## 5.2XML Http Request Object and CALLBACK()

The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. **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.

**Creatingan XMLHttpRequest Object**

**Syntax:**

variable=new XMLHttpRequest(); (new version)

variable=new ActiveXObject("Microsoft.XMLHTTP"); (old version)

**Processing Requests in AJAX**

The following are the sequence of operations request is initiated:

- 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.

- **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();">.
```

- **XMLHttpRequest object is created**

```
varajaxRequest; // AJAX variable
functionajaxFunction()
{ try
{ // This code is for browsers 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**

```
functionvalidateUserId()
{ ajaxFunction(); // Here processRequest() is the callback function.
ajaxRequest.onreadystatechange = processRequest;
if (!target) target = document.getElementById("userid");
varurl = "validate?id=" + escape(target.value);
ajaxRequest.open("GET", url, true);
ajaxRequest.send(null);       }
```

- **Making an asynchronous request to the Webserver**

  This is done using the XMLHttpRequest object ajaxRequest. Assume that the user enters Sona in the userid box, then in the above request, the URL is set to "validate?id=Sona".

- **Webserver Returns the Result Containing XML Document**

  Server-side script is implemented 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 the user is going to write a servlet, then:

```
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");  } }
```

- **Callback Function processRequest() is Called**

The callback function is responsible for checking the progress of requests, identifying the result of the request and handling data returned from the server. Callback functions also serve as delegators, handing off to other areas of the application code. 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.

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

- **The HTML DOM is updated.**

This is the final step and in this step, the 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"), // userIdMessage is 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.

```
<script type="text/javascript">
<!— functionsetMessageUsingDOM(message)
{ varuserMessageElement = document.getElementById("userIdMessage");
varmessageText;
if (message == "false")
{ userMessageElement.style.color = "red";
messageText = "Invalid User Id"; }
else {
```

```
userMessageElement.style.color = "green";
messageText = "Valid User Id";    }
varmessageBody = document.createTextNode(messageText);
if (userMessageElement.childNodes[0])
{  userMessageElement.replaceChild(messageBody,  userMessageElement.childNodes[0]);
}
 Else   { userMessageElement.appendChild(messageBody);   } }
</script> <body> <div id="userIdMessage"><div> </body>
```

**XMLHttpRequest Methods**

| Method | Description |
|---|---|
| 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 )<br>open( method, URL, async )<br>open( method, URL, async, userName )<br>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" 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.

| 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 the user have created the XMLHttpRequest object, but before the call of the open() method.
- readyState = 1 After the user have called the open() method, but before the call of send().
- readyState = 2 After the user 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").