html>
```

In the above JavaScript code, we try three times to make our XMLHttpRequest object. Our first attempt:

- ajaxRequest = new XMLHttpRequest();

It is for Opera 8.0+, Firefox, and Safari browsers. If it fails, we try two more times to make the correct object for an Internet Explorer browser with:

- ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
- ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");

If it doesn't work, then we may be using an outdated browser that doesn't support XMLHttpRequest, i.e. Ajax.

AJAX In Action : Different steps how AJAX Operates.

1. The client generates an event.
2. The XMLHttpRequest object is created.
3. The XMLHttpRequest object is configured.
4. The XMLHttpRequest object makes an asynchronous request to the Webserver.
5. The Webserver returns the result containing XML document.
6. The XMLHttpRequest object calls the callback() function and processes the result.
7. The HTML DOM is updated.

Step-1 : The Client generates an Event.

- A JavaScript function is called as the result of an event.
- Example: validateUserId() JavaScript function is mapped as an event handler to an onkeyup event on input form field whose id is set to "userid"
- <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId();" >

Step-2 : The XMLHttpRequest Object is Created

```
var ajaxRequest;  // The variable that makes Ajax possible!
function ajaxFunction(){
  try{
    // Opera 8.0+, Firefox, Safari
    ajaxRequest = new XMLHttpRequest();
  }catch (e){

    // Internet Explorer Browsers
    try{
      ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
    }catch (e) {
```

```
      try{
        ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
      }catch (e){

        // Something went wrong
        alert("Your browser broke!");
        return false;
      }
    }
  }
}
```

Step- 3 : The XMLHttpRequest Object is Configured
In this step, we will write a function that will be triggered by the client event and a callback function processRequest() will be registered.

```
function validateUserId() {
  ajaxFunction();

  // Here processRequest() is the callback function.
  ajaxRequest.onreadystatechange = processRequest;

  if (!target)
      target = document.getElementById("userid");
  var url = "validate?id=" + escape(target.value);

  ajaxRequest.open("GET", url, true);
  ajaxRequest.send(null);
}
```

Step-4: Making Asynchronous Request to the Webserver
Source code is available in the above piece of code. Code written in bold typeface is responsible to make a request to the webserver. This is all being done using the XMLHttpRequest objectajaxRequest.

```
function validateUserId() {
  ajaxFunction();

  // Here processRequest() is the callback function.
  ajaxRequest.onreadystatechange = processRequest;

  if (!target) target = document.getElementById("userid");
  var url = "validate?id=" + escape(target.value);

  ajaxRequest.open("GET", url, true);
  ajaxRequest.send(null);
}
```

Assume you enter Zara in the userid box, then in the above request, the URL is set to "validate?id=Zara".

Step-5: Webserver returns the result containing XML Document
You can implement your server-side script in any language, however its logic should be as follows.

- Get a request from the client.
- Parse the input from the client.
- Do required processing.
- Send the output to the client.

If we assume that you are going to write a servlet, then here is the piece of code.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
  String targetId = request.getParameter("id");

  if ((targetId != null) && !accounts.containsKey(targetId.trim())){
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("true");
  }  else{
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("false");
  }
}
```

Step-6: Callback Function processRequest() is Called
The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to thereadyState of the XMLHttpRequest object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

```
function processRequest() {
  if (req.readyState == 4) {
    if (req.status == 200) {
      var message = ...;
...
}
```

Step-7 : The HTML DOM is Updated
This is the final step and in this step, your HTML page will be updated. It happens in the following way:

- JavaScript gets a reference to any element in a page using DOM API.
- The recommended way to gain a reference to an element is to call.

```
document.getElementById("userIdMessage"),
// where "userIdMessage" is the ID attribute
// of an element appearing in the HTML document
```

- JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements. Here is an example:

```
<script type="text/javascript">
<!--
function setMessageUsingDOM(message) {
  var userMessageElement = document.getElementById("userIdMessage");
  var messageText;

  if (message == "false") {
    userMessageElement.style.color = "red";
    messageText = "Invalid User Id";
  }
  else
  {
    userMessageElement.style.color = "green";
    messageText = "Valid User Id";
```

```
  }

  var messageBody = document.createTextNode(messageText);

  // if the messageBody element has been created simple
  // replace it otherwise append the new element
  if (userMessageElement.childNodes[0]) {
    userMessageElement.replaceChild(messageBody, userMessageElement.childNodes[0]);
  }
  else
  {
    userMessageElement.appendChild(messageBody);
  }
}
-->
</script>
<body>
<div id="userIdMessage"><div>
</body>
```

AJAX - XMLHttpRequest

The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.

XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

- abort() - Cancels the current request.
- getAllResponseHeaders() - Returns the complete set of HTTP headers as a string.
- getResponseHeader(headerName)- Returns the value of the specified HTTP header.
- open(method, URL)
  open(method, URL, async)
  open(method, URL, async, userName)
  open(method, URL, async, userName, password) - Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods, such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

- send(content) - Sends the request.
- setRequestHeader(label, value) - Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties

- onreadystatechange - An event handler for an event that fires at every state change.
- readyState - The readyState property defines the current state of the XMLHttpRequest object.

The following list provides the possible values for the readyState property:

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process.
4	The request is completed.

readyState = 0 After you have created the XMLHttpRequest object, but before you have called the open() method.
readyState = 1 After you have called the open() method, but before you have called send().
readyState = 2 After you have called send().
readyState = 3 After the browser has established a communication with the server, but before the server has completed the response.
readyState = 4 After the request has been completed, and the response data has been completely received from the server.

- responseText - Returns the response as a string.
- responseXML - Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
- status - Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
- statusText - Returns the status as a string (e.g., "Not Found" or "OK").

AJAX - Database Operations
Example-1: Let us create a login application.
Store this file as **checklogin.html** under root folder :

```
<html><body>
<script language="javascript" type="text/javascript">
<!--
//Browser Support Code
function ajaxFunction1(){
 var ajaxRequest;  // The variable that makes Ajax possible!

 try{
   // Opera 8.0+, Firefox, Safari
   ajaxRequest = new XMLHttpRequest();
 }catch (e){
   // Internet Explorer Browsers
   try{
     ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
   }catch (e) {
     try{
       ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
     }catch (e){
       // Something went wrong
       alert("Your browser broke!");
       return false;
     }
   }
 }

// Create a function that will receive data sent from the server and will update
//  div section in the same page.
```

```
 ajaxRequest.onreadystatechange = function(){
   if(ajaxRequest.readyState == 4){
      var ajaxDisplay = document.getElementById('ajaxDiv');
      ajaxDisplay.innerHTML = ajaxRequest.responseText;
   }
 }
 // Now get the value from user and pass it to server script.
 var name = document.getElementById('name').value;
 var queryString = "?name=" + name;
 ajaxRequest.open("GET", "checklogin.jsp" + queryString, true);
 ajaxRequest.send(null);
}
//-->
</script>
<form name='myForm'>
Enter Login Name : <input type='text' id='name' onMouseOut='ajaxFunction1()' /><div id='ajaxDiv'>
</div>
<input type='text' id='pwd' />
<input type='button' value='Submit'/>
</form>

</body></html>
```

Store the following file as **checklogin.jsp** in the same root folder :

```
<%@ include file="dbcon.jsp" %>
<html><body>
<%
String name="", qry="";
try{
name=request.getParameter("name");
qry="select * from stud where name='"+name+"'";
ResultSet rs= st.executeQuery(qry);
String data="";
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
 // write out the response string
if(rs.next()){
   data="<font color=\"red\" size=\"3\">Name is not Available.</font>";
}else{
   data="<font color=\"green\" size=\"3\">Name is Available.</font>";
}
con.close();
response.getWriter( ).write(data);
}catch(Exception e){ out.println(e); }
%>
</body></html>
```

Example-2: Let us create a retrieval application.

Store the following file as **getdata.html** in the same root folder :

```
<html><body>
<script language="javascript" type="text/javascript">
<!--
//Browser Support Code
function ajaxFunction(){
```

```
var ajaxRequest;  // The variable that makes Ajax possible!

try{
  // Opera 8.0+, Firefox, Safari
  ajaxRequest = new XMLHttpRequest();
}catch (e){
  // Internet Explorer Browsers
  try{
    ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
  }catch (e) {
    try{
      ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
    }catch (e){
      // Something went wrong
      alert("Your browser broke!");
      return false;
    }
  }
}
// Create a function that will receive data
// sent from the server and will update
// div section in the same page.
ajaxRequest.onreadystatechange = function(){
  if(ajaxRequest.readyState == 4){
    var ajaxDisplay = document.getElementById('ajaxDiv');
    ajaxDisplay.innerHTML = ajaxRequest.responseText;
  }
}
// Now get the value from user and pass it to
// server script.
var roll = document.getElementById('roll').value;
var queryString = "?roll=" + roll;
ajaxRequest.open("GET", "getdata.jsp" +  queryString, true);
ajaxRequest.send(null);
}
//-->
</script>
<form name='myForm'>
Enter Roll : <input type='text' id='roll' />
<input type='button' onclick='ajaxFunction()' value='Submit'/>
</form>
<div id='ajaxDiv'>Your result will display here</div>
</body></html>
```

Store the following file as **getdata.jsp** in the same root folder :

```
<%@ include file="dbcon.jsp" %>
<html><body>
<%
String roll="", qry;
try{
roll=request.getParameter("roll");
}catch(Exception re){ }

try{
if(roll.equals(""))
```

```
        qry="select * from stud";
else
         qry="select * from stud where roll="+roll;

ResultSet rs= st.executeQuery(qry);
String data="";
data="<table><tr><th>Name</th><th>Roll</th><th>Sem</th><th>Mark</th></tr>";

response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
         // write out the response string
while(rs.next()){
data=data+"<tr><td>"+rs.getString("name")+"</td><td>"+   rs.getString("roll")   +"</td><td>"+
rs.getString("sem") +"</td><td>"+ rs.getString("mark") +"</td></tr>";
}
data=data+"</table>";
 con.close();
//out.print(data);
response.getWriter( ).write(data);
}catch(Exception e){ System.out.println(e); }
%>
</body></html>
```

Example-3: Let us create an insertion application.

Store the following file as **insertdata.html** in the same root folder :

```
<html>
<body bgcolor="#FFCCFF">
<script language="javascript" type="text/javascript">
<!--
//Browser Support Code
function ajaxFunction(){
 var ajaxRequest;  // The variable that makes Ajax possible!

 try{
   // Opera 8.0+, Firefox, Safari
   ajaxRequest = new XMLHttpRequest();
 }catch (e){
   // Internet Explorer Browsers
   try{
     ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
   }catch (e) {
     try{
       ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
     }catch (e){
       // Something went wrong
       alert("Your browser broke!");
       return false;
     }
   }
 }
 // Create a function that will receive data
 // sent from the server and will update
 // div section in the same page.
 ajaxRequest.onreadystatechange = function(){
   if(ajaxRequest.readyState == 4){
```

```
    var ajaxDisplay = document.getElementById('ajaxDiv');
    ajaxDisplay.innerHTML = ajaxRequest.responseText;
  }
 }
 // Now get the value from user and pass it to
 // server script.
 var name = document.getElementById('name').value;
 var roll = document.getElementById('roll').value;
 var sem = document.getElementById('sem').value;
 var mark = document.getElementById('mark').value;
 var queryString = "?name="+name+"&roll=" + roll+"&sem="+sem+"&mark="+mark;
 ajaxRequest.open("GET", "insertdata.jsp" +  queryString, true);
 ajaxRequest.send(null);
}
//-->
</script>
<form name='myForm'>
<table>
<tr><td>Enter Name </td><td> <input type='text' id='name' /> </td></tr>
<tr><td>Enter Roll </td><td> <input type='text' id='roll' /> </td></tr>
<tr><td>Enter Sem  </td><td> <input type='text' id='sem' /> </td></tr>
<tr><td>Enter Mark </td><td> <input type='text' id='mark' /> </td></tr>
<tr><td></td><td><input type='button' onclick='ajaxFunction()' value='Submit'/></td></tr>
</table>
</form>
<div id='ajaxDiv'>Your result will display here</div>
</body></html>
```

Store the following file as **insertdata.jsp** in the same root folder :

```
<%@ include file="dbcon.jsp" %>
<html><body>
<%
String name="",roll="",sem="",mark="", qry;
try{
        name=request.getParameter("name");
        roll=request.getParameter("roll");
        sem=request.getParameter("sem");
        mark=request.getParameter("mark");
        qry="insert into stud values('"+name+"','"+roll+"','"+sem+"', "+mark+")";
        int z= st.executeUpdate(qry);
        con.close();
        String data="";
        if(z>0)
          data=name+" inserted successfully";
        else
          data=name+"is not inserted";
        //out.print(data);
        response.getWriter( ).write(data);
}catch(Exception e){ response.getWriter( ).write(e.toString()); }
%>
</body></html>
```