AJAX - Quick Guide

Previous Page Next Page

What is AJAX?

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

Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.

Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.

With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.

AJAX is a web browser technology independent of web server software.

A user can continue to use the application while the client program requests information from the server in the background.

Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.

Data-driven as opposed to page-driven.

Rich Internet Application Technology

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

AJAX is Based on 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

Here is a list of some famous web applications that make use of AJAX.

Google Maps

A user can drag an entire map by using the mouse, rather than clicking on a button.

https://maps.google.com/

Google Suggest

As you type, Google offers suggestions. Use the arrow keys to navigate the results.

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

Gmail

Gmail is a webmail built on the idea that emails can be more intuitive, efficient, and useful.

https://gmail.com/

Yahoo Maps (new)

Now it's even easier and more fun to get where you're going!

https://maps.yahoo.com/

Difference between AJAX and Conventional CGI Program

Try these two examples one by one and you will feel the difference. While trying AJAX example, there is no discontinuity and you get the response very quickly, but when you try the standard GCI example, you would have to wait for the response and your page also gets refreshed.

AJAX Example

[ ] * [ ] = [ ] AJAX

Standard Example

[ ] * [ ] = [ ] Standard

NOTE – We have given a more complex example in AJAX Database .

AJAX - Browser Support

All the available browsers cannot support AJAX. Here is a list of major browsers that 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.

Konqueror.

Opera 7.6 and above.

When you write your next application, do consider the browsers that do not support AJAX.

NOTE – When we say that a browser does not support AJAX, it simply means that the browser does not support the 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();
            } 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;
                  }
               }
            }
         }
         //-->
      </script>

      <form name = 'myForm'>
         Name: <input type = 'text' name = 'username' /> <br />
         Time: <input type = 'text' name = 'time' />
      </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 can use a very outdated browser that doesn't support XMLHttpRequest, which also means it doesn't support AJAX.

Most likely though, our variable ajaxRequest will now be set to whatever *XMLHttpRequest* standard the browser uses and we can start sending data to the server. The step-wise AJAX workflow is explained in the next chapter.

AJAX - Action

This chapter gives you a clear picture of the exact steps of AJAX operation.

Steps of AJAX Operation

A client event occurs.

An XMLHttpRequest object is created.

The XMLHttpRequest object is configured.

The XMLHttpRequest object makes an asynchronous request to the Webserver.

The Webserver returns the result containing XML document.

The XMLHttpRequest object calls the callback() function and processes the result.

The HTML DOM is updated.

Let us take these steps one by one.

A Client Event Occurs

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();">.

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;
         }
      }
   }
}
```

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);
}
```

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 object *ajaxRequest*.

```
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".

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("<valid>true</valid>");
   } else {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("<valid>false</valid>");
   }
}
```

Callback Function processRequest() is Called

The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the *readyState* 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 = ...;
...
}
```

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>
```

If you have understood the above-mentioned seven steps, then you are almost done with AJAX. In the next chapter, we will see *XMLHttpRequest* object in more detail.

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.

You already have seen a couple of examples on how to create an XMLHttpRequest object.

Listed below are some of the methods and properties that you have to get familiar with.

XMLHttpRequest Methods

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 table provides a list of 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

To clearly illustrate how easy it is to access information from a database using AJAX, we are going to build MySQL queries on the fly and display the results on "ajax.html". But before we proceed, let us do the ground work. Create a table using the following command.

NOTE – We are assuming you have sufficient privilege to perform the following MySQL operations.

```
CREATE TABLE 'ajax_example' (
   'name' varchar(50) NOT NULL,
   'age' int(11) NOT NULL,
   'sex' varchar(1) NOT NULL,
   'wpm' int(11) NOT NULL,
   PRIMARY KEY  ('name')
)
```

Now dump the following data into this table using the following SQL statements –

```
INSERT INTO 'ajax_example' VALUES ('Jerry', 120, 'm', 20);
INSERT INTO 'ajax_example' VALUES ('Regis', 75, 'm', 44);
INSERT INTO 'ajax_example' VALUES ('Frank', 45, 'm', 87);
INSERT INTO 'ajax_example' VALUES ('Jill', 22, 'f', 72);
INSERT INTO 'ajax_example' VALUES ('Tracy', 27, 'f', 0);
INSERT INTO 'ajax_example' VALUES ('Julie', 35, 'f', 90);
```

Client Side HTML File

Now let us have our client side HTML file, which is ajax.html, and it will have the following code –

```
<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 age = document.getElementById('age').value;
            var wpm = document.getElementById('wpm').value;
            var sex = document.getElementById('sex').value;
            var queryString = "?age = " + age ;

            queryString +=  "&wpm = " + wpm + "&sex = " + sex;
            ajaxRequest.open("GET", "ajax-example.php" + queryString, true);
            ajaxRequest.send(null);
         }
         //-->
      </script>

      <form name = 'myForm'>
         Max Age: <input type = 'text' id = 'age' /> <br />
         Max WPM: <input type = 'text' id = 'wpm' /> <br />
         Sex:

         <select id = 'sex'>
            <option value = "m">m</option>
```

```
            <option value = "f">f</option>
         </select>

         <input type = 'button' onclick = 'ajaxFunction()' value = 'Query MySQL'/>
      </form>

      <div id = 'ajaxDiv'>Your result will display here</div>
   </body>
</html>
```

NOTE – The way of passing variables in the Query is according to HTTP standard and have formA.

```
URL?variable1 = value1;&variable2 = value2;
```

The above code will give you a screen as given below –

Max Age:

Max WPM:

Sex: m ▼

Query MySQL

Your result will display here in this section after you have made your entry.

NOTE – This is a dummy screen.

Server Side PHP File

Your client-side script is ready. Now, we have to write our server-side script, which will fetch age, wpm, and sex from the database and will send it back to the client. Put the following code into the file "ajax-example.php".

```
<?php
$dbhost = "localhost";
$dbuser = "dbusername";
$dbpass = "dbpassword";
$dbname = "dbname";

//Connect to MySQL Server
mysql_connect($dbhost, $dbuser, $dbpass);

//Select Database
mysql_select_db($dbname) or die(mysql_error());

// Retrieve data from Query String
$age = $_GET['age'];
$sex = $_GET['sex'];
$wpm = $_GET['wpm'];

// Escape User Input to help prevent SQL Injection
$age = mysql_real_escape_string($age);
$sex = mysql_real_escape_string($sex);
$wpm = mysql_real_escape_string($wpm);
```

```php
//build query
$query = "SELECT * FROM ajax_example WHERE sex = '$sex'";

if(is_numeric($age))
   $query .= " AND age <= $age";

if(is_numeric($wpm))
   $query .= " AND wpm <= $wpm";

//Execute query
$qry_result = mysql_query($query) or die(mysql_error());

//Build Result String
$display_string = "<table>";
$display_string .= "<tr>";
$display_string .= "<th>Name</th>";
$display_string .= "<th>Age</th>";
$display_string .= "<th>Sex</th>";
$display_string .= "<th>WPM</th>";
$display_string .= "</tr>";

// Insert a new row in the table for each person returned
while($row = mysql_fetch_array($qry_result)) {
   $display_string .= "<tr>";
   $display_string .= "<td>$row[name]</td>";
   $display_string .= "<td>$row[age]</td>";
   $display_string .= "<td>$row[sex]</td>";
   $display_string .= "<td>$row[wpm]</td>";
   $display_string .= "</tr>";
}

echo "Query: " . $query . "<br />";
$display_string .= "</table>";

echo $display_string;
?>
```

Now try by entering a valid value (e.g., 120) in *Max Age* or any other box and then click Query MySQL button.

Max Age:

Max WPM:

Sex: m ▼

Query MySQL

Your result will display here in this section after you have made your entry.

If you have successfully completed this lesson, then you know how to use MySQL, PHP, HTML, and Javascript in tandem to write AJAX applications.

AJAX - Security

AJAX Security: Server Side

AJAX-based Web applications use the same server-side security schemes of regular Web applications.

You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programmatic).

AJAX-based Web applications are subject to the same security threats as regular Web applications.

AJAX Security: Client Side

JavaScript code is visible to a user/hacker. Hacker can use JavaScript code for inferring server-side weaknesses.

JavaScript code is downloaded from the server and executed ("eval") at the client and can compromise the client by mal-intended code.

Downloaded JavaScript code is constrained by the sand-box security model and can be relaxed for signed JavaScript.

AJAX - Current Issues

AJAX is growing very fast and that is the reason that it contains many issues with it. We hope with the passes of time, they will be resolved and AJAX will become ideal for web applications. We are listing down a few issues that AJAX currently suffers from.

Complexity is increased

Server-side developers will need to understand that presentation logic will be required in the HTML client pages as well as in the server-side logic.

Page developers must have JavaScript technology skills.

AJAX-based applications can be difficult to debug, test, and maintain

JavaScript is hard to test - automatic testing is hard.

Weak modularity in JavaScript.

Lack of design patterns or best practice guidelines yet.

Toolkits/Frameworks are not mature yet

Most of them are in beta phase.

No standardization of the XMLHttpRequest yet

Future version of IE will address this.

No support of XMLHttpRequest in old browsers

Iframe will help.

JavaScript technology dependency and incompatibility

Must be enabled for applications to function.

Still some browser incompatibilities exist.

JavaScript code is visible to a hacker

Poorly designed JavaScript code can invite security problems.

Previous Page Next Page

Enter email for newsletter go