6

Web Forms–Get Wet in AJAX

In this chapter we will look at:

- JavaScript basics
- Understanding DOM
- How to place asynchronous requests to the server
- How to handle Key Events
- How form elements can be accessed through JavaScript
- Performing XMLHttp Post Requests
- How to separate JavaScript code from the web page
- How to specify our own function in .js file
- How to use CSS (cascading style sheets)
- Sending data from combobox to server asynchronously
- Sending items selected from listbox to server asynchronously
- Sending items selected from radio and checkboxes to server asynchronously
- AJAX, PHP and MySQL combined for accessing database
- Accessing database in AJAX

6.1 JAVASCRIPT BASICS

JavaScript is a lightweight, fully interpreted language which is supported by all major web browsers like Internet Explorer, Opera, Safari, Firefox etc. JavaScript code can be embedded in an HTML program by writing its statements in `<script>` element or can also be written in a separate file with

extension .js and then including that JavaScript code file in <script> element. The later method is usually preferred to avoid clutter. Statements in JavaScript are terminated by a semi colon.

The following are the characteristics of JavaScript:

- It is a scripting language which is interpreted
- The syntax of commands is easier to learn
- Can automate several tasks like validation of userid, email id etc.

In the following example we create a JavaScript file named prog1.js in `htdocs` subfolder of the directory where Apache is installed and then this JavaScript file is referenced in HTML file in order to invoke the methods defined in it. The contents of both files are as under:

prog1.html

```
1. <html>
2. <head>
3. <script type="text/JavaScript" src="prog1.js"></script>
4. </head>
5. <body onload="showdata()">
6. Welcome to our Shopping Mall <br>
7. <div id="info" />
8. </body>
9. </html>
```

prog1.js

```
1. function showdata()
2. {
3. var msg;
4. msg = "We provide wide variety of items at reasonable price";
5. document.getElementById("info").innerHTML = msg;
6. }
```

Explanation of prog1.html program

```
3. <script type="text/JavaScript" src="prog1.js"></script>
```

The above instruction imports the JavaScript file: prog1.js in the current web page.

There are two ways to include JavaScript in our web page:

1. Place JavaScript in the <head> element
2. Place JavaScript in a separate file, save it with extension .js and use <script> element to include the code file (by including the JavaScript file, its codes will be merged in the HTML at that location). This approach is preferred as it keeps HTML code clean and all the JavaScript code at one place

<script> element is usually included within the <head> section to include JavaScript code file. The type of this tag is set to "text/JavaScript" to specify the type of script to be included in the <script> tag. The `src` specifies the JavaScript file name along with the path.

For example src="/javascript/prog1.js" means to look for the prog1.js script file in JavaScript folder and import it into the current web page.

```
5. <body onload="showdata()">
```

`onload()` function used in the body tag is an event handler which means that when the page is all rendered, the browser looks for the code specified in the `onload()` event and executes it. That is, the code specified in `showdata()` method (present in prog1.js script file) will be executed.

```
7. <div id="info" />
```

 element is a block element used for specifying the location for displaying the results. Since it takes up all the available width of the browser, larger information including tables can also be displayed with this element. We can have several elements in a web page, so they are assigned separate ids.

Note: Beside element, we can also use element for showing results.

Why we use or element?

 and element have a property called innerHTML, the contents of which can be changed dynamically. The idea is that we set up a element on a page and with that page still showing, the browser sends the asynchronous request to the server and gets the result which is then placed in the innerHTML property of the element.

Explanation of prog1.js

```
5. document.getElementById("info").innerHTML = msg;
```

`document.getElementById()` method is used for searching a web page for an object with the specified id. The object placed anywhere on the page with the given id is searched for by this method. The above instruction is searching for an element on the web page with id `info` (our element) and sets the contents of its innerHTML property equal to that of string `msg` in order to display string contents at the location of the element.

The output of the program prog1.html may appear as shown in Figure 6.1.

Output:

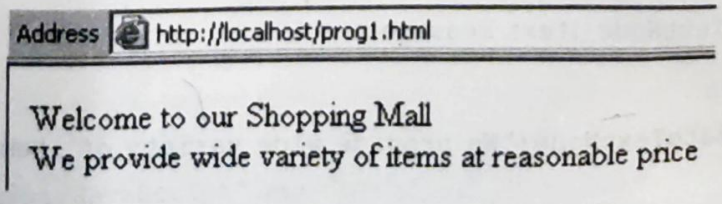

Figure 6.1 ***Output of the Program Prog1.html***

6.2 UNDERSTANDING DOM

Document Object Model is an object by which all the elements in an HTML document can be accessed by a JavaScript engine. When a HTML page is loaded and rendered by a browser, a document tree is constructed that represents the displayed document. This document tree is composed of elements or nodes which may contain child nodes within them that can be manipulated and updated programmatically using JavaScript code.

The root node of the web page is accessed by the global variable `document` and is considered as the starting point of all DOM manipulations. Every node in the DOM is a child of `document`. Since DOM nodes are arranged in the form of a tree structure, every DOM node has a parent and one or more children which can be accessed by `parentNode` and `childNodes` properties respectively. `parentNode` returns a DOM node object and `childNodes` returns a JavaScript array of nodes.

6.2.1 CreateElement()

This is the method to create new nodes to be added to the document. We can create any HTML element by this method.

Syntax:

```
document.createElement(tag type)
```

`tag type` is the argument sent to it

Example:

```
obld=document.createElement("b");
```

It creates a bold element by name obld

6.2.2 CreateTextNode()

This creates a DOM node representing the supplied text. This text is then nested inside the given tag. Text nodes are different from elements in the sense that no styles can be applied to them directly and hence they consume less memory. Styles are applied to the text node with the help of DOM elements containing it.

Syntax:

```
document.createTextNode (text message)
```

Example:

```
omsg=document.createTextNode("We provide wide variety of items at reasonable
price");
```

A DOM node named `omsg` is created with the above text.

6.2.3 AppendChild()

This method is for attaching the given node to the document. A node never appears in the browser window until and unless it is attached to the document using `appendChild` method. The child node can be attached to any element.

Syntax:

```
appendChild (child node)
```

Example:

```
obld.appendChild(omsg);
```

The text node `omsg` made in the above example is attached to the `obld` element made earlier. Now the text matter of the `omsg` node will appear in bold (because the `obld` element has the bold tag applied to it).

So we see that a new structure can be added to a document using these three methods: `createElement()`, `createTextNode()` and `appendChild()`.

6.2.4

In AJAX usually we use `<div>` element for displaying or modifying dynamic contents. In order to identify a `<div>` element uniquely, we tag it with a unique id. An id can be assigned to any DOM node which can then be used to refer that element later.

6.2.5 GetElementById()

This is a method to get the programmatic reference to the node with the specified id.

Syntax:

```
document.getElementById(ID);
```

This method refers to the element whose ID is supplied as argument.

Now let's understand the above DOM methods with an example. In the following example we create a JavaScript file named prog2.js in `htdocs` subfolder of the `apache` directory and then the JavaScript code is invoked by the HTML program prog2.html. The contents of both files are as under:

prog2.html

```
<html>
<head>
<script type="text/JavaScript" src="prog2.js"></script>
</head>
```

```
<body onload="showdata()">
Welcome to our Shopping Mall <br>
<div id="info" />
</body>
</html>
```

What this Program is Doing

This program is first importing the JavaScript file prog2.js in the current web page so that the functions defined in that file can be invoked from the current web page. The `onload()` event is fired when the page is all rendered, invoking the `showdata()` method specified in the JavaScript file. Also, a message "Welcome to our Shopping Mall" followed by a line break appears on the browser screen. A `<div>` element is tagged with an id `info` so that it can be referred by `getElementById()` method in the `showdata()` method of JavaScript file to display contents dynamically.

prog2.js

```
function showdata()
{
obld=document.createElement("b");
omsg=document.createTextNode("We provide wide variety of items at reasonable
price");
obld.appendChild(omsg);
getdiv= document.getElementById("info");
getdiv.appendChild(obld);
}
```

What this Program is Doing

This JavaScript file contains a single function `showdata()` which when invoked creates an element named `obld` with `<b>` (bold) tag applied to it. Thereafter, a text node named `omsg` is created with a message: "We provide wide variety of items at reasonable price". This text node is appended to the `obld` element (so that text appears in bold when displayed). In short, the `obld` element is made containing a text message in bold. To make this text appear on the browser screen, we first find the element in the document with id `info` (It is for this reason that we have created a div element in the web page with id `info`). To make the text message in bold appear on the browser screen, the `obld` element is appended to the div element with id `info`.

The output of the program prog2.html may appear as shown in Figure 6.2.

Output:

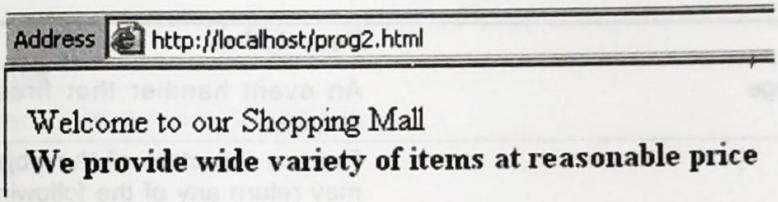

Figure 6.2 ***Output of the Program Prog2.html***

6.3 STEPS TO PLACE ASYNCHRONOUS REQUESTS TO THE SERVER

We know that the XMLHttpRequest object is used for making asynchronous calls to the web server. So before we start with its examples, let's see the outline of the methods and properties of the XMLHttpRequest object.

Table 6.1 ***Methods of the XMLHttpRequest Object***

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","boolean","uname", "pswd")	Specifies the method, URL, and other attributes of a request. The method can be of GET, POST, or PUT type. The boolean parameter specifies whether the request should be handled asynchronously or not. `True` means the request is asynchronous and `false` means the request is a synchronous one.
send(content)	Sends the request.
SetRequestHeader("label", "value")	Adds a label/value pair to the HTTP header to be sent.

Table 6.2 *Properties of the XMLHttpRequest Object*

Property	Description
onreadystatechange	An event handler that fires at every state change.
readystate	Returns the state of the object:(read-only). It may return any of the following values: 0 = uninitialised 1 = loading 2 = loaded 3 = interactive 4 = complete
responseText	Returns the response as a string (read-only)
responseXML	Returns the response as XML.
status	Returns the status as a number (like 404 for "Not Found" or 200 for "OK", 500 for "Server Error").
statusText	Returns the status as a string: "Not Found" or "OK".

In all there are five steps involved in facilitating asynchronous requests (to perform server side processing without a postback):

1. Create an XMLHttpRequest object.
2. Using this object, request data from the server.
3. Monitor for the changing of state of the request.
4. If the response is successful, retrieve data from the response stream
5. Use the response in the web page.

These steps can be better explained with the help of a running example. The following program asks the user to enter his name. As the user begins typing his name, a welcome message appears on the screen. The JavaScript code is embedded in this program (but we can always make a separate file for keeping JavaScript code). Let's make two files named ajaxform1.php and ajaxdata1.php with the contents as below in the `htdocs` subfolder of the `apache` directory.

ajaxform1.php

```
1. <head>
2. <script language="JavaScript" type="text/JavaScript">
3. function makeRequestObject(){
4. var xmlhttp=false;
5. try {
6. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
```

```
7. } catch (e) {
8. try {
9. xmlhttp = new
10. ActiveXObject('Microsoft.XMLHTTP');
11. } catch (E) {
12. xmlhttp = false;
13. }
14. }
15. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
16. xmlhttp = new XMLHttpRequest();
17. }
18. return xmlhttp;
19. }

20. function showdata(user)
21. {
22. var xmlhttp=makeRequestObject();
23. var file = 'ajaxdata1.php?usernme=';
24. xmlhttp.open('GET', file + user, true);
25. xmlhttp.onreadystatechange=function() {
26. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
27. var content = xmlhttp.responseText;
28. if( content ){
29. document.getElementById('info').innerHTML = content;
30. }
31. }
32. }
33. xmlhttp.send(null)
34. }
35. </script>
36. </head>

37. <body>
```

```
38. Enter your Name: <input type="text" onkeyup="showdata(this.value)"/>
39. <div id="info"></div>

40. </body>
41. </html>
```

ajaxdata1.php

```
1. <?php
2. echo "Welcome " . $_GET['usernme'] . " to our Shopping Mall" ;
3. ?>
```

Explanation of the ajaxform1.php program

First of all the <body> section of any web page is executed. In the body section the code is:

```
Enter your Name: <input type="text" onkeyup="showdata(this.value)"/>
```

This code asks the user to enter his/her name and as any key is released (while typing the name), the matter typed so far is passed to the `showdata()` method as argument. Before we go further let's quickly understand different key events.

6.3.1 Handling Key Events

There are three events which deal with keys:

- keydown
- keypress
- keyup

Event	Description
keydown event	fires when a user presses a key on the keyboard but before any changes occur to the textbox
keypress event	fires only when a character key is pressed
keyup event	fires after changes are made to the textbox

In the `showdata()` method a XMLHttpRequest object is made. Let's recall the steps involved in performing asynchronous request to the server:

1. Create an XMLHttpRequest object.
2. Using this object, request data from the server.
3. Monitor for the changing of state of the request.
4. If the response is successful, retrieve data from the response stream
5. Use the response in the web page.

Step 1. Create an XMLHttpRequest object

XMLHttpRequest object is the object which enables JavaScript code to make asynchronous HTTP server requests. It is using this object that we can make HTTP requests, receive responses and update a region of the page completely in the background.

XMLHttpRequest object was implemented by Microsoft as an ActiveX object in Internet Explorer and is supported by all its versions except IE6.

Since in some browsers XMLHttpRequest is a native object whereas in Internet Explorer XMLHttpRequest is implemented as an ActiveX control, the method of making its instance is different for each browser.

```
var xmlhttp=makeRequestObject();
```

We are creating an XMLHttpRequest object named xmlhttp.

In order to make an HTTP request to the server using JavaScript, we need an instance of a class that provides this functionality. Such a class was originally introduced in Internet Explorer as an ActiveX object, called XMLHTTP. Then Mozilla, Safari and other browsers followed, implementing an XMLHttpRequest class that supports the methods and properties of Microsoft's original ActiveX object.

JavaScript has a built-in XMLHttpRequest() function which we can use for browsers like Firefox, Safari, and Opera. For Internet Explorer, we use the ActiveXObject; there is also a difference between IE 5.0 and IE 6.0+ in how to create the object. The following code creates an XMLHttpRequest for all browsers:

```
var xmlhttp;
if(window.XMLHttpRequest){
//For Firefox, Safari, Opera
xmlhttp = new XMLHttpRequest();
}
else if(window.ActiveXObject){
//For IE 5
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
} else if(window.ActiveXObject){
//For IE 6+
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
}
else{
//Error for an old browser
alert('Your browser is not IE 5 or higher, or Firefox or Safari or Opera');
  }
```

Here, first we are using the built-in JavaScript function XMLHttpRequest() for creating an XMLHttpRequest object for Firefox, Safari and Opera. If the browser does support window. Active XObject, then it is Internet Explorer. For IE versions 5.0+, use new ActiveXObject ("Microsoft. XMLHTTP") and for IE 6.0+ use new ActiveXObject("Msxml2.XMLHTTP"). If the browser does not support the built-in JavaScript function XMLHttpRequest() or ActiveXObject, then it does not support AJAX. We can also use JavaScript try-catch blocks for creating XMLHttpRequest object as used in above program:

```
4. var xmlhttp=false;
5. try {
6. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
7. } catch (e) {
8. try {
9. xmlhttp = new
10.ActiveXObject('Microsoft.XMLHTTP');
11.} catch (E) {
12.xmlhttp = false;
13.}
14.}
15.if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
16.xmlhttp = new XMLHttpRequest();
17.}
```

Here first we are trying to create a XMLHttpRequest object using Active XObject ("Msxml2.XMLHTTP") assuming that the browser is IE6+, and if it fails we try ActiveXObject("Microsoft.XMLHTTP") assuming the browser is IE5.0. If all these fail, the built-in function XMLHttpRequest() is used to create the XMLHttpRequest object.

Step 2. Using this object, request data from the server

After creating the XMLHttpRequest object, we now need to send the web request to get data from the server. The request is made using the open method.

Syntax:

```
XMLHttpRequest object.open(HTTP request method, filename to execute, boolean)
```

Where

HTTP request method:	This may be GET, POST or any other method that is supported by our server. Note: These methods are always written in upper case.
filename:	It is the URL of the file residing on the server that we are requesting. This file when executed retrieves the desired information.

boolean: It is an optional parameter to specify whether the request is asynchronous or not. If it is set to `true`, it means the request is asynchronous, that is the execution of the JavaScript function will continue without waiting for the server response (in that case, we must use an event handler to watch for the response to the request). If this parameter is set to `false`, the request is set synchronously which means JavaScript waits for a response from the server before continuing execution of code.

Example:

```
xmlhttp.open("GET","filename.php", true);
```

Here, xmlhttp is the XMLHttpRequest object. It requests the server to execute filename.php file using GET method.

Let's understand the following instruction used in the program above:

```
23. var file = 'ajaxdata1.php?usernme=';
24. xmlhttp.open('GET', file + user, true);
```

In the instruction above, we are making a request to the server with GET method and passing the file name ajaxdata1.php to be executed (residing on the server). A variable `usernme` set to the name entered by the user is also sent.

```
33. xmlhttp.send(null)
```

This is the instruction which actually sends the request to the server. The argument in it is a string for the requested file. If the request file doesn't require anything, the argument passed to this method is null. In other words, if there is no query string to be sent to the file being invoked, ajaxdata1.php, we write null in the `send()` method otherwise the query string is sent instead of null. (Query string is used while using POST method explained later).

Note: GET request doesn't require any query string to be sent to the requested file.

But things will be different in the case of the POST method.

Step 3. Monitor for the changing of state of the request

```
25. xmlhttp.onreadystatechange=function() {
26. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
27. var content = xmlhttp.responseText;
```

As said earlier when an asynchronous request is made to the server, we need to watch for the state of the request and the response to the request. For doing so, we assign a function to xmlhttp.onreadystatechange which continuously checks the status of the request:

```
xmlhttp.onreadystatechange=function()
{
if(xmlhttp.readyState==4 && xmlhttp.status == 200)
```

```
{
var resp = xmlhttp.responseText;
}
}
```

Or like this,

```
xmlhttp.onreadystatechange = functionname;

function functionname (){
if(xmlhttp.readyState == 4 && xmlhttp.status == 200){
var response = xmlhttp.responseText;
}
}
```

The readyState property holds the status of the server's response. Each time the readyState changes, the onreadystatechange function will be executed. Here are the possible values for the readyState property:

State	Description
0	The request is not initialised
1	The request has been set up
2	The request has been sent
3	The request is in process
4	The request is complete

And status is the status of the HTTP Request, like 500 for Internal Server Error, 400 for Bad Request, 401 for Unauthorized, 403 for Forbidden, 404 for Not Found, 200 means no error etc.

The onreadystatechange function checks for the state of the request. If the state has a value of 4, that means that the full server response is received. If the status has a value of 200, that means the response is OK and without any error and we can continue processing it.

Step 4. If the response is successful, retrieve data from the response stream

```
26. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
27. var content = xmlhttp.responseText;
```

Since the response is successful, we need to retrieve and process the response data. To retrieve data from the response stream, we can use the following properties:

xmlhttp.responseText will return the server response as a string of text

xmlhttp.responseXML will return the response as an XMLDocument object which we can traverse using the JavaScript DOM functions

In the above instructions, we are retrieving the server response in the form of text and assigning it to the variable `content`.

Step 5. Use the response in the web page

In this step, the response retrieved from the server is displayed on the web page.

```
28. if( content ){
29. document.getElementById('info').innerHTML = content;
30. }
```

In the instructions above, the object with id `info` is searched for in the document (web page). Recall that element(s) are placed in the web page tagged with unique ids. These elements are searched for with the help of `getElementById()` methods and their contents can be modified dynamically. In the instructions above, the element with id `info` is searched for in the web page and its innerHTML property (property used to display results) is assigned the server response stored in variable `content`. So, the response retrieved from the server is displayed at the place of element of id: `info`.

Note the response generated by the server is based on the execution of the file ajaxdata1.php (placed on the server). Let's analyse what it returns.

Explanation of the ajaxdata1.php program

ajaxdata1.php

```
1. <?php
2. echo "Welcome " . $_GET['usernme'] . " to our Shopping Mall" ;
3. ?>
```

This program returns the text "Welcome name of the user to our Shopping Mall" to the client. The name of the user is retrieved from `$_GET` array (explained in Chapter 4).

The output of the program ajaxform1.php may appear as shown in Figure 6.3.

Output:

As we can see in the figure, the screen displays a textbox for the user to enter his name.

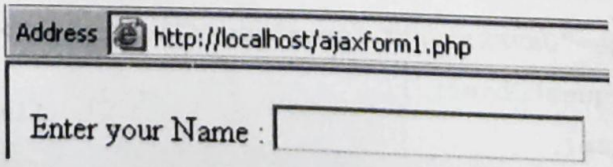

Figure 6.3 ***Screen Displaying a Textbox to Enter Name***

The moment the user presses a key, a welcome message is displayed along with the matter typed by him (Figure 6.4).

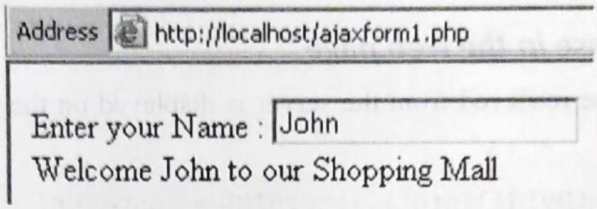

Figure 6.4 ***As Soon as the User Finishes Typing, a Message is Displayed on the Screen***

That is, with every character typed, we get the output displayed on the screen.

6.3.2 Using onblur() Event

While running the program above, we find that it doesn't wait for feeding the complete name, instead every character typed is displayed on the screen. If we want the name to be displayed when the user has finished, then the statement in <body> element is changed as follows:

```
38. Enter your Name: <input type="text" onblur="showdata(this.value)"/>
```

onblur event on a control results when a user press tab key (after feeding data) or clicks elsewhere on the web page, meaning when the focus on that control is lost.

6.4 ACCESSING FORM ELEMENTS

We know that `getElementById()` method is used for searching any element in the form with the given ID. Let's see the following example which demonstrates how this method can be used to access the data stored in different form elements.

This program asks the user to enter his name and email id which are then displayed on screen when the submit button is pressed. Let's make two files named ajaxaccess.php and ajaxaccessdata1.php with the contents as below in the `htdocs` subfolder of the `apache` directory.

ajaxaccess.php

```
1. <head>
2. <script language="JavaScript" type="text/JavaScript">
3. function makeRequestObject(){
4. var xmlhttp=false;
5. try {
6. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
7. } catch (e) {
```

```
8. try {
9. xmlhttp = new
10. ActiveXObject('Microsoft.XMLHTTP');
11. } catch (E) {
12. xmlhttp = false;
13. }
14. }
15. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
16. xmlhttp = new XMLHttpRequest();
17. }
18. return xmlhttp;
19. }

20. function showdata()
21. {
22. var xmlhttp=makeRequestObject();
23. user=document.getElementById('user_name').value;
24. email=document.getElementById('email_id').value;
25. var file = 'ajaxaccessdata1.php?usernme=';
26. xmlhttp.open('GET', file + user+'&emailid='+email, true);
27. xmlhttp.onreadystatechange=function() {
28. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
29. var content = xmlhttp.responseText;
30. if( content ){
31. document.getElementById('info').innerHTML = content;
32. }
33. }
34. }
35. xmlhttp.send(null)
36. }
37. </script>
38. </head>
```

```
39. <body>
40. Enter your Name: <input type="text" name="user_name"><br>
41. Enter your email id : <input type="text" name="email_id"><br>
42. <input type="button" onclick="showdata()" value="Submit"><br><br>
43. <div id="info"></div>
44. </body>
45. </html>
```

Explanation of ajaxaccess.php program

First of all the <body> section of any web page is executed. In the body section the code is:

```
40. Enter your Name: <input type="text" name="user_name"><br>
41. Enter your email id : <input type="text" name="email_id"><br>
42. <input type="button" onclick="showdata()" value="Submit"><br><br>
43. <div id="info"></div>
```

The statements above prompt the user to feed his/her name and email id which are stored in variables named `user_name` and `email_id` respectively. Also it specifies that when submit button is clicked, it will invoke `showdata()` method. Finally, a div element with id `info` is defined for displaying the response sent by the server.

In the `showdata()` method as explained in the earlier program, an XMLHttpRequest object is created.

```
23. user=document.getElementById('user_name').value;
24. email=document.getElementById('email_id').value;
```

The values entered in the textboxes named `user_name` and `email_id` in the <body> section are retrieved and stored in variables `user` and `email` respectively.

```
25. var file = 'ajaxaccessdata1.php?usernme=';
26. xmlhttp.open('GET', file + user+'&emailid='+email, true);
```

We are making a request to the server with GET method and pass the filename ajaxaccessdata1.php to be executed and also pass the name and email id of the user to this filename. The data is passed to a file in the following format:

```
filename?variable1=value1&variable2=value2.....
```

In the target filename, the values: value1, value2 can be accessed using `$_GET` array

```
Example: echo $_GET['variable1'];
```

Since an asynchronous request is made to the server, we need to watch for the state of the request and the response to the request. For doing so, we take the help of the event handler: ***onreadystatechange*** which fires at every state change. So, whenever the state of the request

changes, the `function()` is executed where value of the *readyState* property is checked (to see if it has become 4 meaning the request is complete i.e. full server response is received) and also the status of the HTTP request is checked to see if its value is 200 which means no error.

The response is then retrieved from the response stream in the form of text (it can be in XML and JSON form also) and is assigned to the variable `content`.

Then, the element with id `info` is searched for in the document (web page) and its innerHTML property (property used to display results) is assigned the server response stored in the variable. So, the response retrieved from the server is displayed at the place of element `info`.

The instruction: `xmlhttp.send(null)` is used to actually send the request to the server. We don't want to pass anything to the requested file, so, the argument passed to this method is null. That is, if there is no query string to be sent to the file being invoked ajaxaccessdata1.php, we write null in the `send()` method otherwise the query string is sent instead of null.

Note the response generated by the server is based on the execution of the file ajaxaccessdata1.php. Let's analyse what it returns.

ajaxaccessdata1.php

```
<?php
echo "Welcome " . $_GET['usernme'] . " to our Shopping Mall <br>" ;
echo "Your email id is " . $_GET['emailid'] ;
?>
```

This returns two lines of text to the client: "Welcome `name` of the user to our Shopping Mall" and "Your email id is ". The name and email id of the user are retrieved from `$_GET` array.

The output of the program ajaxaccess.php will be as shown in Figure 6.5.

Output:

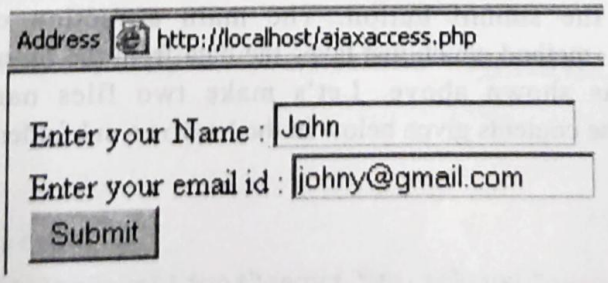

Figure 6.5 ***Screen to Enter Name and Email Id***

After feeding the name and email id, when we press Submit button, both are displayed (Figure 6.6).

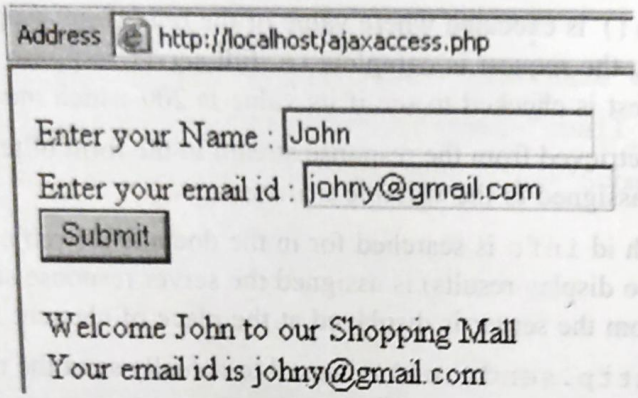

Figure 6.6 *After Pressing Submit Button, Both Messages are Displayed*

6.5 XMLHTTP POST REQUESTS

While using POST requests, we have to pass the information to the requested file in the form of a query string as shown:

```
name=value1&address=value2&….
```

The data is first arranged in this format before being sent to the server. One more thing to remember is that if we want to POST data, we have to change the MIME type of the request using the following line:

```
httpRequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

Otherwise, the server will discard the POSTed data.

The following program asks the user to enter his name and then he is greeted with a welcome message on clicking the submit button. The main attraction of this program is the `setQueryString()` method which just takes the data from the form and converts them in the query string format as shown above. Let's make two files named ajaxpost1.php and ajaxpostdata1.php with the contents given below in the `htdocs` subfolder of the `apache` directory.

ajaxpost1.php

```
1. <head>
2. <script language="JavaScript" type="text/JavaScript">
3. var queryString;
4. function makeRequestObject(){
5. var xmlhttp=false;
6. try {
7. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
8. } catch (e) {
```

```
9. try {
10. xmlhttp = new
11. ActiveXObject('Microsoft.XMLHTTP');
12. } catch(E) {
13. xmlhttp = false;
14. }
15. }
16. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
17. xmlhttp = new XMLHttpRequest();
18. }
19. return xmlhttp;
20. }

21. function showdata()
22. {
23. var xmlhttp=makeRequestObject();
24. setQueryString();
25. var file = 'ajaxpostdata1.php';
26. xmlhttp.open('POST', file, true);
27. xmlhttp.setRequestHeader("Content-Type",
28. "application/x-www-form-urlencoded; charset=UTF-8");
29. xmlhttp.onreadystatechange=function() {
30. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
31. var content = xmlhttp.responseText;
32. if( content ){
33. document.getElementById('info').innerHTML = content;
34. }
35. }
36. }
37. xmlhttp.send(queryString)
38. }

39. function setQueryString(){
40. queryString="";
41. var frm = document.forms[0];
42. var numberElements = frm.elements.length;
43. for(var i = 0; i < numberElements; i++) {
```

```
44. if(i < numberElements-1) {
45. queryString += frm.elements[i].name+"="+
46. encodeURIComponent(frm.elements[i].value)+"&";
47. } else {
48. queryString += frm.elements[i].name+"="+
49. encodeURIComponent(frm.elements[i].value);
50. }
51. }
52. }

53. </script>
54. </head>
55. <body>
56. <form method="post" onsubmit="showdata(); return false">
57. Enter your Name: <input type="text" name="usernme"/><br/>
58. <button type="submit">Submit</button>
59. </form>
60. <div id="info"></div>
61. </body>
62. </html>
```

ajaxpostdata1.php

```
1. <?php
2. echo "Welcome " . $_POST['usernme'] . " to our Shopping Mall" ;
3. ?>
```

Explanation of the ajaxpost1.php program

The program begins its execution from the <body> element onwards.

```
56. <form method="post" onsubmit="showdata(); return false">
```

onsubmit is an event handler which executes when submit type button is pressed. This event handler calls the `showdata()` function. This event handler returns `false` to prevent actual form submission. That is, we don't want the form to be submitted to server instead we just want the control to invoke the `showdata()` function.

In the `showdata()` method, we are first making an object to make an HTTP request to the server.

Since in POST requests, data sent to the server is in the form of a query string, like:

```
name=value1&address=value2&.....
```

So, to convert the data fed by the user in the form into above format, the `setQueryString()` method is invoked.

In the query string, the name and the value of each parameter must be URL-encoded in order to avoid data loss during transmission. `encodeURIComponent()` is a built-in function in JavaScript which is used to perform this encoding.

Note that if you want to use POST request, you have to change the MIME type of the request using the following line:

```
httpRequest.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

Otherwise, the server will discard the POSTed data.

The output of the program ajaxpost1.php may appear as shown in Figure 6.7.

Output:

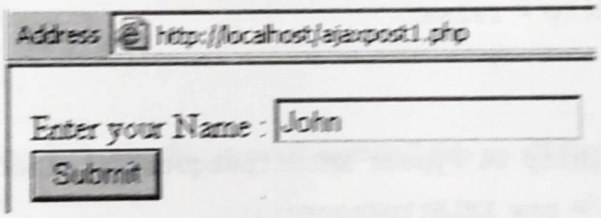

Figure 6.7 *Enter the Name*

We feed the user name and press Submit button, we get the server response as shown in Figure 6.8.

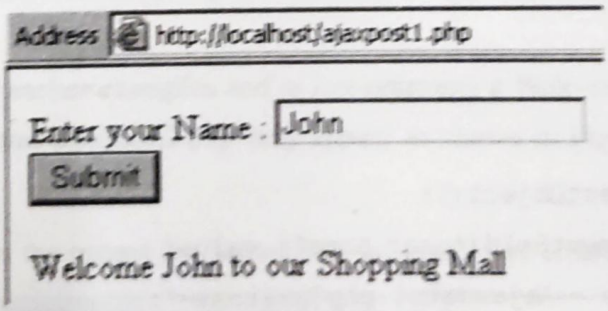

Figure 6.8 *Server Response is Displayed*

6.6 SEPARATING JAVASCRIPT CODE IN ANOTHER FILE

Instead of writing the JavaScript code in the <script> tag of a HTML file which looks a bit untidy, it is always better to write the JavaScript code in a separate file with .js extension and then refer the JavaScript file in the current web page. This makes the web page appear neat and readable. For

example in the following program, we are writing the JavaScript code in a separate file ajaxform2.js which is then referred in the file ajaxform2.php file.

ajaxform2.js

```
function makeRequestObject(){
var xmlhttp=false;
        try {
              xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
        } catch (e) {
              try {
                        xmlhttp = new
                        ActiveXObject('Microsoft.XMLHTTP');
            } catch (E)  {
              xmlhttp = false;
                        }
            }
            if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
            xmlhttp = new XMLHttpRequest();
            }
return xmlhttp;
}

function showdata()
{
var xmlhttp=makeRequestObject();
user=document.getElementById('user_name').value;
            var file = 'ajaxdata1.php?usernme=';
        xmlhttp.open('GET', file + user, true);
        xmlhttp.onreadystatechange=function() {
            if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
                        var content = xmlhttp.responseText;
                        if( content ){
                            document.getElementById('info').innerHTML = content;
```

```
            }
        }
        }
        xmlhttp.send(null)
}
```

ajaxform2.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="ajaxform2.js"></
script>
</head>
<body>
Enter your Name: <input type="text" name="user_name"><br>
<input type="button" onclick="showdata()" value="Submit"><br>
<div id="info"></div>
</body>
</html>
```

We can see that the above php script file appears neat and tidy as all the JavaScript code is now in a separate file ajaxform2.js.

ajaxdata1.php

```
<?php
echo "Welcome " . $_GET['usernme'] . " to our Shopping Mall" ;
?>
```

This file is reused from earlier examples and is just returning a Welcome message to the client.

The output of the program ajaxform2.php may appear as shown in Figure 6.9.

Output:

There will be no change in the output (on separating the JavaScript code).

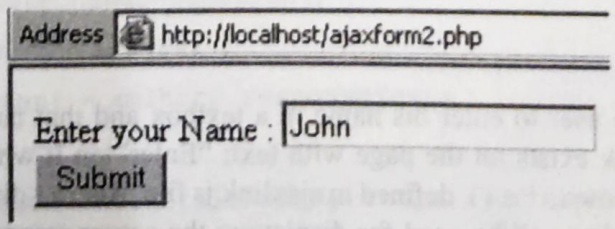

Figure 6.9 ***Enter the User Name***

On entering the user's name and selecting Submit button, a Welcome message appears as shown in Figure 6.10.

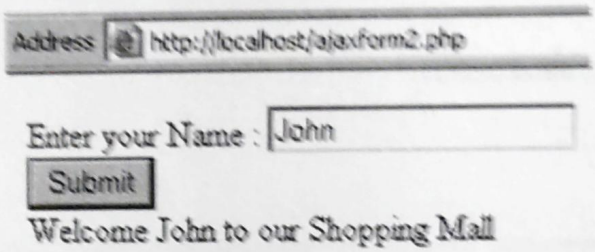

Figure 6.10 *Server Response is Displayed (no Change in the Output)*

6.7 ACCESSING JAVASCRIPT FUNCTION USING HYPERLINK

The following program asks the user to enter his/her name. After entering the name when the user selects a hyperlink, a JavaScript method is invoked which displays a Welcome message to the user. In other words instead of using any events like onkeyup or onblur etc., in this example, a hyperlink is used for sending a request to the server asynchronously. Let's make two files named ajaxlink.php and ajaxlink.js with the contents below in the `htdocs` subfolder of the `apache` directory. We also need a file ajaxdata1.php created in earlier examples for returning server response.

ajaxlink.php

```
<head>
<script language="JavaScript" type="text/JavaScript" src="ajaxlink.js">
</script>
</head>
<body>
Enter your Name: <input type="text" name="user_name"><br>
<a href="javascript:showdata()"> Enter</a><br>
<div id="info"></div>
</body>
</html>
```

This program asks the user to enter his name in a textbox and that name is assigned to variable "user_name". A hyperlink exists on the page with text: "Enter" on it which when selected invokes the JavaScript method `showdata()` defined in ajaxlink.js file. Also a element with id `info` is defined in the web page that will be used for displaying the server response.

ajaxlink.js

```
function makeRequestObject(){
var xmlhttp=false;
      try {
            xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
      } catch (e) {
            try {
                  xmlhttp = new
                  ActiveXObject('Microsoft.XMLHTTP');
         } catch (E) {
            xmlhttp = false;
                  }
      }
      if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
            xmlhttp = new XMLHttpRequest();
      }
return xmlhttp;
}

function showdata()
{
var xmlhttp=makeRequestObject();
user=document.getElementById('user_name').value;
      var file = 'ajaxdata1.php?usernme=';
   xmlhttp.open('GET', file + user, true);Query String
   xmlhttp.onreadystatechange=function() {
      if (xmlhttp.readyState==4) {
            var content = xmlhttp.responseText;
            if( content ){
                  document.getElementById('info').innerHTML = content;
            }
      }
      }
```

```
        xmlhttp.send(null)
}
```

This program is first making an XMLHttpRequest object and then making a request to the server for the file ajaxdata1.php, sending the user name entered by the user to it. The ajaxdata1.php generates the Welcome message as server response. The div element with id `info` is located in the web page and the server response is pasted at that element by setting its innerHTML property.

ajaxdata1.php

```
<?php
echo "Welcome " . $_GET['usernme'] . " to our Shopping Mall" ;
?>
```

This file is reused from earlier examples and is just returning a Welcome message to the client.

Output:

The user is prompted to enter his name (Figure 6.11).

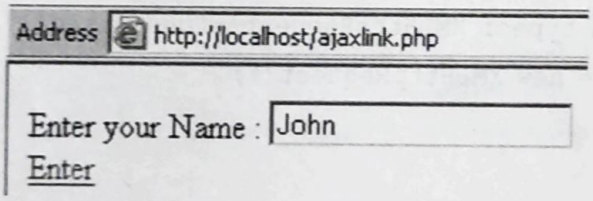

Figure 6.11 *Enter the User Name*

After entering the name, when user clicks the link: "Enter", a welcome message is displayed (Figure 6.12).

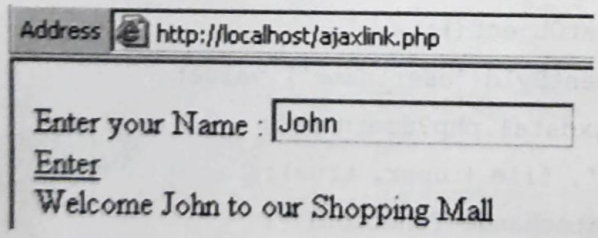

Figure 6.12 *Welcome Message is Displayed*

6.8 SPECIFYING OUR FUNCTION IN .JS FILE

In the above JavaScript file, though the function is executed when the status of the request changes and the event is handled by onreadystatechange handler, we haven't specified any name for the function. The code is written is as follows:

```
function showdata()
{
var xmlhttp=makeRequestObject();
user=document.getElementById('user_name').value;
        var file = 'ajaxdata1.php?usernme=';
        xmlhttp.open('GET', file + user, true);
        xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
                var content = xmlhttp.responseText;
                if( content ){
                        document.getElementById('info').innerHTML = content;
                }
        }
        }
        xmlhttp.send(null)
}
```

We can see in the above code, that the statement:

```
xmlhttp.onreadystatechange=function() {
```

specifies that if any change in the status of the request takes place, execute the following function body `function()`, but the function name is not specified.

In the following sample, we have re-written this code with the difference that we have specified the function to be executed if any change in the status of the request takes place. The function name specified is `getdata()` and it is assigned to the onreadystatechange event handler with the following command:

```
xmlhttp.onreadystatechange=getdata;
```

This instruction executes the `getdata()` method if there is any change in the status of the request. So, our `showdata()` method will be modified as follows:

```
function showdata()
{
xmlhttp=makeRequestObject();
var user=document.getElementById('user_name').value;
        var file = 'ajaxdata1.php?usernme=';
        xmlhttp.open('GET', file + user, true);
    xmlhttp.onreadystatechange=getdata;
```

```
        xmlhttp.send(null)
}

function getdata()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
                var content = xmlhttp.responseText;
                if( content ){
                    document.getElementById('info').innerHTML = content;
                }
        }
}
```