3

Starting with PHP Programming

PHP is an open source server side scripting language with no license fees. Moreover, it is flexible and very easy to learn, which makes it very popular among web developers. In the previous chapters we saw the process of installation and configuration of AMP (Apache, MySQL and PHP) as all three combine to make an environment for developing dynamic websites. We have also seen MySQL commands for creating and deleting databases and tables, defining and altering the structure of tables and maintaining the contents of a table. This chapter covers all the essential concepts of PHP programming which a user "must know" before beginning with the practical examples of "PHP and MySQL programming", i.e. before learning how the MySQL database can be manipulated via PHP scripts. The chapter provides information on the following:

- Basic syntax and commands of PHP
- Echo command
- Variables in PHP
- Comments in PHP
- PHP operators
- Conditional statements
- Looping
- Arrays
- Associative array
- Multi-dimensional arrays for each statement
- Forms
- $_GET Array
- $_POST Array

- $_REQUEST
- Functions
- Passing parameters to functions
- Returning a value
- Global variables

3.1 STARTING WITH PHP PROGRAMMING

A PHP file normally contains HTML tags, and some embedded PHP scripting code. It certainly reveals one more advantage of PHP that it can be embedded in HTML pages. To enable the PHP interpreter to distinguish PHP codes from HTML tags, the PHP commands are enclosed within special start and end tags: <?php and ?>. Let's see the first example:

```
<html>
   <body>
      <?php echo "Believe in God"; ?>
   </body>
</html>
```

We can see in this simple script that PHP commands can be easily embedded with HTML with the help of special start and end tags:

```
<?php
PHP commands
?>

<?php tag means open the PHP mode and
?> tag means close the PHP mode.
```

There are two basic statements to output text with PHP: `echo` and `print`. In the example above we have used the `echo` statement to output the text "Believe in God".

Let us see how to execute this PHP script.

3.1.1 Steps to Run PHP Script

To run the above PHP script, the steps are as under:

- Type the above program using any editor and save it by name, say, `prog1.php` (file extension must be php) in the `htdocs` subfolder of apache directory. Recall from Chapter 1 that on installation, the Apache web server creates an `apache` directory with one of its

subfolders named: "`htdocs`". In order for the Apache web server to recognise PHP scripts, they have to be placed in the `htdocs` sub folder.

- Make sure that the Apache web server is running. To confirm it, we need to open the Apache Service Monitor dialog box. To open the Apache Service Monitor dialog box, right click on Apache server icon in the taskbar and select the "`Open Apache Monitor`" option (Figure 3.1).

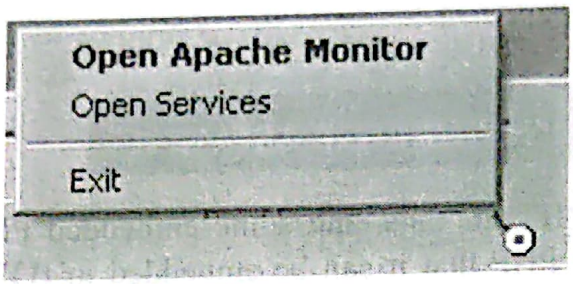


Figure 3.1 *Apache Icon in Taskbar*

The Apache Service Monitor dialog box shows the status of the Apache server i.e., whether it is in running mode or is in stopped mode (Figure 3.2). Not only can we view the status of the Apache server from this dialog box but also control its status.

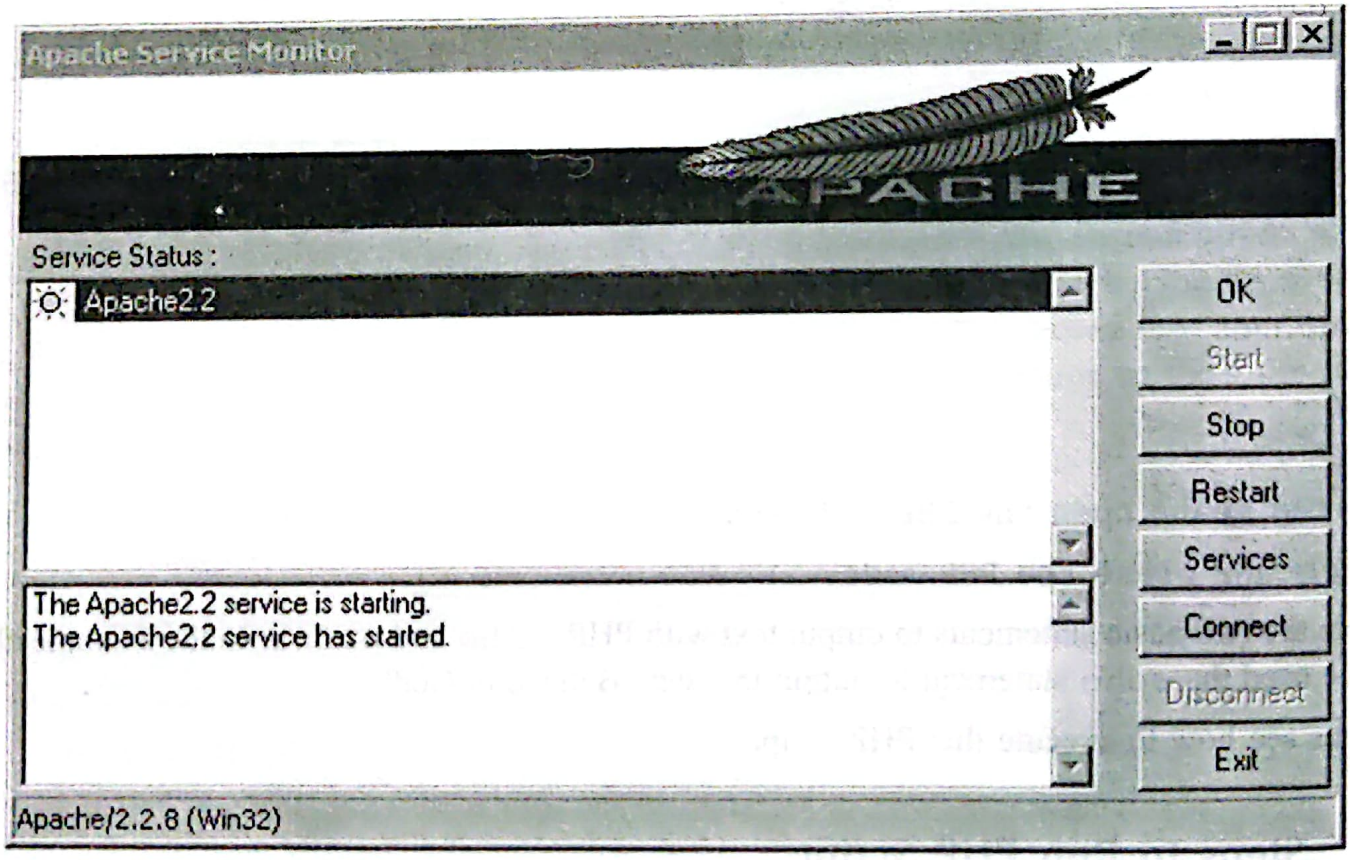


Figure 3.2 *Apache Service Monitor Window Showing Status of Apache Server*

Figure 3.2 confirms that the PHP server is running. If we find it in Stop mode, we can run it by clicking the `Start` button.

- We can now execute our PHP script by opening the browser and pointing it to the following address: `http://localhost/prog1.php`. The PHP script displays the output in the form of a text message: "Believe in God" (Figure 3.3).

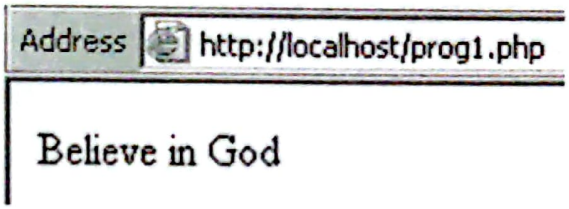


Figure 3.3 ***Output of prog1.php Script***

Note: The term "`localhost`" used in the URL: "http://localhost/prog1.php" signifies that our PHP script is located on the local web server. It is replaced by the domain name when the PHP script is hosted on the remote server. For example: http://bmharwani.com/prog1.php (assuming that the script is uploaded on server: bmharwani.com).

In the above example, when we request our local Apache web server for PHP script prog1.php, it assigns the script to the PHP interpreter. PHP parses the script and executes the code between the special start and end tags (`<?php` and `?>`), generates the output and sends it back to the web server. The web server transmits the output in the form of HTML code to our browser screen (client's browser).

3.2 BASIC SYNTAX AND COMMANDS

PHP code is written between the special start and end tags: `<?php` and `?>` tags.

- <?php means "begin PHP code" and
- ?> means "end PHP code".

The PHP interpreter reads and interprets only the statements between these tags and the statements before and after these tags are simply ignored, assumed to be HTML code. The PHP scripting block can be placed anywhere in the HTML document. Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

Example:

```
<html>
    <head>
    </head>
    <body>
       This is HTML code
    ..........
    ..........
```

```
        <?php
            PHP statement must be terminated with semi colon;
            PHP statement;
            PHP statement;
        ?>
    </body>
</html>
```

The best part of using PHP is that not only can it be easily embedded in HTML but we can also switch between PHP and HTML and vice versa easily. Let's see this with the following example:

```
<html>
    <head>
    </head>
    <body>
        <h1>Welcome to PHP</h1>
        <?php
            echo '<b>Believe in God</b>';
        ?>
    </body>
</html>
```

Note: The `echo` statement is used to send output to the browser.

As we can see the above program appears as a simple HTML document with an open HTML tag, head, body tag, etc. `<?php` tag marks the beginning of PHP code and **?>** tag designates that PHP coding is over. The PHP code in between <?php and ?> is converted into HTML code.

The point to remember is that not PHP code but the output of the PHP code (in the form of HTML) is actually sent to the client. The output that may be sent to the client for the example above is as follows:

```
<html>
    <head>
    </head>
    <body>
        <h1>Welcome to PHP</h1>
        <b>Believe in God</b>
    </body>
</html>
```

If a user views the source of a PHP page in a text editor they will not see the PHP code; all they will see is the output of that code. Again, this is another advantage of using PHP; nobody can see the actual PHP script used in our web application but only the generated HTML code, making it more secure.

3.2.1 Variables in PHP

While making web applications, we need to store the information entered by the user temporarily. We also need to store intermediate results (of processing) for final outcome. To serve this requirement of temporary storage, variables are used. Variables may be used not only for storing the data entered by the user but also for storing constant numerical values or text. The value to the variable is assigned with the help of the assignment operator (=). All variables in PHP start with a $ sign symbol followed by either a letter or underscore. After the $ sign, the variable should not begin with a numerical or any other symbol other than underscore.

Example:

```
<html>
   <body>
      <?php
         $name="John";
         echo $name;
         $a=10;
         $b=20;
         echo $a+$b;
      ?>
   </body>
</html>
```

In this PHP script, the string: "John" (strings have to be enclosed in double quotes) is assigned to variable `$name` and two numericals, 10 and 20, are assigned to variables `$a` and `$b` respectively. The contents of `$name` variable and the sum of variables $a and $b are displayed using the `echo` statement.

When we run the above PHP script, the output we get is:

```
John 30
```

3.2.2 Echo

An `echo` statement is used for displaying the output on the browser screen. There are two basic statements to output text with PHP: `echo` and `print`.

Example:

```
<?php
    echo 'Believe in God';
    echo '<b>Be Polite in your approach <b>';
?>
```

`<?php` tag switches on the PHP mode. The first `echo` statement displays the string "Believe in God" and the second `echo` statement displays the string "Be Polite in your approach" in bold on the screen.

Note: The string is enclosed in single quotes.

An `echo` statement can be used with single quotes, double quotes or no quotes. Let's see when to use what.

Single quotes

We use single quotes when we want to print a message without any variables.

```
<?php
    echo 'Believe in God';
?>
```

No quotes

If we are just printing a variable, there is no need to use quotes at all.

```
<?php
    $msg = 'Believe in God';
    echo $msg;
?>
```

In the above example, we simply display the contents of the variable `$msg` without any accompanying text message.

Double quotes

These are used when we want to print contents of the variable along with a text message.

```
<?php
    $msg = 'Believe in God';
    echo "The slogan of the day is: $msg";
?>
```

In above example, the variable `$msg` is enclosed in double quotes along with the text: `"The slogan of the day is:"`. The output on the browser screen will be:

```
The slogan of the day is: Believe in God
```

3.2.3 String Concatenation

String concatenation deals with addition of two or more text messages, addition of two or more string variables or addition of text messages with string variables. For string concatenation, `dot (.)` operator is used:

```
<html>
   <body>
      <?php
         $a="Thanks";
         $b="Good Bye";
         echo "The two strings are " . $a . " and " . $b ;
      ?>
   </body>
</html>
```

The output of the script above will be:

```
The two strings are Thanks and Good Bye
```

3.2.4 Comments in PHP

Comments are an important part of program documentation. They are helpful in explaining the working of an application as well as for specifying the reason for using a particular function or code. Sometimes comments are used in debugging too, i.e., the part of the code giving erroneous results can be isolated by commenting out the rest of the code. In PHP, we use // to make a single-line comment. For comments extending over more than a line, enclose them between a pair of /* and *.

Example:

```
<html>
   <body>
      <?php
         //This is a comment
         /*
            This is
```

```
            a comment
            block
        */
    ?>
  </body>
</html>
```

3.3 PHP OPERATORS

Operators are used for performing certain operations. In PHP, we have the following types of operators:

Arithmetic Operators

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulus (division remainder)
++	Increment
– –	Decrement

Assignment Operators

Operator	Example	Meaning
=	x=y	x=y
+=	x+=y	x=x+y
–=	x–=y	x=x–y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description
==	is equal to
!=	is not equal
>	is greater than

<	is less than
>=	is greater than or equal to
<=	is less than or equal to

Logical Operators

Operator	Description
&&	and
||	or
!	not

The usage of these operators we will soon see in the examples that follow.

3.4 CONDITIONAL STATEMENTS

Conditional statements are used for choosing a set of statements out of two or more sets depending on the validity of the logical expression included. In PHP, we have two conditional statements:

- if (...else) statement
- switch statement

3.4.1 If Statement

An `if` statement consists of a logical expression which may result in `true` or `false`. If the result of the logical expression is true then the set of statements defined in the `if` block will be executed otherwise the set of statements in the `else` block will be executed.

Syntax:

```
if (logical expression)
      code to be executed if condition is true;
else
      code to be executed if condition is false;
```

Example:

```
<?php
   $a=20;
   $b=40;
   if($a > $b)
   {
      echo $a. " is larger ";
```

```
    }
    else
    {
        echo $b. " is larger ";
    }
?>
```

Output:

The statement defined in the `else` block will be executed and the output will be: "`40 is larger`" (Figure 3.4) because the logical expression: "`20 > 40`" is false

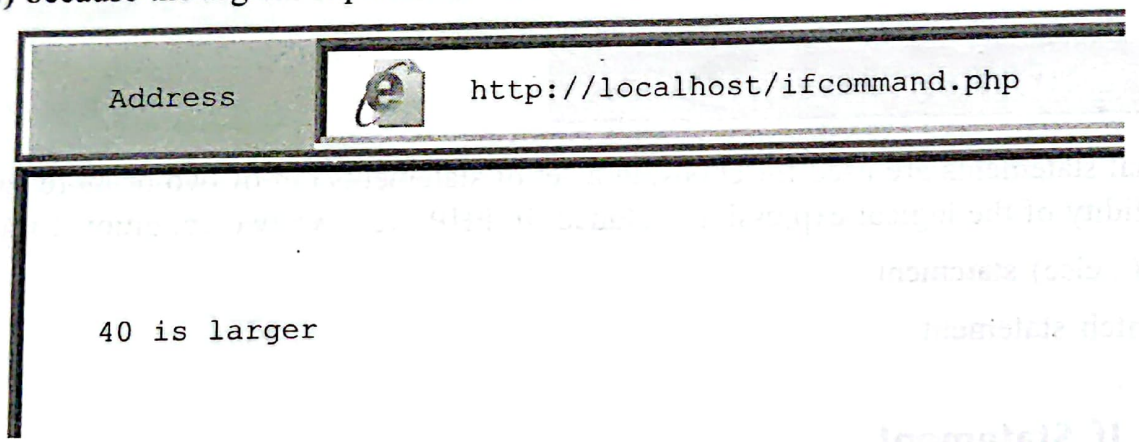


Figure 3.4 *Output of Ifcommand.php Script*

3.4.2 Switch Statement

If we have to choose one statement out of several statements depending on the value of an expression, then instead of using several `if else` statements, a `switch` statement is preferred. That is, the `switch` statement is a better alternative when there are several conditions to deal with.

Syntax:

```
switch (expression)
{
   case value1:
         code to be executed if expression = value1;
         break;
   case value2:
         code to be executed if expression = value2;
         break;
```

```
    :       :       :       :
    :       :       :       :
    :       :       :       :
    default:
          code to be executed if none of condition is met
}
```

The value of the expression is compared with the values for each `case` in the structure. If there is a match, the block of code associated with that case is executed. The `break` statement is used for coming out of the `switch` statement. If `break` is not used, it will execute the code of the next **case** automatically. The `default` statement is executed if the expression does not match any value specified in cases.

Example:

```
<?php
    $k=2;
    switch($k)
    {
       case 1:   echo "One";
                 break;
       case 2:   echo "Two";
                 break;
       case 3:   echo "Three";
                 break;
       case 4:   echo "Four";
                 break;
       default:
             echo "Sorry the value is not between 1 and 4 ";
    }
?>
```

The output of above script will be:

```
Two
```

Since the value of `$k` is `2`, it will execute the statement defined in `case 2` i.e., it will print the string: `Two`

3.5 LOOPING STATEMENTS

Looping statements in PHP are used to execute the same block of code a specified number of times. In PHP we have the following looping statements:

- while — loops through a block of code for the time the specified condition is true
- do...while — loops through a block of code once, and then repeats the loop as long as a special condition is true. Here the validity condition is checked at the end of the loop
- for — loops through a block of code for a specified number of times. This loop is usually used when we know beforehand how many times a loop is supposed to be executed
- foreach — loops through a block of code for each element in an array

3.5.1 While Loop

The `while` statement executes a block of code for the time the specified condition is true.

Syntax:

```
while (condition)
{
code to be executed;
.....
.....
}
```

Example:

```
<?php
   $i=1;
   while($i<=5)
   {
      echo $i . "<br>";
      $i++;
   }
?>
```

Output

We can see that the `while` loop in the above PHP script executes until value of `$i` crosses the value 5, hence displaying the sequence numbers from 1 to 5 (Figure 3.5).

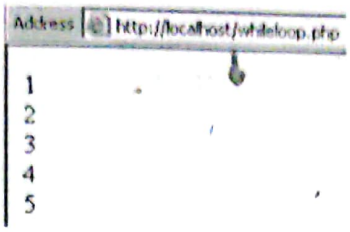


Figure 3.5 *Output of Whileloop.php Script*

3.5.2 Do. . . While Loop

In this loop too, a block of statements is executed for the time the given condition is true. But since here the validity condition is specified at the end of the loop, this loop executes at least once.

Syntax:

```
do
{
code to be executed;
.....
.....
}while (condition);
```

Example:

```
<?php
   $i=1;
   do
   {
      echo $i . "<br>";
      $i++;
   }while($i<=5);
?>
```

The above script also prints the sequence numbers from 1 to 5. The only difference is that here the validity check of the expression is done at the end of the loop as a result whatever may be the initial value of variable `$i`, the loop will execute at least once.

3.5.3 For Loop

This loop is used for repeating a set of statements a specified number of times. This loop is generally used when we already know how many times a statement(s) has to be executed.

Syntax:

```
for (initialization; condition; increment)
{
   code to be executed;
}
```

Note: A `for` statement has three parameters. The first parameter is for initialising variables, the second parameter holds the validity condition, and the third parameter contains any increment/decrement operator.

Example:

```
<?php
   for ($i=1; $i<=5; $i++)
   {
      echo $i . "<br>";
   }
?>
```

Note: Since foreach loop can work with arrays only, it is explained after the description of arrays.

3.6 ARRAYS

An array is a special kind of variable consisting of a collection of several memory locations in order to store multiple values. An array in PHP is created using the built-in array function: array()

The syntax of using array() is:

```
arrayname= array ( [element1, element2 ...] )
```

Example:

```
$p = array(10, 15, 22);
```

The above example creates an array by name `$p` which stores three values: 10, 15 and 22. The values are stored in the array at different memory locations referred to by different indices/subscripts. An index begins with 0. So 10 is stored at `$p[0]` index, 15 is stored at `$p[1]` index and 22 is stored at `$p[2]` index (Figure 3.6).

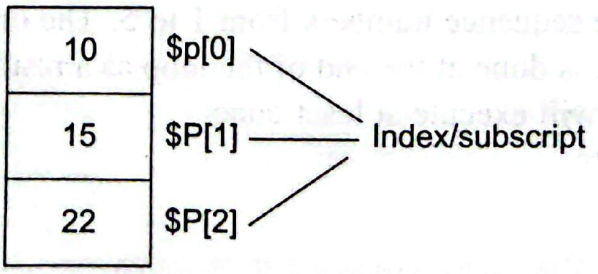


Figure 3.6 *Array p and its Contents*

These indices/subscripts i.e. 0, 1, 2... are used to update and retrieve the values stored in an array.

Example:

```
echo $p[0];
```

This will display value 10 on the screen (as 10 is stored at index `p[0]`).

An index can also be used to update or assign a new value to an array.

Example:

```
$p[1]=50;
```

It will update the value at `p[1]` index i.e. the value 15 will be removed and 50 will be stored at that index location.

```
$p[3] = 40;
```

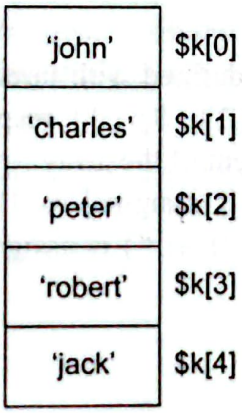


Figure 3.7 ***Contents of Array k***

It will add a new value to the array.

We can also create an array of strings:

```
$k = array('john', 'charles', 'peter', 'robert', 'jack');
```

Every string will be stored at its respective index, i.e., 'john' will be stored at `$k[0]` index, 'charles' will be stored at `$k[1]` index and so on (Figure 3.7).

3.6.1 Foreach Statement

This statement loops over the array. On each loop, the value of the current element is assigned to a variable and the array pointer is advanced by one.

Syntax:

```
foreach (array as value)
{
```

```
    code to be executed;
}
```

Example:

The following example demonstrates a loop that will print the values of the given array:

```
<?php
    $p=array("apple", "bat", "cat");
    echo "Data in array are: <br />";
    foreach ($p as $k)
    {
        echo "$k <br>";
    }
?>
```

In the above program, an array `$p` is defined with three strings: "apple", "bat", and "cat" which will be stored at locations: `$p[0]`, `$p[1]`, `$p[2]` respectively. In a `foreach` loop, `$k` is the variable which will be assigned one element of the array `$p` at a time. That is, the first element of the array `$p` is assigned to `$k` ("apple" is assigned to `$k`) and the loop is executed. In the next iteration, the next element of the array ("bat") is assigned to variable `$k` and again the loop is executed and so on.

Output of the above program:

```
Data in array are:
apple
bat
cat
```

3.6.2 Associative Array

This is an array which stores an element (value) bound to a key. That is, every element in this array is in terms of pairs: key-value pair

```
$student["roll"]=101;
$student["name"]="John";
$student["marks"]=85;
```

By the above examples of associative array, we can conclude that array indices can be strings too. This type of array is called associative because it associates values with meaningful indices.

We can also create an associative array by using the array() function.

```
$student=array("roll" => 101, "name" =>"John", "marks" => 85);
```

As with any array, to retrieve the values stored in an associative array, indices are used.

Example :

```
echo $student["name"];
```

This statement will print "John" on the screen.

Arrays can be single dimensional (as explained above) or multidimensional.

3.6.3 Multidimensional Arrays

These arrays are also known as array of arrays. These arrays therefore use two or more indices/subscripts. For a two-dimensional array, there are two index/subscript – one represents row and the other represents column.

Example:

```
$p[3][3];
```

This example defines an array of 3 rows and 3 columns. Since the indices/subscripts begin with 0, the first location of this two dimensional array is represented by `p[0][0]` (for 0^{th} row and 0^{th} column).

Let us assign the values to the two dimensional array as:

```
$p[0][0]=10;
$p[0][1]=20;
$p[1][0]=30;
$p[1][1]=40;
```

This array can be represented as shown in Figure 3.8.

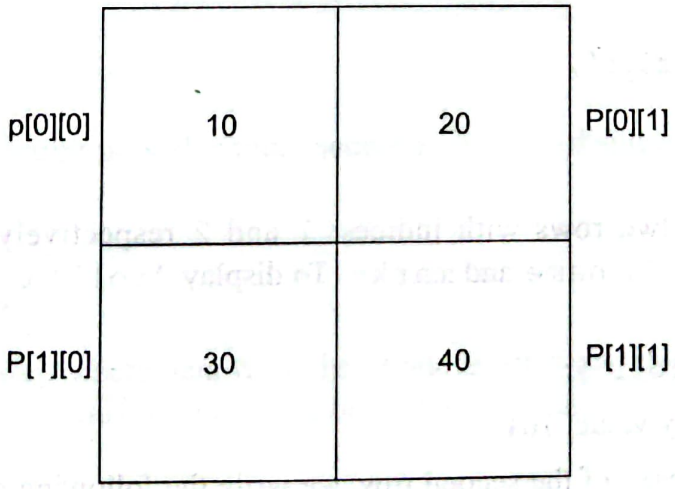


Figure 3.8 *Contents of a Two-dimensional Array: p*

Note: An index/subscript can be in form of strings too.

Example:

```
p['tea']=100;
```

Syntax of creating a multidimensional array:

```
$array = array(
        'key1' => 'value1',
        'key2' => 'value2',
        'key3' => array(
                'key1' => 'othervalue1',
                'key2' => 'othervalue2',
                'key3' => 'othervalue3'
                )
);
```

Example:

```
$student = array (
    '1' => array (
        'roll' => 101,
        'name' => 'John',
        'marks' => 85),
    '2' => array (
        'roll' => 102,
        'name' => 'Charles',
        'marks' => 72)
);
```

This example creates two rows with indices: `1` and `2` respectively and each row has three columns with indices: `roll`, `name` and `marks`. To display `'roll'` of the first row, we write the following command:

```
echo $student['1']['roll'];
```

This command will display value: 101

Similarly, to display `'name'` of the second row, we write the following command:

```
echo $student['2']['name'];
```

This command will display Charles.

3.7 FORMS

When we want some feedback from the user on some topic or want some information to be supplied by the user, we take the help of forms. A form consists of certain tags to accept user data and this data is then passed on to a specified PHP script for processing on the click of the submit button.

```
<form action="display.php" method="get">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
</form>
```

This form has two input boxes named "`name`" and "`age`". The form "`action`" points to a PHP file: "`display.php`" and the "`method`" is "`get`". The contents of the form will be submitted to the PHP script: "display.php" using the HTTP request-method `GET` for further processing.

The information is passed from one PHP script to another using any of the following HTTP request methods:

- GET
- POST

GET

In this method the information passed is less secure as it is displayed in the browser's address bar and moreover, there is a limit on the amount of information passed (it can be of a maximum 100 characters).

POST

In this method, the information passed is more secure as it is not displayed in the browser's address bar.

3.7.1 $_GET Array

The $_GET array is an array where data from the previous form sent using HTTP GET method are stored. The data from the previous form are sent in the form of pairs: `variable name(s)` and its `value(s)`.

Recall that information sent from a form with the GET method is less secure as it is displayed in the browser's address bar and the size of the information passed is limited.

When the user clicks the "`Submit`" button, the URL sent will look something like this:

```
http://localhost/display.php?name=john&age=22
```

We can see that when we use the GET method, all the information passed is attached in the browser address bar after the ? mark.

The target file: "display.php" file can now extract the data from $_GET array using following statements:

```
Welcome <?php echo $_GET["name"]; ?>.<br>
Your age is <?php echo $_GET["age"]; ?>
```

The name and age sent from the source PHP script is accessed from $_GET array and displayed.

3.7.2 $_POST Array

The $_POST array is an array which collects the values sent from a form using HTTP POST method.

Example:

```
<form action="display.php" method="post">
    Enter your name: <input type="text" name="name" />
    Enter your age: <input type="text" name="age" />
    <input type="submit" />
</form>
```

When the user clicks the "Submit" button, the $_POST["name"] and $_POST["age"] variables will be automatically set by PHP. The $_POST array contains all POST data and the URL will not display any form data (so it is more secure).

The destination PHP script: "display.php" file can now use the $_POST array to retrieve the form data with the help of the following code:

```
Welcome <?php echo $_POST["name"]; ?>.<br>
Your age is <?php echo $_POST["age"]; ?>
```

Note: Information sent from a form with the POST method is invisible to others and there is no limit to the amount of information sent.

To conclude: HTTP POST method is a better way of transmitting data because:

- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit

3.7.3 $_REQUEST Array

$_REQUEST array contains the contents of both $_GET and $_POST. That is, it is used to collect the information from a form which is sending data by either method: GET or POST method.

So, if we are not aware of the HTTP method used by the source PHP script, it is wise to access the information using a $_REQUEST array. The target file: display.php made in the earlier examples can be rewritten as:

Example:

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
Your age is <?php echo $_REQUEST["age"]; ?>
```

This will access the information sent from the source PHP file sent by any method: GET or POST.

3.8 FUNCTIONS

Functions are small modules or sub programs which are written once and can be invoked several times hence avoiding repetition of statements. Functions are usually made to manipulate and process data. Functions are of two types: built-in and user defined. PHP has hundreds of built-in functions each meant for a particular purpose. These built-in functions can be directly used in PHP scripts. User defined functions are made by the programmer and are also known as custom functions.

Example:

```
function disp_welcome() {
    echo 'Welcome to our Shopping Mall <br>';
    echo "You will find different varieties of items at lowest price';
}
```

We can see that a function is defined by writing function followed by the function name, followed by two brackets. We then open a curly bracket to start the function statements. It ends with a closing curly bracket. This function is invoked by just writing its name as shown:

```
<?php
    disp_welcome();
?>
```

The above function call will result in execution of statements defined inside the disp_welcome() function.

3.8.1 Passing Parameters to Functions

A function can be made more useful by asking it to work on the given data. We can then process different data by the same function by sending different values to it as parameters.

```
function disp_welcome($st1, $st2) {
```

```
    echo $st1 . '<br>';
    echo $st2;
}
```

This function is made to accept two parameters. So, while invoking this function, we have to send two parameters as shown:

```
<?php
disp_welcome('Welcome to our Shopping Mall', 'You will find different varieties
of items at lowest price');
?>
```

As we can see the two strings are passed while invoking the function which will be assigned to two arguments: `$st1` and `$st2`. These arguments: `$st1` and `$st2` are then displayed.

3.8.2 Returning a Value

A function can also return processed information. To return something to the caller program, a `return` statement is used.

```
function average($a, $b, $c) {
    $av=($a+$b+$c)/3;
    return $av;
}
```

This function calculates the average of three variables and returns the computed result to the caller.

3.9 GLOBAL VARIABLES

When any variable is defined within a function, it becomes its local variable and if we want that variable to be used in another function, we have to send it to another function as an argument. But if we define a variable as global, then it can be accessed by any of the functions and the effect of processing done by one function on that variable will be visible to another function.

To make a variable global, just define it at the top, that is, above the body of any function as shown:

```
<?php
    $l=50;
    $b=10;
    $a=0;
function area() {
```

```
    $a=$l*$b;
}
echo "Area of rectangle is " . $a;
?>
```

We can see in the above program that we have made three global variables, `$l`, `$b` and `$a`. Variables `$l` and `$b` are initialised to values 50 and 10 respectively and being global variables (declared before the function), they can be directly accessed in the `area()` function. Moreover, the area of rectangle computed by `area()` function in `$a` variable can be accessed outside the body of the function.

3.10 DISPLAYING INFORMATION ABOUT PHP

The phpinfo() function is used to output PHP information and is very useful for troubleshooting. The phpinfo() function options are:

Name	Description
INFO_GENERAL	The configuration line, php.ini location, build date, web server, system etc.
INFO_CREDITS	PHP 5 credits
INFO_CONFIGURATION	Local and master values for PHP directives
INFO_MODULES	Loaded modules
INFO_ENVIRONMENT	Environment variable information
INFO_VARIABLES	All predefined variables from EGPCS (Environment, GET, POST, Cookie, Server)
INFO_LICENSE	PHP license information
INFO_ALL	Shows all of the above. This is the default value

Example:

```
<html>
   <body>
      <?php
         // Show all PHP information
         phpinfo();
      ?>
      <?php
         // Show only the general information
```

```
        phpinfo(INFO_GENERAL);
    ?>
  </body>
</html>
```

Output:

All the information of PHP will be displayed (Figure 3.9).

PHP Version 5.2.5

System	Windows NT ADMIN-6BO2EOQSE 5.1 build 2600
Build Date	Nov 8 2007 23:18:08
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\Program Files\PHP\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

Figure 3.9 *Output of Phpinfo() Function*

SUMMARY

The outline of the topics that we covered in this chapter is:

- how to make and run small PHP programs
- how HTML and PHP scripts are embedded
- how variables are initialised in PHP scripts
- Usage of conditional statements: if else and switch statements
- how different loops work: while, do..while, for, foreach etc.

- how arrays are used: single dimensional, associative and multidimensional arrays
- how to get feedback from users with the help of forms
- role of $_GET, $_POST and $_REQUEST array in passing information from one form to another
- what are functions and how parameters are passed to them
- what are global variables

In the next chapter, we will concentrate on the creation of a database and tables via PHP scripts. That is, we will learn how to maintain the information in the data tables through PHP programs. We will see how to get information from the user and how that information can be stored in the back-end tables. We will also see how to retrieve desired information from the tables through the PHP interface and finally how to delete the undesired records from the table.

4

PHP and MySQL Programming

This chapter covers how to create databases and tables in MySQL via PHP scripts. Instead of maintaining the information in the data tables through the MySQL prompt, we will learn doing it through PHP programming. Maintenance of database tables through the MySQL prompt is cumbersome and also requires the user to be well versed with SQL commands. To make it easier for visitors to our website, we will provide them an easy-to-use GUI form (made in PHP) and the information entered in the form will be automatically stored in the database tables. Similarly, to fetch some information from the tables, the visitor will be asked to fill an enquiry form and the desired information will be retrieved from the database and displayed. The information of the database name, table name and different MySQL commands used to extract information from the database will not be visible to the visitor, making the web application more secure. In this chapter we will make and understand small PHP modules that perform the following tasks:

- Creating a table in PHP
- Getting information from the user
- Inserting records in the table
- Retrieving all records from the table
- Searching records from the table (retrieving only the desired records, not all)
- Deleting records from the table

Before we begin with these tasks, let us first look at all the statements and functions that we will require for accessing MySQL from PHP.

4.1 FUNCTIONS ACCESSING MYSQL FROM PHP

The following are the functions that we will be using for selecting databases, accessing tables, executing SQL statements and fetching results. Let us look at them one by one:

4.1.1 MySQL_Connect()

This function is used for establishing the connection to the MySQL database. It returns `true` if the connection is established else `false` is returned. This function takes three parameters; the first one is hostname, then user-id and then password.

Syntax:

```
mysql_connect ("$servername","$dbuser","$dbpassword");
```

Example:

```
$connect = mysql_connect("localhost", "root", "mce");
if(!$connect) { die("Please, check your server connection."); }
```

`Die` is the command to exit. If the `mysql_connect()` returns false then we exit from the PHP scripts using `die` command after displaying the message: "Please, check your server connection".

This example can also be written as:

```
$connect = mysql_connect("localhost", "root", "mce") or die("Please, check your
server connection.");
```

When two statements are connected with `or` operator, then the second statement is only executed if the first statement returns `false`.

4.1.2 Mysql_Select_db()

This function is used for selecting a MySQL database to be operated on.

Syntax:

```
bool mysql_select_db (string $database_name [, resource $link_identifier])
```

This selects the active database on the server with which connection is established by using a link identifier. If the link identifier is not specified, the last link opened by `mysql_connect()` is assumed. If no such link is found, it will create one by executing the MySQL with no arguments. An error is displayed if a connection is not established. It returns `true` if the database is found in the MySQL server else `false` is returned.

Examples:

- `mysql_select_db("shopping",$connect);`
- `mysql_select_db("shopping");`

4.1.3 Mysql_Query()

This function sends the query to the currently active database on the server and the records fetched are stored in an array.

Syntax:

```
resource mysql_query (string $query [, resource $link_identifier])
```

where `link_identifier` identifies the connection established with MySQL. If the link identifier is not specified, the last link opened by `mysql_connect()` is assumed. If no such link is found, it tries to create one by executing `mysql_connect()` with no arguments. An error is displayed if a connection is not established.

The returned result resource may be passed to `mysql_fetch_array()`, to display fetched records one by one.

Example:

```
$result = mysql_query($query) or die(mysql_error());
```

The `$result` returned by `mysql_query()` contains all the records fetched from the table on the basis of the specified query and by passing this array of records `$result` to `mysql_fetch_array()`, we can extract one record at a time from it.

4.1.4 $_POST()

The `$_POST` array is an array which collects the values sent from a form using HTTP POST method. When the user clicks the `"Submit"` button, the value of the `$_POST["variable_name"]` will be automatically set by PHP where `variable_name` is the id of the input HTML tag used on the form. The destination PHP script can use the `$_POST` array to retrieve the form data with the help of the following code:

```
local_variable=$_POST["variable_name"];
```

Hence `local_variable` will contain the information passed by another web form.

4.1.5 mysql_fetch_array()

This function is used for fetching one record at a time from the array of records (produced as a result of a query). It gets one row from the recordset and returns `true`, and `false` when there are no more rows left in the recordset. It returns a row from a recordset as an associative array or a numeric array.

Syntax:

```
row=mysql_fetch_array(data,array_type)
```

where the data pointer is the result from the `mysql_query()` function and

array_type is optional that specifies what kind of array to return. Its value can be:

MYSQL_ASSOC -	Associative array
MYSQL_NUM -	Numeric array
MYSQL_BOTH -	Default. Both associative and numeric array

Note: After the data is retrieved, this function moves to the next row in the recordset. Each subsequent call to `mysql_fetch_array()` returns the next row in the recordset.

Example:

```
$row = mysql_fetch_array($results)
```

One record is extracted from the recordset `$results` and is stored in variable: `$row`

4.1.6 Extract()

This function is used for extracting all the variables stored in the specified array (or record).

Syntax:

```
extract(array name)
```

Example:

```
extract($row);
```

If `$row` contains a record (row) of a table, say `customers`, with two fields say `userid` and `address`, then by the above `extract()` function, two variables `$userid` and `$address` will be generated that will contain the information of `userid` and `address` in that particular row (record).

4.1.7 mysql_close()

This is a very important command as it closes the connection to the database server. It is a better idea to close the connection with the server when all the operations are performed on the database or else it may diminish the efficiency of the web host.

4.1.8 mysql_numrows()

To know how many rows there are in the resultset, we use this function. Recall that we get the rows extracted from the table and stored in the form of resultset when an SQL query consisting of "`select`" clause is executed. Thus this function counts how many rows there are in the given resultset.

Syntax:

```
mysql_numrows(resultset name);
```

Example:

```
$num=mysql_numrows($result);
```

This will set the value of `$num` to be the number of rows stored in `$result` (the array where the records retrieved from the table are stored on execution of the SQL statement with select clause).

4.2 CREATING A TABLE IN PHP

Creating a table in a MySQL database from PHP script requires the following steps:

1. Establishing connection with MySQL server
2. Creating a database (if it does not exist). If database already exists then skip this step
3. Selecting the database (in which table is to be created)
4. Writing the SQL statement for creating a table
5. Executing the SQL statement

To understand the above steps let us look at the following program that creates a table named `products` in the `shopping` database. The products table that we are going to create will have five fields (columns): `id`, `item_code`, `item_name`, `quantity` and `price`. The fields `id`, `item_code` and `quantity` are considered to be of `int` (integer) data type, `item_name` of `character` data type and `price` of `float` data type.

createtable_products.php

```
<?php
$user="username";
$password="password";
$connect = mysql_connect("localhost", $user, $password) or die ("Please
check your server connection.");

//create the database if it doesn't already exist
$create = mysql_query("CREATE DATABASE IF NOT EXISTS shopping") or
die(mysql_error());
mysql_select_db("shopping");

$sql = "CREATE TABLE products(
   id int(6) NOT NULL auto_increment,
   item_code int(4),
   item_name varchar(50),
```

```
        quantity int(4),
        price float)";
        $res = mysql_query($sql) or die (mysql_error());
    echo "products table successfully created!";
?>
```

The above program has all the five steps of creating a table. In the first two lines, two variables `$user` and `$password` are initialised to the username and password of the MySQL server respectively. From the third line onwards, we see that the five steps for creating a table are executed. Let us see look at each statement step by step:

1. Establishing connection with MySQL server

Before proceeding with the creation of a database or a table, first we need to connect to the MySQL server. The connection is established with the help of a valid user name and password.

```
$connect = mysql_connect("localhost", $user, $password) or die ("Please check your server connection.");
```

In the above command `localhost` signifies that the MySQL server is installed on the local machine. The string `localhost` will be replaced by the IP address of the server or server name (Example: sqlserver.com or 216.237.120.79) if we try to connect to the remote server. The `$user` and `$password` variables contain the valid userid and password supplied by the administrator. The keyword `die` is for displaying error messages if any information supplied is wrong.

2. Creating a database

After we are connected to the MySQL server, we must then create the database (if it does not exist) followed by selecting it i.e. making it active in memory. The command for creating database is:

```
$create = mysql_query("CREATE DATABASE IF NOT EXISTS shopping") or die(mysql_error());
```

The above statement creates a database named `"shopping"` if it doesn't already exist. If the database is already there, this command won't do anything.

3. Selecting the database

Selecting the database means loading the database in memory (i.e. making it active). The command for selecting a given database is:

```
mysql_select_db("shopping");
```

The above command selects the database `shopping`. Now all SQL statements that are executed will be applied to the tables belonging to this (`shopping`) active database.

4. Writing the SQL statement for creating the table

The SQL statement given below creates a table named `products` that has five fields out of which `id` field is set to `auto_increment` i.e. its value will automatically increase by 1 with every record inserted.

```
$sql = "CREATE TABLE products(
id int(6) NOT NULL auto_increment,
item_code int(4),
item_name varchar(50),
quantity int(4),
price float)";
```

The rest of the fields (columns) in the table are `item_code` that will store an integer values of up to four digits, `item_name` field will store the name of the item of up to 50 characters long and `quantity` field can store an integer value of up to four digits and `price` field can store a fractional value (value with decimal points).

5. Executing the SQL statement

The statement given below executes the above SQL statement of creating the products table

```
$res = mysql_query($sql) or die (mysql_error());
```

The `$res` variable is for storing the result of the execution of the SQL statement.

4.3 GETTING INFORMATION FROM THE USER

In every website, getting information from the visitor is an essential requirement. Either in the form of a Sign Up form, Order form or a Feedback form, we gather information from the visitor. Getting information from the visitor requires the following steps:

1. Providing the text boxes, radio buttons, checkboxes etc. to the visitor to enter information
2. Submitting the information for some action. The action may be either storing the information in the tables, fetching information from the tables or updating information in the tables

Consider the following program, where we provide two textboxes to the visitor to fill in `userid` and `address`. After filling up the information, the visitor is supposed to select `submit` button to proceed with necessary action.

enterinfo.php

```
<html>
   <head>
```

```
    </head>
    <body>
        <form action="insertrec.php" method="post">
            <table border="0" cellspacing="1" cellpadding="3">
                <tr><td>Userid: </td><td> <input size="20" type="text"
                    name="userid"></td></tr>
                <tr><td>Address: </td><td> <input size="80" type="text"
                    name="add"></td></tr>
                <tr><td><input type="submit" name="submit" value="Submit">
                    </td><td><input type="reset" value="Cancel"></td></tr>
            </table>
        </form>
    </body>
</html>
```

Explanation

`<form action="insertrec.php" method="post">`

The above statement means that when the `Submit` button is pressed, we will be navigated to the file: `insertrec.php` and the HTTP Request method that is used for passing the information entered in the current file is `Post` method.

In other words, when the user clicks the "Submit" button, the `$_POST["userid"]` and `$_POST["add"]` variables will be automatically initialised to the data entered by the user. The information in these variables can be retrieved in the `insertrec.php` file to physically store it in the table.

`<table border="0" cellspacing="1" cellpadding="3">`

This statement defines a table with the specified cell spacing and cell padding. Tables are usually used for the purpose of aligning labels and textboxes on the web form.

```
<tr><td>Userid: </td><td> <input size="20" type="text"
            name="userid"></td></tr>
<tr><td>Address: </td><td> <input size="80" type="text"
            name="add"></td></tr>
```

The user is provided with two textboxes to enter the information (we can use other controls like checkboxes, radio buttons, drop down lists etc. to get information from the user). The information entered by the user will be automatically stored in `$_POST["userid"]` and `$_POST["add"]` variables.

```
<tr><td><input type="submit" name="submit" value="Submit">
```

This will navigate us to another PHP web form: `insertrec.php` file.

The conclusion is that the information filled up by the user is stored in `$_POST["userid"]` and `$_POST["add"]` variables and is sent to `insertrec.php` file for further action.

4.4 INSERTING RECORDS IN THE TABLE

The information sent by the user via enterinfo.php program: userid and add (for address) is passed to insertrec.php program (as declared by the statement: <form action="insertrec.php" method="post">). In the insertrec.php program we will be storing the received information in a table. Insertion of records in a table through PHP program requires the following steps:

1. Establishing connection with the MySQL server
2. Selecting the database containing the table in which record is supposed to be inserted
3. Collecting information to be stored in the table
4. Writing the SQL statement for inserting the record
5. Executing the SQL statement

Let us observe the following PHP program where a step-by-step approach is taken for inserting the information (`userid` and `add`) passed from the previous PHP program (`enterinfo.php`) into a table named `customers`. The program is as shown below:

insertrec.php

```
<?php
    connect = mysql_connect("localhost", "root", "mce") or die ("Please,
        check the server connection.");
    mysql_select_db("shopping");
    userid = $_POST['userid'];
    add = $_POST['add'];

    $sql = "INSERT INTO customers (userid, address) VALUES ('$userid',
    '$add')";
    $result = mysql_query($sql) or die(mysql_error());
    echo "Record is saved";
?>
```

Explanation

The five steps of inserting records can be seen:

1. Establishing connection with MySQL server

```
connect = mysql_connect("localhost", "root", "mce") or die ("Please,
    check the server connection.");
```

The above statement establishes the connection with the local MySQL server with the `userid` as `"root"` and the `password` of the root is assumed to be `"mce"` (any text).

2. Selecting the database

```
mysql_select_db("shopping");
```

This statement selects the database "`shopping`" (i.e. loads it in memory) so that we can insert record in its `"customers"` table (We assume that a `customers` table exists in the shopping table and the table has two fields (columns): `userid` and `address`).

3. Collecting information to be stored in the table

```
userid = $_POST['userid'];
add = $_POST['add'];
```

These two statements are for collecting information from the `$_POST` array (containing the information – `userid` and `add` passed by the `enterinfo.php` program).

4. Writing SQL statement for inserting the record

```
$sql = "INSERT INTO customers (userid, address) VALUES ('$userid', '$add')";
```

This is the SQL statement for inserting the record into `customers` table of `shopping` database. It is a simple SQL statement where `$userid` and `$add` represent the information entered by the visitor and are enclosed in single quotes as both contain text matter.

5. Executing the SQL statement

```
$result = mysql_query($sql) or die(mysql_error());
```

This statement executes the SQL statement stored in `$sql` variable and the result of the execution (whether record insertion was successful or failure) is stored in `$result` variable.

Note: Inserting records in a table requires two PHP programs; one gathers information from the visitor and passes it to the other program, while the second program processes the information received from the earlier program. Processing here means any task such as inserting the information into the table, searching information from the table, deleting information from the table etc.

It also means that the process of searching records from the table (Section 4.6) and deleting records from the table (in Section 4.7) will also be requiring two programs, one for gathering information and the other for processing information.

4.5 RETRIEVING ALL RECORDS FROM THE TABLE

The information stored in the back-end database tables is often required to be retrieved for displaying it to visitors to our website. Good examples of records retrieval include the list of the products for sale on a website, display of facilities provided on different plans (PostPaid or Prepaid) on a telecommunication website or display of examination results on a university website etc.

Retrieving all records from the table requires following five steps:

1. Establishing a connection with the MySQL server
2. Selecting the database containing the table whose records have to be retrieved
3. Writing the SQL statement for selecting the desired fields from the table
4. Executing the SQL statement and storing the fetched records in a resultset
5. Fetching one row at a time from the result set and displaying it one by one

Let us look at the following program that fetches all the records from the `customers` table of `shopping` database and displays them on the screen in tabular format. All the steps followed in retrieval of records can be clearly seen in the program below:

customers_list.php

```
<?php
   $connect = mysql_connect("localhost", "root", "mce")
      or die("Please, check your server connection.");
   mysql_select_db("shopping");
   $query = "SELECT userid, address from customers ";
   $results = mysql_query($query) or die(mysql_error());
   echo "<table border=\"1\">\n";
 echo "<th>Userid</th><th>Address</th>\n";
 while ($row = mysql_fetch_array($results)) {
   extract($row);
   echo  "<tr>";
   echo  "<td>";
   echo  $userid;
   echo  "</td>";
   echo  "<td>";
   echo  $address;
   echo  "</td>";
   echo  "</tr>\n";
 }
```

```
    echo  "</table>\n";
?>
```

Explanation

The five steps of retrieving all records from the table are clearly visible:

1. Establishing a connection with the MySQL server

```
$connect = mysql_connect("localhost", "root", "mce")
or die("Please, check your server connection.");
```

This statement establishes the connection with the local MySQL server with the userid as "root" and the password of the root is assumed to be "mce" (any text).

2. Selecting the database

```
mysql_select_db("shopping");
```

This statement selects the database "shopping" so that we can retrieve records from its customers table. (We assume that a customers table exists in the shopping database and it consists of two fields (columns): userid and address).

3. Writing the SQL statement for selecting the desired fields

```
$query = "SELECT userid, address from customers ";
```

This is the SQL statement that will retrieve userid and address fields (columns) from the customers table.

4. Executing the SQL statement

```
$results = mysql_query($query) or die(mysql_error());
```

This statement executes the SQL statement, and all the fetched records (rows) from the customers table are stored temporarily in the resultset: $results. The resultset is a sort of array where rows of the tables are fetched and temporarily stored.

5. Fetching one row at a time from the result set

```
while ($row = mysql_fetch_array($results)) {
```

The mysql_fetch_array() is a function used for extracting one record (row) at a time from the array of records - $results (resultset). In the above statement, we are using a while loop for extracting one row at a time from the resultset and storing it in a variable $row. When all rows from the resultset are extracted (i.e. when there are no rows left in the $results array),

`mysql_fetch_array()` function will return `false`, hence will come out of the `while` loop. The row returned by the `mysql_fetch_array()` from the resultset will be in a form of an associative array or a numeric array. In other words, the `$row` that will contain one row (record) at a time is an associative array or a numeric array.

In the `while` loop, we use extract function to extract the fields (columns) from array `$row` that is assumed to contain one row at a time of the table:

```
extract($row);
```

The extract function extracts all the fields (columns) stored in the specified array. After getting all the fields extracted from `$row` array, they are displayed one by one with the help of the following statements:

```
echo "<tr>";
echo "<td>";
echo $userid;
echo "</td>";
echo "<td>";
echo $address;
echo "</td>";
echo "</tr>\n";
```

The `$userid` and `$address` are the variables or fields that are extracted from `$row` array (by using `extract()` function) which contain the information of the userid and address.

4.6 SEARCHING RECORDS FROM THE TABLE

Searching is the most popular and frequent activity among visitors to a website. Visitors enter certain keywords (text) related to the product, service etc. which they are interested in and submit it. That keyword is searched for in the database tables and the matching rows (records) are displayed on the screen. A search engine is the most common example of searching records as here we specify the keyword related to the object whose information we want to look at. On submission of keyword, we get the desired information.

Let us make a searching program where a visitor enters the `userid` of the user whose information (address) he wants to access. The specified userid will be then searched in the `customers` table of `shopping` database and the information (address) of the matching record is displayed on the screen.

As said earlier, the tasks of insertion, searching, deletion etc. in the database require two PHP programs, one gathers information from the visitor and passes on to the next program which subsequently takes the necessary action.

Let us name the first PHP program which will prompt the visitor to enter `userid` (whose information has to be searched) as `search.php` and its code may appear as shown below:

search.php

```
<html>
   <head>
   </head>
   <body>
      <form action="searchcustomer.php" method="post">
         <table border="0" cellspacing="1" cellpadding="3">
            <tr><td>Userid: </td><td> <input size="20" type="text"
               name="userid"></td></tr>
            <tr><td><input type="submit" name="submit" value="Submit">
               </td><td><input type="reset" value="Cancel"></td></tr>
         </table>
      </form>
   </body>
</html>
```

We can see that in the above program, the `userid` entered by the visitor will be passed to the PHP program named: `searchcustomer.php` that will search the information of the user from the tables whose userid is passed. Searching records from the table requires the following steps:

1. Establishing a connection with the MySQL server
2. Selecting the database containing the table in which the desired record is supposed to be searched
3. Getting the criteria i.e. the condition to retrieve the record
4. Writing the SQL statement for searching the record
5. Executing the SQL statement and retrieving the desired record in the resultset
6. Fetching one row at a time from the result set and displaying it one by one

Let us study the following `searchcustomer.php` program that fetches the `userid` sent by `search.php` program and searches the `customers` table of `shopping` database and retrieves the desired information (address) of the `userid` supplied and displays it on the screen.

searchcustomer.php

```
<?php
   $connect = mysql_connect("localhost", "root", "mce")
```

```
    or die("Please, check your server connection.");
    mysql_select_db("shopping");
    $uid=$_POST['userid'];
    $query = "SELECT userid, address from customers where userid like
    '$uid'";
    $results = mysql_query($query) or die(mysql_error());
    echo "<table border=\"1\">\n";
    echo "<th>Userid</th><th>Address</th>\n";
    while ($row = mysql_fetch_array($results)) {
        extract($row);
        echo "<tr>";
        echo "<td>";
        echo $userid;
        echo "</td>";
        echo "<td>";
        echo $address;
        echo "</td>";
        echo "</tr>\n";
    }
    echo "</table>\n";
?>
```

Explanation

The six steps of searching the desired record from the table can be easily seen in the above program:

1. Establishing a connection with the MySQL server

```
$connect = mysql_connect("localhost", "root", "mce")
    or die("Please, check your server connection.");
```

This statement establishes the connection with the local MySQL server with the `userid` as `"root"` and the password as `"mce"`.

2. Selecting the database containing the table to be searched

```
mysql_select_db("shopping");
```

This statement selects the database "shopping" so that we can retrieve the desired record from its customers table.

3. Getting the criteria i.e. the condition to retrieve the record

```
$uid=$_POST['userid'];
```

As we know the current program is invoked by the earlier program search.php where the visitor was prompted to enter the userid of the person whose information is required. In search.php program, after entering the userid of the person, when Submit button is clicked, the current PHP program: searchuser.php will be invoked and here, we retrieve that userid (entered in the search.php program) through $_POST array and store it in a local variable: $uid.

4. Writing the SQL statement for searching the record

```
$query = "SELECT userid, address from customers where userid like '$uid'";
```

This is the SQL statement that retrieves the userid and address fields from the customers table whose userid is supplied from search.php file.

5. Executing the SQL statement and retrieving the desired record in the resultset

```
$results = mysql_query($query) or die(mysql_error());
```

Executing the SQL statement and storing the fetched records from the table in the resultset: $results

6. Fetching one row at a time from the result set and displaying it one by one

```
while ($row = mysql_fetch_array($results)) {
    extract($row);
    echo "<tr>";
    echo "<td>";
    echo $userid;
    echo "</td>";
    echo "<td>";
    echo $address;
    echo "</td>";
    echo "</tr>\n";
}
```

The mysql_fetch_array() is a function used for fetching one record at a time from the array of records ($results). It gets one row from the resultset and stores it in the variable $row.

The `extract()` function is used for extracting all the variables i.e. fields or columns stored in the `$row` array. The `$row` is an array that contains one record of the customer and the elements of this array are the fields (columns) of the record. The extracted variables `$userid` and `$address` (from `$row`) are displayed one by one with the help of `while` loop.

4.7 DELETING RECORDS FROM THE TABLE

In the task of deleting a record from the database table, again we will require two PHP programs; one of them will prompt the visitor to enter the `userid` of the user to be deleted. After entering the `userid`, when the visitor clicks the Submit button, that `userid` will be passed to another PHP program which then deletes the record from the `customers` table that contains the supplied `userid`. There is no need of making the first PHP program (that asks the visitor to enter `userid`) from scratch as we can use the same one that we used in searching program (Section 4.6), i.e. `search.php`. The only change we need to do in that program is that we replace the following statement:

```
<form action="searchcustomer.php" method="post">
```

with the below statement:

```
<form action="deletecustomer.php" method="post">
```

because this time, we want the `userid` to be passed to `deletecustomer.php` program instead of `searchcustomer.php` program.

The task of deleting records from the table requires the following steps:

1. Establishing a connection with the MySQL server
2. Selecting the database containing the table from where we want to delete the record(s)
3. Getting the criteria i.e. the condition to delete the record
4. Writing the SQL statement for deleting the record(s)
5. Executing the SQL statement to physically delete the record(s) from the table

Let us study the following `deleteuser.php` program that fetches the `userid` sent by `search.php` program and deletes the record with the matching `userid` from the `customers` table of `shopping` database and displays a message: "Record successfully deleted" on successful deletion of record.

deleteuser.php

```
<?php
   $connect = mysql_connect("localhost", "root", "mce")
      or die("Please, check your server connection.");
   mysql_select_db("shopping");
   $uid=$_POST['userid'];
```

```
    $sql = "DELETE from customers where userid like '$uid'";
    $res = mysql_query($sql) or die (mysql_error());
    echo "Record successfully deleted";
?>
```

Explanation

The five steps of deleting the desired record(s) from the table can be seen:

1. Establishing a connection with the MySQL server

```
$connect = mysql_connect("localhost", "root", "mce")
or die("Please, check your server connection.");
```

This statement establishes the connection with the local MySQL server with the `userid` as `"root"` and the password as `"mce"`.

2. Selecting the database containing the table

```
mysql_select_db("shopping");
```

This statement selects the database `"shopping"` so that we can delete the desired record from its `customers` table.

3. Getting the criteria i.e. the condition to delete the record

```
$uid=$_POST['userid'];
```

We know that the current program `deleteuser.php` is invoked by an earlier program named `search.php` where the user was asked to enter the `userid` of the person whose information has to be erased. The `userid` entered in that program is passed to the current program and here, we retrieve it through `$_POST` array and store it in a local variable: `$uid`.

4. Writing the SQL statement for deleting the record(s)

```
$sql = "DELETE from customers where userid like '$uid'";
```

This is the SQL statement that deletes the record from the `customers` table whose `userid` is supplied from `search.php` file.

5. Executing the SQL statement to delete the record(s)

```
$res = mysql_query($sql) or die (mysql_error());
```

Executing the SQL statement that physically deletes the record from the `customers` table of the `shopping` database.

SUMMARY

In this chapter, we have seen different functions that are used for accessing MySQL tables from PHP programs. Then using these functions we have seen all the modules that are essential in making a complete website. We also saw the statements used in these modules along with their meanings. The modules that we saw in this chapter were: Creation of table, Getting information from the user, Inserting records in the table, Retrieving all records from the table, Searching desired records from the table and Deleting records from the table.

In the next chapter, we will see the sample output of our Shopping Cart Project. We will see step-by-step the different outputs that will appear on selecting different links and buttons on our website. Also we will see how different items can be selected in the Shopping Cart, how Shopping Cart is updated and finally how an order is placed on our website.

5

AJAX

In this chapter we will look at:

- AJAX basics
- Limitations of traditional web applications
- Items for implementing AJAX

5.1 AJAX BASICS

Ajax stands for Asynchronous JavaScript and XML. Usually in a traditional web application, when we want to access certain data from the database on the server, an HTTP request from the client is made to the server either by GET or POST method. After receiving the data from the server, the web page needs to be *reloaded* to show the fetched data. In AJAX technology we can request and receive the data from the server in the background and can display it on the page without a reload.

With AJAX, JavaScript communicates directly with the server, through the JavaScript XMLHttpRequest object (XML over HTTP) and it is with the help of this object that a web page can make a request to, and get a response from a web server without reloading the page.

A traditional web page takes a longer time to get the desired results because of the *round trip*: all the information entered by the user on the form is sent from the client to the web server. The web server processes the data and the desired information is sent back to the client. Even if small changes are made in the form, all the data on the form is sent to the web server and the entire page is refreshed. In a traditional web application, we don't have a facility to refresh only a small portion of the web page; instead the complete page is refreshed, which is very time consuming

In Figure 5.1, we can clearly see the three steps of communication performed in the traditional web application model:

The client makes an HTTP request to the web server.

The web server searches for the desired data from the database and

The fetched data (from the database) is sent back (postback) to the client (the whole page is reloaded with the new information).

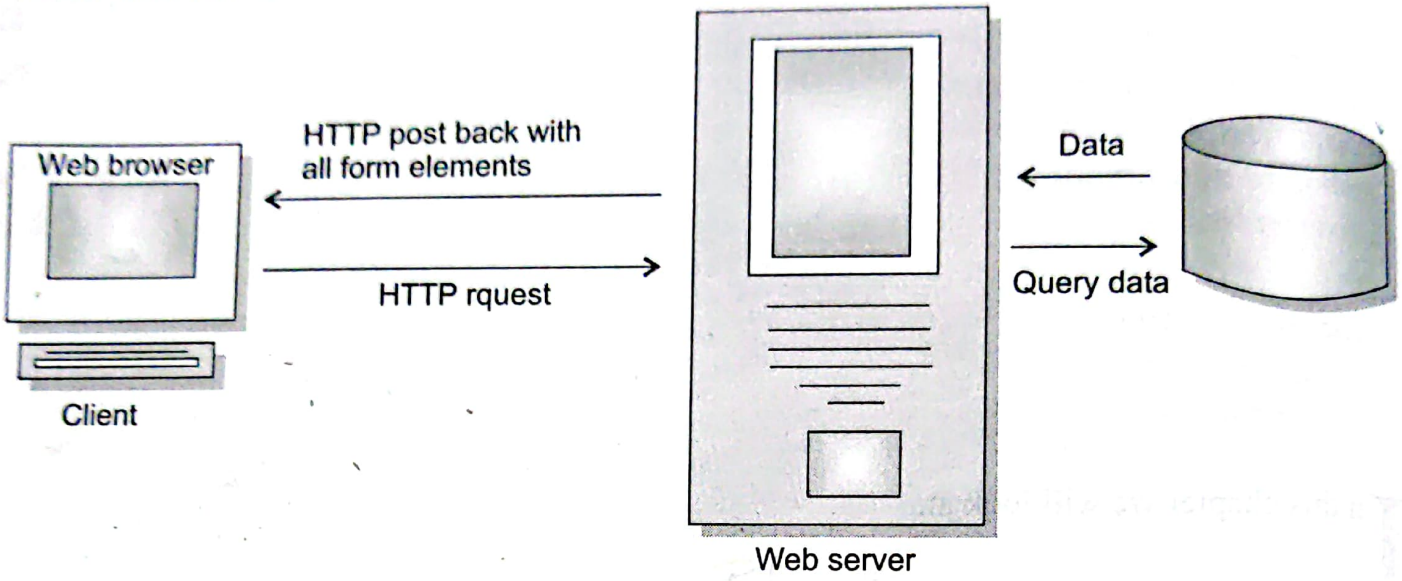


Figure 5.1 *Traditional Web Application Model*

We face the following limitations in a conventional web application.

5.2 LIMITATIONS OF TRADITIONAL WEB APPLICATIONS

- All the data of the form is sent to the web server—even if small changes are made
- Results in network congestion because of large amount of data transferred during a postback
- Until and unless the user clicks a button or another control that posts data back to the server, no result will be displayed

In AJAX, A stands for *Asynchronous* and it means getting server response without refreshing the whole page. We can even update a portion of a web page with this technology. Let's try for a deeper understanding of what AJAX consists of.

With AJAX all the above limitations of traditional web applications are removed as it consists of following items:

- XmlHttpRequest
- JavaScript
- DOM (Document Object Model)
- CSS (Cascading Style Sheets)

5.3 ITEMS FOR IMPLEMENTING AJAX

5.3.1 XmlHttpRequest

It is this object that is used to talk to the server asynchronously, meaning it allows the browser to talk to the server without requiring a postback of the entire web page. The way this capability is provided varies from browser to browser. For example, in Internet Explorer, this capability is provided by the MSXML ActiveX component. With Mozilla Firefox and other web browsers, this capability is provided by the object called XmlHttpRequest.

5.3.2 JavaScript

Ajax applications use JavaScript code for the following reasons:

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

Note: JavaScript is the client side scripting language which is supported by all major browsers.

5.3.3 DOM (Document Object Model)

It provides a tree-like structure to a web page as a set of programmable objects which can be manipulated using JavaScript code. It allows dynamic updating of a part of a web page.

5.3.4 CSS (Cascading Style Sheets)

It is a centralised way of defining all the styles to be applied to different elements of a web page at one place. It makes web applications appear more consistent and attractive. CSS styles are implemented by defining style rules in a separate document which is then referred to by the web page where styles have to be applied.

In Figure 5.2, we can clearly see that XMLHttpRequest plays a major role in performing asynchronous request in the Ajax web application model:

- The client makes an XMLHTTP Request object to communicate with the web server (fetching data from server without postback). The client may use:
- JavaScript to automate validation or navigation tasks
- CSS for applying styles uniformly
- DOM to update a part of the web page
- The Web Server searches for the desired data from the database and

- The fetched data (it may constitute a part of the whole page) is sent back to the client so as to update a portion of the web page without delay.

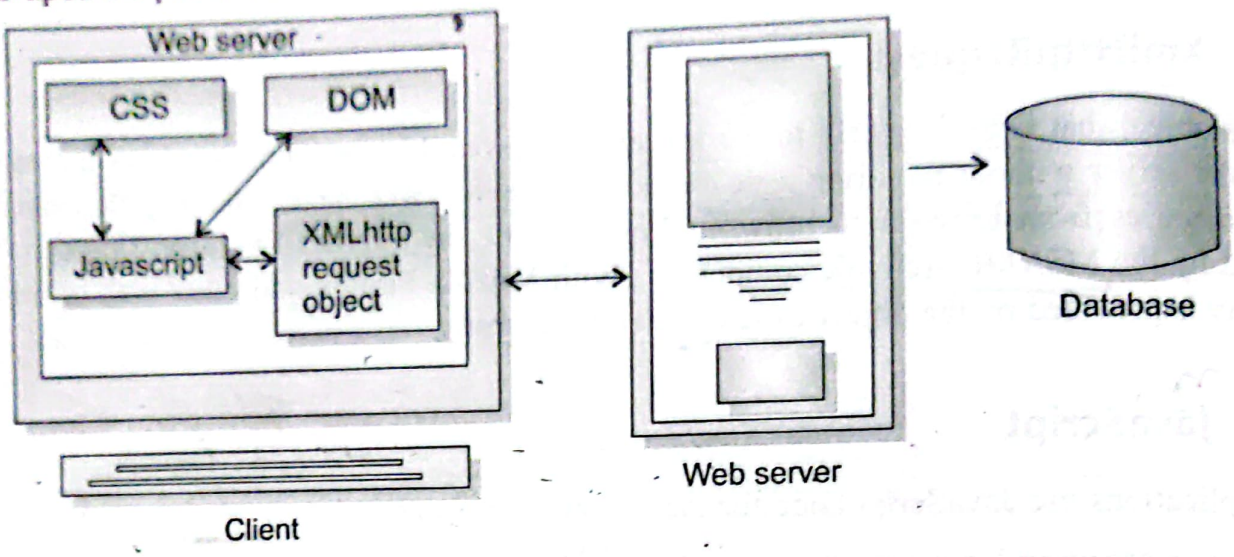


Figure 5.2 *Ajax Web Application Model*

So, it is clear that Ajax applications make use of XmlHttpRequest object for talking to the server asynchronously and to retrieve only the data that is needed (while the user is working). And on the client side, JavaScript plays a major role as it processes the server's response and modifies the document contents by manipulating its DOM to specify that action is completed. The job of CSS is to provide a consistent look and feel to the web application.

Note: The data are mostly communicated to the web server in XML format but it may also be in JSON (JavaScript Object Notation) format (we will see it in Chapter 8).

SUMMARY

In this chapter we learnt that the main components that make up AJAX are Asynchronous JavaScript and XML and it is a technique which makes a web page highly responsive. It makes use of a XMLHttpRequest object with the help of which a web page can make a request to, and get a response from a web server without reloading the page. Also in this technique, even a small portion of a web page can be refreshed instead of the entire page. We also saw the limitation of the traditional web page in this chapter. Finally we had an introduction to all the four items used in implementing AJAX: XmlHttpRequest, JavaScript, DOM and CSS.

In the next chapter, we will see the practical demonstration of AJAX with the help of running examples. We will learn the basics of JavaScript, role of DOM in accessing form elements, the way in which an asynchronous request is placed to the server, handling of Key Events, using JavaScript in accessing form elements, performing XMLHttp Post Requests, separating JavaScript code from the PHP program, specifying our own function in JavaScript file, using CSS (cascading style sheets), sending data selected from the combo box, items selected from a list box, items selected from radio and checkboxes to the server asynchronously. Finally, we will also see how AJAX, PHP and MySQL all combine for accessing a database.

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:

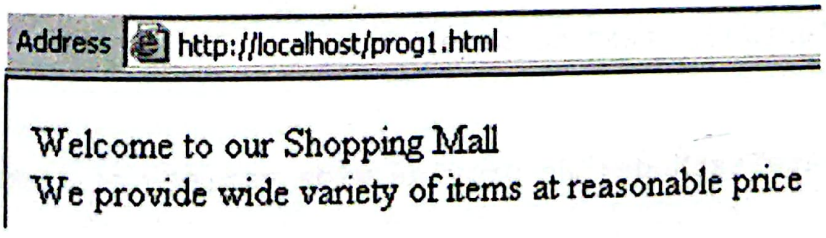


Figure 6.1 *Output of the Program Prog1.html*

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:

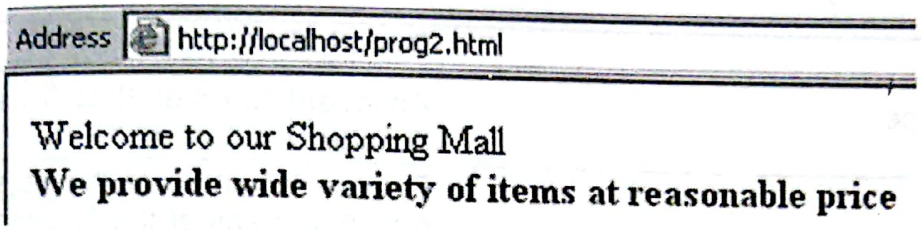


Figure 6.2 *Output of the Program Prog2.html*

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. ActiveXObject, 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.

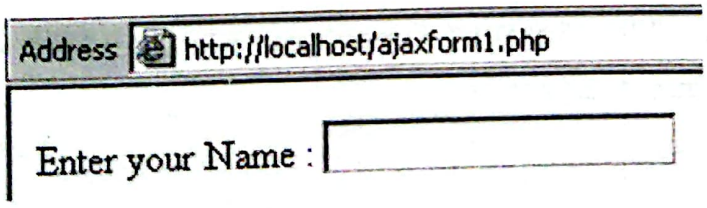


Figure 6.3 *Screen Displaying a Textbox to Enter Name*

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

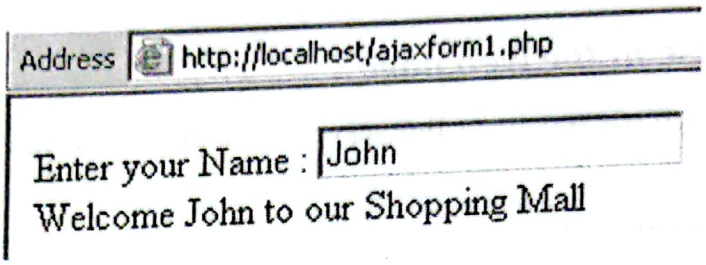


Figure 6.4 ***As Soon as the User Finishes Typing, a Message is Displayed on the Screen***

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:

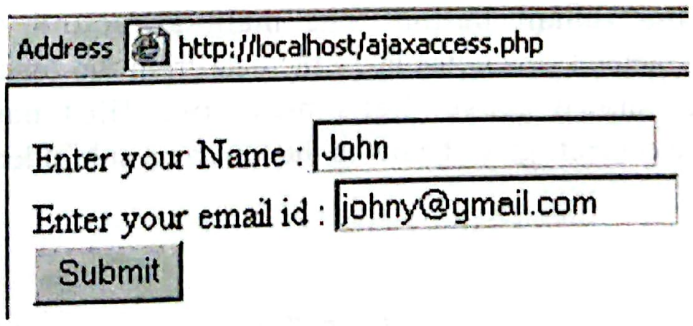


Figure 6.5 ***Screen to Enter Name and Email Id***

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:

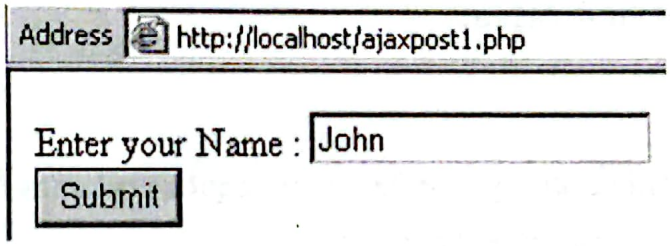


Figure 6.7 *Enter the Name*

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

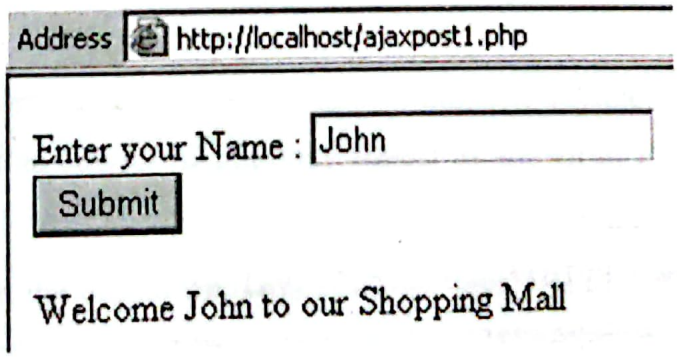


Figure 6.8 *Server Response is Displayed*

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 (Server)

```
<?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).

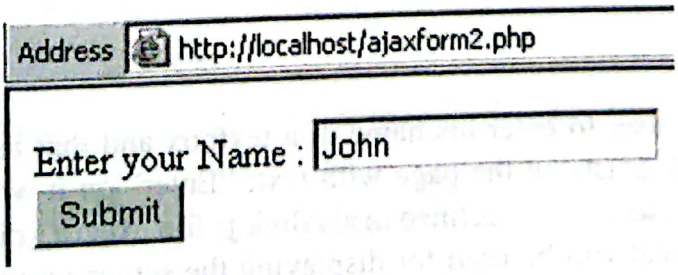


Figure 6.9 *Enter the User Name*

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

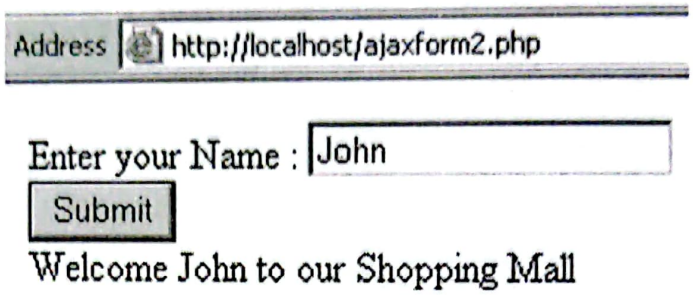


Figure 6.10 *Server Response is Displayed (no Change in the Output)*

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).

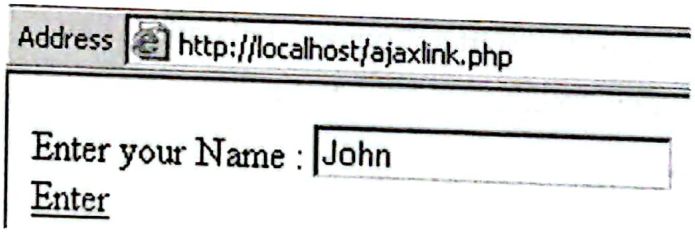


Figure 6.11 *Enter the User Name*

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

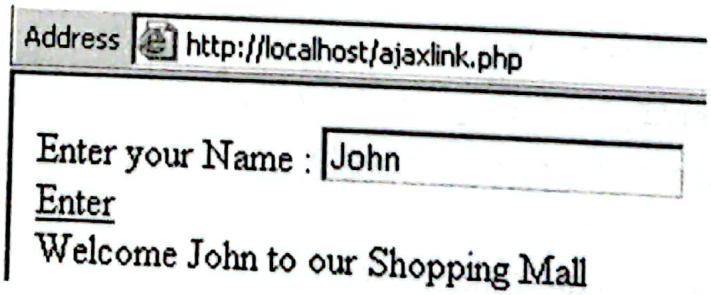


Figure 6.12 *Welcome Message is Displayed*

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;
                    }
            }
    }
```

6.9 CONVERTING A STRING INTO UPPERCASE

The example below asks the user to enter a string which is converted into uppercase asynchronously and is displayed to the client. Let's make three files ajaxcaps.php, ajaxcaps.js and convertcaps.php with the following contents in the `htdocs` subfolder of the `apache` directory.

ajaxcaps.php

```
<html>
<head>
<script language="javascript" type="text/javascript" src="ajaxcaps.js">
</script>
</head>
<body>
Enter text: <input type="text" onblur="showdata();" name="inputname"
id="inputname" /> <br>
Text in Capital is: <input type="text" name="outputname" id="outputname" />
</body>
</html>
```

A JavaScript file ajaxcaps.js is imported into the current web page so that the methods defined in it can be accessed from here. The user is prompted to enter a text in one textbox. After feeding the text, as tab key is pressed i.e. the moment focus to the textbox is lost, `showdata()` method is invoked (which will convert the text to upper case and display it in the second textbox).

ajaxcaps.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();

xmlhttp.open("GET",
"convertcaps.php?inputname="+document.getElementById('inputname').value, true);
   xmlhttp.onreadystatechange=function() {
   if (xmlhttp.readyState==4) {
document.getElementById('outputname').value = xmlhttp.responseText;
}
}
xmlhttp.send(null);
}
```

This program is first making an XMLHttpRequest object and then making a request to the server for the file convertcaps.php, sending the text entered by the user to it. The text entered by the user is stored in textbox with name "inputname". The requested file convertcaps.php converts the text to uppercase and is the file which generates a server response. The response generated by the server (text in upper case) is retrieved in the form of text and is assigned to the textbox with name:

"outputname" ("outputname" is the second textbox made on the web page to display server response). The statement: `xmlhttp.send(null);` is to actually send the request to the server. Since we don't want to pass any query string to the requested file convertcaps.php, the argument passed to this method is null.

convertcaps.php

```
<?php
if (isset($_GET['inputname']))
echo strtoupper($_GET['inputname']);
?>
```

This program is retrieving the text sent from the client (in the inputname variable) and returning it to the client after converting it into upper case (Figure 6.13).

Output:

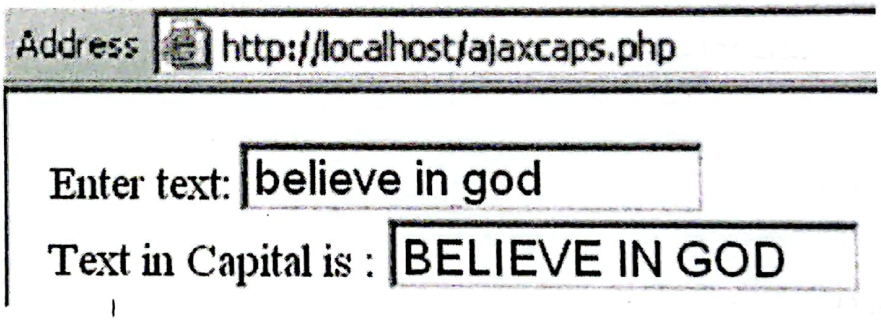


Figure 6.13 *Output of the Program ajaxcaps.php*

6.10 CSS (CASCADING STYLE SHEETS)

Using CSS styles is considered as the consistent method of setting the styles of the elements on a web page. All the font styles and their appearance that we want to apply to different elements on a web page are stored in a separate file with extension .css which is then referenced in the web page. The style sheet contains different selectors each with different font size, colours etc. To apply the characteristics of the CSS selector on any web page element, that element has to specify the particular CSS selector as its class.

Since in CSS method, the information is separated from presentation of the information i.e. all the formatting attributes are at one place, any changes if desired in the format can also be easily applied.

Let's use CSS style sheet in an example. The following example asks the user to enter his name and the moment user presses the tab key or clicks somewhere out of the textbox, a welcome message will be displayed to him (as is done in earlier programs). But here, the welcome message will be applied some characteristics like fonts, colours etc. Let's make three files named styles1.css, phpajaxstyle1.php and phpajaxstyle1.js with the contents below in the `htdocs` subfolder of the `apache` directory. For the server response, we need the file ajaxdata1.php (made earlier).

styles1.css

```
.name
{
   font-family: Verdana;
   font-weight:bold;
   background-color: Cyan;
   font-size: 20pt;
}
```

This code is defining a CSS selector named `name` which contains font Verdana of size 20 points which will appear in bold and its background colour will be cyan. We can have any number of selectors in a style sheet.

The CSS selectors are mapped to the HTML via the class attribute. That is, any element in the HTML document that specifies any selector as its class will inherit all the characteristics specified in that selector.

phpajaxstyle1.php

```
<html>
   <head>
      <script type="text/JavaScript" src="phpajaxstyle1.js"></script>
      <link href="styles1.css" type="text/css" rel="stylesheet"/>
   </head>
      <body>
         Enter your Name: <input type="text" onblur="showdata(this.value)"/>
<div class="name" id="info"></div>
</body>
</html>
```

As we can see in the above program, we are referencing the CSS style sheet styles1.css (so as to use the characteristics of the CSS selectors defined in it).

Since the element specifies the selector name (defined in CSS style sheet) as its class, it will inherit its characteristics i.e. the text within the tag will have the characteristics of the CSS selector named `name` i.e. it will appear in Verdana font of size 20 points in bold and its background colour will be cyan.

phpajaxstyle1.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(user)
{
var xmlhttp=makeRequestObject();
            var file = 'ajaxdata1.php?usernme=';
          xmlhttp.open('GET', file + user, true);
          xmlhttp.onreadystatechange=function() {
            if (xmlhttp.readyState==4) {
                var content = xmlhttp.responseText;
                if( content ){
                      document.getElementById('info').innerHTML = content;
                }
            }
            }
            xmlhttp.send(null)
}
```

This program is doing the same thing, that is making an XMLHttpRequest object and 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 the server response. The div element with id `info` is located in the web page and the server response is pasted at that element. The server response will appear in the style defined by the CSS selector `name`.

ajaxdata1.php

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

The output of the program phpajaxstyle1.php may appear as shown in Figure 6.14.

Output:

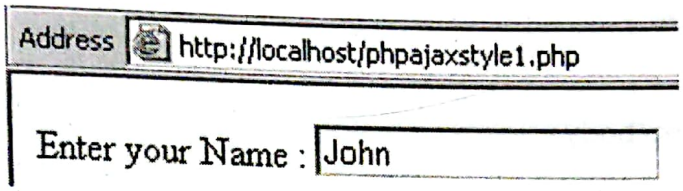


Figure 6.14 *User Name is Entered*

The user name appears in the applied style (Figure 6.15).

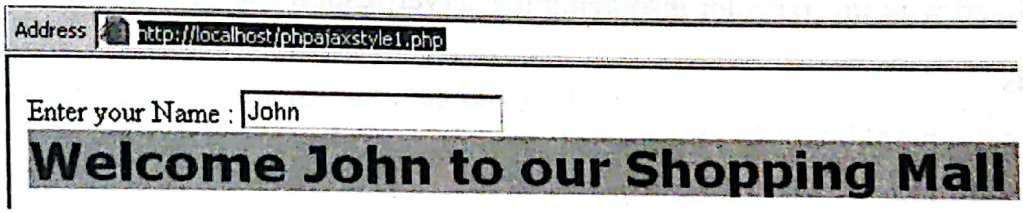


Figure 6.15 *Welcome Message is Displayed in Applied Styles*

6.11 SENDING DATA FROM COMBOBOX TO SERVER ASYNCHRONOUSLY

Let us see how the items selected in the combobox can be sent to the server asynchronously. That is, without a need to press any submit button, just selecting an item from the combobox will display the desired result. Let's make three files named comboajax.php, comboajax.js and showcat.php with the contents below in the `htdocs` subfolder of the `apache` directory.

comboajax.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="comboajax.js"></
script>
</head>
<body>
```

```
Category:
        <select name="Category" onChange='itemdisplay(this.value)' >
            <option value="" selected>Select Products Category</option>
            <option value="Camera">Camera</option>
            <option value="Mobile">Mobile</option>
            <option value="Book">Book</option>
        </select><br><br>

<div id="info"></div></body>
</html>
```

This program just creates a combobox named "Category" with three items Camera, Mobile and Book displayed in it. If any change takes place, that is if any item is selected from this combobox, a method `itemdisplay()` is invoked and the selected item is passed to it as argument. Also a element is defined with id `info` for displaying the server response.

comboajax.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 itemdisplay(cat)
{
```

```
var xmlhttp=makeRequestObject();
   xmlhttp.open('GET', 'showcat.php?&Category='+cat, 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)
}
```

This program is doing the same thing, that is making an XMLHttpRequest object and making a request to the server (by GET method) for the file showcat.php, sending the category of item selected by the user (from the combobox) to it. This category is retrieved in showcat.php file using `$_GET` array.

Since an asynchronous request is made to the server, the state of the request and the response to the request is checked. If the value of *readyState* property becomes 4 (meaning the request is complete) and the value of the status of the HTTP Request is 200 (which means there is no error), the response is then retrieved from the response stream and is assigned to variable content which is then applied to the innerHTML property (property used to display results) of the element with id `info`. It is the same element with id `info` we have defined in the web page for displaying the server response. Hence, the response retrieved from the server is displayed at the place of element `info`.

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

showcat.php

```
<?php
echo "Category selected is: " . $_GET['Category'] ;
?>
```

This file returns the text "Category selected is " and the category name selected by the user. The category selected by the user is passed while making the XMLHttpRequest for the file showcat.php in the following command:

```
xmlhttp.open('GET', 'showcat.php?&Category='+cat, true);
```

The output of the program comboajax.php may appear as shown in Figure 6.16.

Output:

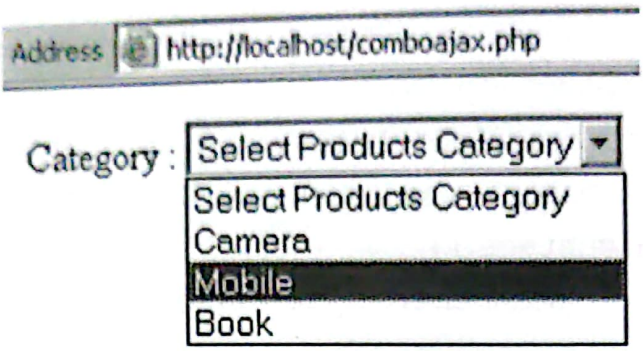


Figure 6.16 *Combobox Displaying Various Items*

The moment we select an item from the combobox, the message displaying the selected option appears on the screen (Figure 6.17).

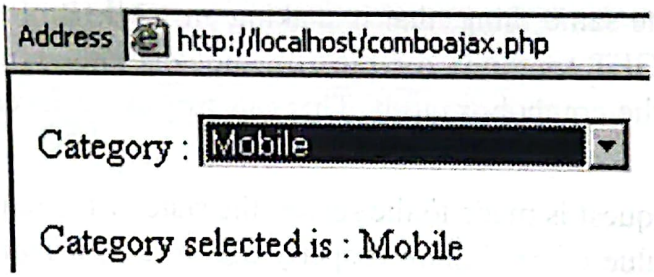


Figure 6.17 *The Selected Item is Displayed on the Screen*

6.12 SENDING MULTIPLE ITEMS SELECTED FROM LISTBOX TO THE SERVER ASYNCHRONOUSLY

In the following example we demonstrate how multiple items selected from the listbox are sent to the server asynchronously. Let's make three files named listajax.php, listajax.js and showlistdata.php with the contents below in the `htdocs` subfolder of the `apache` directory.

listajax.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="listajax.js"></
script>
</head>
<body>
Choose one or more category: <br>
<select name="Category" multiple="multiple" size="4" onblur='itemdisplay()' >
```

```
<option value="Camera" selected>Camera</option>
<option value="Mobile">Mobile</option>
<option value="Book">Book</option>
<option value="Computer">Computer</option>
<option value="Washing Machine">Washing Machine</option>
<option value="Refrigerator">Refrigerator</option>
</select><br><br>

<div id=info></div></body>
</html>
```

This program creates a listbox named "Category" with six items: Camera, Mobile and Book, Computer, Washing Machine and Refrigerator displayed in it. The multiselect feature of the listbox is set with the help of "`multiple`" attribute. So, the user can select more than one item from the listbox (by pressing `Ctrl` or shift key). After selecting the item(s), the moment user presses `tab` key or clicks anywhere else on the web page i.e. when the focus of the listbox is lost, a method `itemdisplay()` is invoked. Beside this, a element is defined with id `info` for displaying the server response.

listajax.js

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

```
17. }

18. function itemdisplay()
19. {
20. var xmlhttp=makeRequestObject();
21. optionsArray=document.getElementById('Category').options;
22. var selectedArray = new Array();
23. var x=0;
24. for(var i=0; i< optionsArray.length; i++)
25. {
26. if(optionsArray[i].selected)
27. {
28. selectedArray[x]=optionsArray[i].value;
29. x++;
30. }
31. }
32. var cat="";
33. for(var i=0;i<selectedArray.length;i++)
34. {
35. if( i!=selectedArray.length -1) {
36. cat +=selectedArray[i]+",";
37. }
38. else
39. {
40. cat+=selectedArray[i];
41. }
42. }
43. xmlhttp.open('GET', 'showlistdata.php?&Category='+cat, true);
44. xmlhttp.onreadystatechange=function() {
45. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
46. var content = xmlhttp.responseText;
47. if( content ){
```

```
48. document.getElementById('info').innerHTML = content;
49. }
50. }
51. }
52. xmlhttp.send(null)
53. }
```

Explanation of listajax.js program

Statements 1–17 are making an XMLHttpRequest object.

```
21. optionsArray=document.getElementById('Category').options;
```

The element in the web page named "Category" (our listbox) is searched and all the items available in it are assigned to an array `optionsArray`.

```
22. var selectedArray = new Array();
23. var x=0;
24. for(var i=0; i< optionsArray.length; i++)
25. {
26. if(optionsArray[i].selected)
27. {
28. selectedArray[x]=optionsArray[i].value;
29. x++;
30. }
31. }
```

With the help of a loop, all the elements in the `optionsArray` are checked one by one and all the items selected by the user are stored in a separate array `selectedArray`.

```
32. var cat="";
33. for(var i=0;i<selectedArray.length;i++)
34. {
35. if( i!=selectedArray.length -1) {
36. cat +=selectedArray[i]+",";
37. }
38. else
39. {
40. cat+=selectedArray[i];
41. }
42. }
```

All the items stored in array `selectedArray` (containing only the list of items selected by the user) are retrieved and stored in a variable `cat` (after separating them with commas except the last one).

```
43. xmlhttp.open('GET', 'showlistdata.php?&Category='+cat, true);
44. xmlhttp.onreadystatechange=function() {
45. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
46. var content = xmlhttp.responseText;
47. if( content ){
48. document.getElementById('info').innerHTML = content;
49. }
50. }
51. }
```

XMLHttpRequest request is made to the server (by GET method) for the file showlistdata.php, sending all the item(s) selected by the user (stored in variable `cat`) to it. The selected item list sent to the requested file will be retrieved in showlistdata.php file using `$_GET` array.

The state and the response to asynchronous request is checked. If the value of readyState property becomes 4 (meaning the request is complete) and the value of the status of the HTTP request is 200 (which means there is no error), the response is then retrieved from the response stream and is assigned to variable `content` which is then applied to the innerHTML property (property used to display results) of the element with id `info` to display all the items that are selected by the user in the listbox.

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

showlistdata.php

```
<?php
echo "The categories of items selected are: " . $_GET['Category'] ;
?>
```

This file returns the text: "The categories of items selected are " and the category name(s) selected by the user. The categories selected by the user are passed to this file while making the XMLHttpRequest for the file by following command:

```
43. xmlhttp.open('GET', 'showlistdata.php?&Category='+cat, true);
```

The output of the program listajax.php may appear as shown in Figure 6.18.

Output:

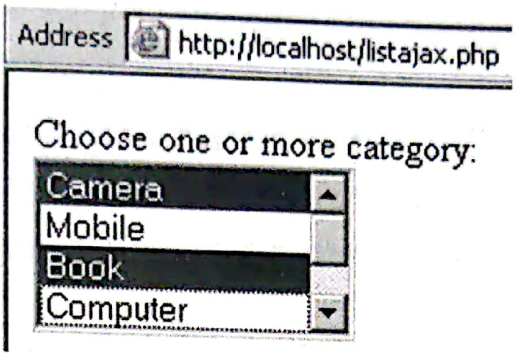


Figure 6.18 *Selection of Various Items by Pressing Tab Key*

We can select as many categories of items as we want followed by pressing the tab key. All the categories selected are displayed separated by commas as shown in Figure 6.19.

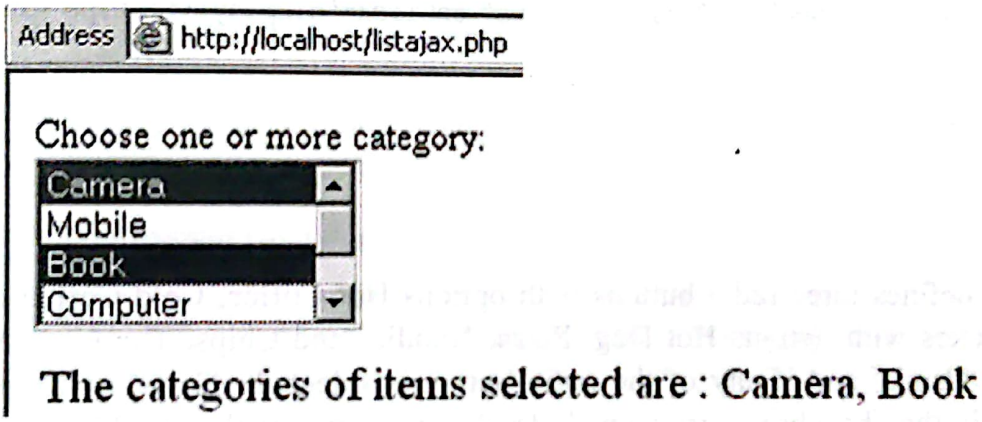


Figure 6.19 *Items Selected are Displayed*

6.13 SENDING ITEMS SELECTED FROM RADIO AND CHECK BOX TO SERVER ASYNCHRONOUSLY

In usually every web application, listboxes, comboboxes, checkboxes and radio buttons play a major role in getting information from the user. So, after understanding how the messages from combobox and listbox are sent to the server asynchronously, it is time to demonstrate how to send the asynchronous messages of the selected checkboxes and radio boxes to the server. Let's make three files named checkboxradioajax.php, checkboxradioajax.js and checkboxradiodata.php with the following contents in the `htdocs` subfolder of the `apache` directory.

checkboxradioajax.php

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

```
</head>
<body>
Choose one: <br>
<input type="radio" name="drink" value ="Hot Coffee" onClick='display()'> Hot Coffee <br>
<input type="radio" name="drink" value ="Cold Coffee" onClick='display()'> Cold Coffee <br>
<input type="radio" name="drink" value ="Soft Drink" onClick='display()'> Soft Drink <br><br>

Choose all the options that you want to have: <br>
<input type="checkbox" name="food" value ="Hot Dog" onClick='display()'> Hot Dog <br>
<input type="checkbox" name="food" value="Pizza" onClick='display()'>Pizza <br>
<input type="checkbox" name="food" value="Noodles" onClick='display()'>Noodles <br>
<input type="checkbox" name="food" value="Chips" onClick='display()'>Chips <br><br>

<div id=info></div>
</body>
</html>
```

This program defines three radio buttons with options Hot Coffee, Cold Coffee and Soft Drink and four checkboxes with options Hot Dog, Pizza, Noodles and Chips. The name assigned to the radio buttons is "drink" and if any of the radio button is selected, `display()` method will be invoked. Similarly the checkboxes are named "food" and if any of the checkboxes is selected, the same `display()` method will be invoked. Beside this, a element is defined with id `info` for displaying the server response. Remember, we can select only one radio button in a group whereas any number of checkboxes can be selected in a group.

checkboxradioajax.js

```
1. function makeRequestObject(){
2. var xmlhttp=false;
3. try {
4. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
5. } catch (e) {
6. try {
7. xmlhttp = new
8. ActiveXObject('Microsoft.XMLHTTP');
9. } catch (E) {
10. xmlhttp = false;
11. }
```

```
12. }
13. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
14. xmlhttp = new XMLHttpRequest();
15. }
16. return xmlhttp;
17. }

18. function display()
19. {
20. var xmlhttp=makeRequestObject();
21. ControlsArray=document.getElementsByTagName('input');
22. var fooditms="";
23. var drinkitms="";
24. for(var i=0; i< ControlsArray.length; i++)
25. {
26. if(ControlsArray[i].type=="checkbox" && ControlsArray[i].checked)
27. {
28. fooditms+=ControlsArray[i].value + " ";
29. }
30. if(ControlsArray[i].type=="radio" && ControlsArray[i].checked)
31. {
32. drinkitms+=ControlsArray[i].value + " ";
33. }
34. }

35. xmlhttp.open('GET','checkradiodata.php?SelectedDrink='+drinkitms+'&SelectedFood='+fooditms,
true);
36. xmlhttp.onreadystatechange=function() {
37. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
38. var content = xmlhttp.responseText;
39. if( content ){
40. document.getElementById('info').innerHTML = content;
41. }
42. }
43. }
44. xmlhttp.send(null)
45. }
```

Explanation of checkboxradioajax.js program

Statements 1–17 are making an XMLHttpRequest object.

```
21. ControlsArray=document.getElementsByTagName('input');
```

The elements in the web page named `input` (usually all the elements in a web page come under this category) are searched and all the items with this name are assigned to an array ControlsArray.

```
22. var fooditms="";
23. var drinkitms="";
24. for(var i=0; i< ControlsArray.length; i++)
25. {
26. if(ControlsArray[i].type=="checkbox" && ControlsArray[i].checked)
27. {
28. fooditms+=ControlsArray[i].value + " ";
29. }
30. if(ControlsArray[i].type=="radio" && ControlsArray[i].checked)
31. {
32. drinkitms+=ControlsArray[i].value + " ";
33. }
34. }
```

With the help of a loop, all the elements in the `ControlsArray` are checked one by one and the value of all the elements in the array which are of type "checkbox" and are checked (selected) are stored in a variable `fooditms`. Similarly, for all of the elements of type "radio" which are checked, their values is assigned to variable `drinkitms`.

```
35. xmlhttp.open('GET','checkradiodata.php?SelectedDrink='+drinkitms+'&
    SelectedFood='+fooditms, true);
36. xmlhttp.onreadystatechange=function() {
37. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
38. var content = xmlhttp.responseText;
39. if( content ){
40. document.getElementById('info').innerHTML = content;
41. }
42. }
43. }
44. xmlhttp.send(null)
45. }
```

An XMLHttp request is made to the server (by GET method) for the file checkboxradiodata.php, sending all the selected drink items and selected food items (stored in variables `drinkitms` and `fooditms` respectively) to it.

The state and the response to asynchronous request is checked. If the value of readyState property becomes 4 and the value of the status of the HTTP Request is 200, the response is then

retrieved from the response stream and is assigned to variable `content` which is then applied to the innerHTML property of the element with id `info` to display the response from the server (i.e. to display all the items selected from the checkbox and radio buttons).

The instruction `xmlhttp.send(null)` is used to actually send the request to the server. And if we don't want to pass any query string to the requested file, the argument passed to this method is null or else it contains the query string.

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

checkboxradiodata.php

```
<?php
$food=$_GET['SelectedFood'];
$drink=$_GET['SelectedDrink'];
if($drink=="")
{
echo "You have not selected any drink <br><br>";
}
else
{
echo "The drink selected is: " . $drink . "<br><br>";
}
if($food=="")
{
echo "You have not selected food <br><br>";
}
else
{
echo "The types of Food selected are: " . $food . "<br><br>";
}
?>
```

The selected food items and selected drink items are retrieved from `$_GET` array and stored in variables `$food`, and `$drink` respectively. The list of items selected from checkboxes and radio buttons are passed to this file while making the XMLHttp request for the file using the following command:

```
35. xmlhttp.open('GET', 'checkradiodata.php? SelectedDrink='+drinkitms+'
&SelectedFood='+ fooditms, true);
```

We can see in the above instruction that the selected drink item and the selected food items are passed to the checkboxradiodata.php file. The retrieved items in `$food` and `$drink` variables are displayed on the screen (if the contents of the variable are non blank or else the message: "You have not selected" food or drink is displayed).

Output:

The initial screen may appear as shown in Figure 6.20.

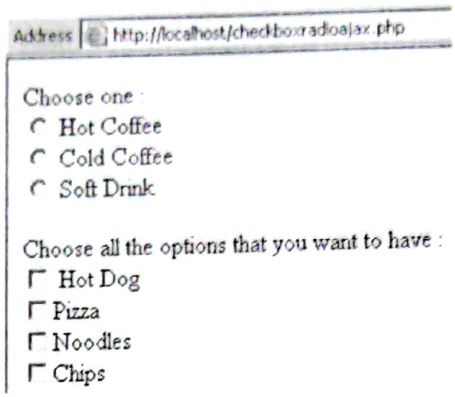


Figure 6.20 *Output of Checkbox Program*

If only drink item is selected, the output will as shown in Figure 6.21.

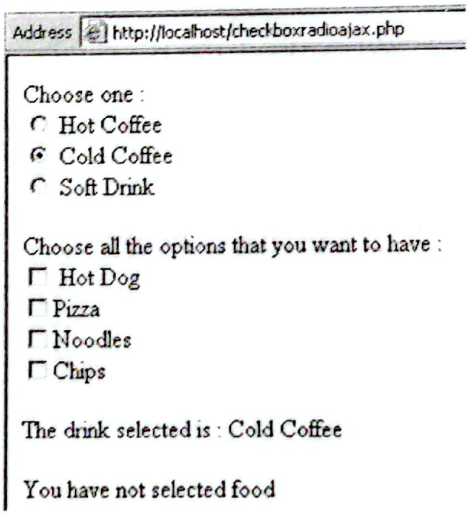


Figure 6.21 *Output if Only Drink Item is Selected*

If only food item is selected, the output may appear as shown in Figure 6.22.

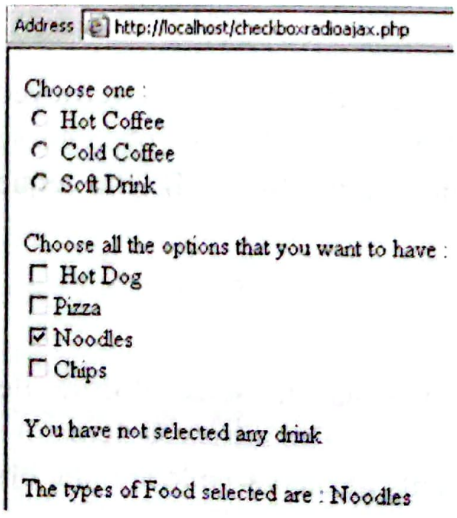


Figure 6.22 *Output if Only Food Item is Selected*

If both drink as well as food items are selected, the output may appear as shown in Figure 6.23.

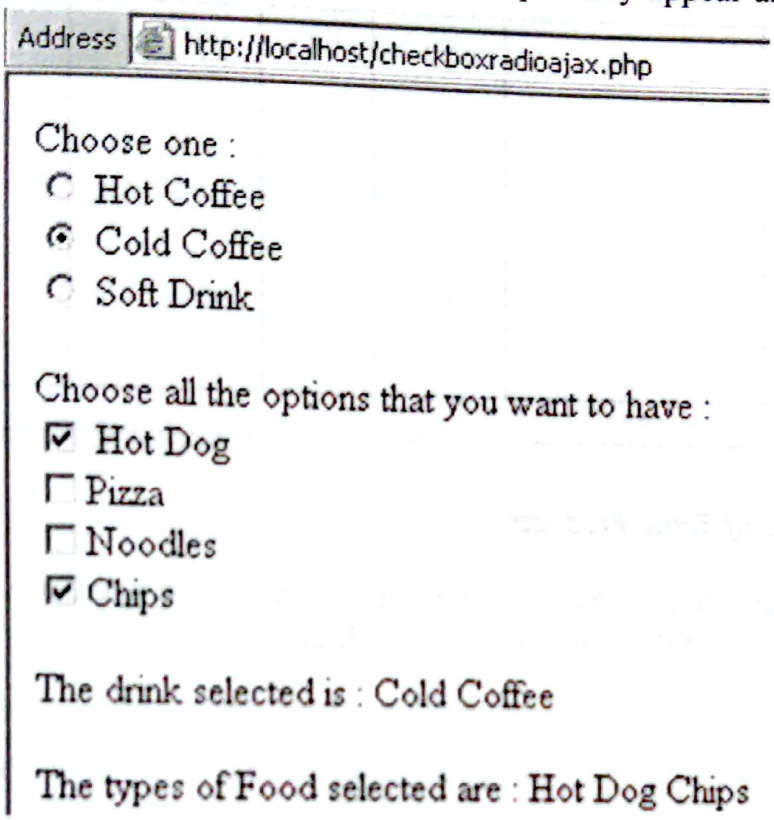


Figure 6.23 ***Output if Both Drink and Food Items are Selected***

6.14 AJAX, PHP AND MYSQL ALL COMBINED FOR ACCESSING DATABASE

Let us place an asynchronous request to access the desired data from the database on server. In the following example, the user is provided two categories of items in the form of hyperlinks Camera and Mobile and he can select either of them to see the products available in that category. When the user selects any link, all the items in the `products` table (of shopping database) under the selected category will be displayed. That is, if the camera hyperlink is selected, all the cameras in the products table will be displayed. We are using the same shopping database and the same `products` table made in the Shopping Cart project included in Part I of this book. For reference purposes, the structure of the table is shown in Figure 6.24.

```
mysql> describe product;
```

Field	Type	Null	Key	Default	Extra
Item_code	Int<4>	NO			auto_increment
item_name	Varchar<50>	NO			
Description	varchar<255>	NO			
quantity	varchar<50>	NO			
price	int<4>	NO			
image	float	YES		NULL	
imagename	blob	YES		NULL	
id	varchar<50>	YES		NULL	
	int<10>	NO		0	

```
9 rows in set <0.28 sec>
```

Figure 6.24 ***Structure of Table Products***

Let's make three files named ajaxform8.php, ajaxform8.js and ajaxdata8.php with the following contents in the `htdocs` subfolder of the `apache` directory.

ajaxform8.php

```
<head>
<script language="JavaScript" type="text/JavaScript" src="ajaxform8.js" >
</script>
</head>
<body>
<div id="links">
<a href="javascript:showdata('Camera')">Camera</a><a href="javascript: showdata
('Mobile')">Mobile</a>
</div>
<div id="content">
</div>
</body>
</html>
```

This program first includes the JavaScript file ajaxform8.js (so that methods in it can be invoked from this web page). It is defining two elements with id `links` and `content` respectively. The two hyperlinks Camera and Mobile are placed inside the `links` element and the `content` element is left for displaying the server response. If either link is selected, a JavaScript method `showdata()` is executed and the name of the link selected (category of item selected) is passed to it as argument.

ajaxform8.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(cat)
{
var xmlhttp=makeRequestObject();
var file = 'ajaxdata8.php?Category=';
      xmlhttp.open('GET', file + cat, true);
   xmlhttp.onreadystatechange=function() {
   if (xmlhttp.readyState==4) {
document.getElementById('content').innerHTML = xmlhttp.responseText;
}
}
xmlhttp.send(null);
return;
}
```

This program is doing the same thing, that is making an XMLHttpRequest object and making request to the server (by GET method) for the file ajaxdata8.php, sending the category of item

selected by the user (through hyperlink) to it. This category is retrieved in ajaxdata8.php file using `$_GET` array.

The state of the request is checked. If the value of *readyState* property becomes 4 (meaning the request is complete), the response is then retrieved from the response stream and is assigned to the innerHTML property (property used to display results) of the element with id `content`.

Since the response generated by the server is based on the execution of the file ajaxdata8.php, let's analyse what it returns.

ajaxdata8.php

```
1. <?php
2. mysql_connect("localhost", "root", "mce");
3. mysql_select_db("shopping");
4. $cat = $_GET["Category"];
5. $sql = "SELECT * FROM products WHERE Category like '$cat'";
6. $query = mysql_query($sql) or die(mysql_error());
7. echo '<table border="1">';
8. echo '<th>Item Code</th><th>Name of Item</th><th>Description</th><th>Price</
   th>';
9. while ($r = mysql_fetch_array($query)) {
10. echo '<tr><td>' . $r["item_code"] . '</td><td>' . $r["item_name"] . '< td>
   <td>' . $r["description"] . '</td><td>' . $r["price"] . '</td></tr>';
11. }
12. echo '</table>';
13. ?>
```

This program is connecting to the MySQL server with user id "root" and password "mce". Then the database "shopping" is selected (as our `products` table is in this database). The category of the item selected by the user is retrieved using `$_GET` array and stored in variable `$cat`. Then, a SQL statement is executed to fetch all the rows from the `products` table with the category specified in **$cat** variable. The rows retrieved from the products table are then returned in the form of a table to be displayed on the client.

Output:

The user is provided two hyperlinks each designating a category of items (Figure 6.25).

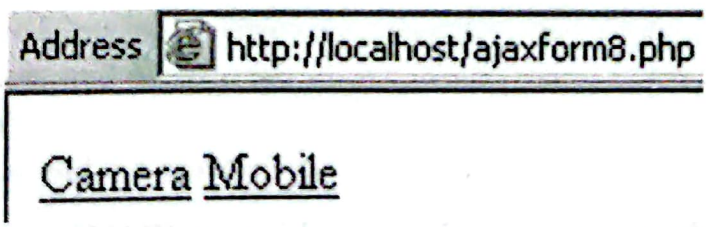


Figure 6.25 *Output Displaying Hyperlinks*

If Camera link is selected, all the products under the camera category will be displayed as shown in Figure 6.26.

Address http://localhost/ajaxform8.php

Camera Mobile

Item Code	Name of Item	Description	Price
101	C1001	This is the best Camera of Model C1001	2200
103	C1003	This is the best Camera of Model C5003	6500
104	C1004	This is the best Camera of Model C1004	5300
102	C1002	This is the best Camera of Model C1002	4500

Figure 6.26 *Description of All Items Under Camera Category if Selected*

If Mobile link is selected, all the products under the mobile category are displayed (Figure 6.27).

Address http://localhost/ajaxform8.php

Camera Mobile

Item Code	Name of Item	Description	Price
201	M1001	This is the best Mobile of Model M1001	2800
202	M1002	This is the best Mobile of Model M1002	2200
203	M1003	This is the best Mobile of Model M1003	3500
204	M1004	This is the best Mobile of Model M1004	5700

Figure 6.27 *All Products Under Mobile Category are Shown if Selected*

6.15 ACCESSING DATABASE IN AJAX—SECOND EXAMPLE

In this example, there is a limitation that the user can see products of only one category at a time. That is, if he selects "Camera" hyperlink then only the products under the camera category are displayed and if the user selects "Mobile" hyperlink, all the items under the mobile category are displayed. What if the user wishes to see the products of both categories together? So, in the following example, we provide the user a set of checkboxes each designating a category of products and the user can select any number of checkboxes to see the products of one or more categories together.

Again, the `shopping` database and the `products` table created in the Shopping Cart project explained in Part I of this book are used in this example. Let's make three files named accessdatabase1.php, accessdatabase1.js and retrievedata.php with the following contents in the `htdocs` subfolder of the `apache` directory.

accessdatabase1.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="accessdatabase1.
js"></script>
</head>
<body>
Choose all the categories of items you want to see: <br>
<input type="checkbox" name="cat" value ="Camera" onClick='display()'> Camera
<br>
<input type="checkbox" name="cat" value="Mobile" onClick='display()'>Mobile
<br>
<input type="checkbox" name="cat" value="Book" onClick='display()'>Book
<br><br>
<div id="itminfo"></div>
</body>
</html>
```

This program includes the JavaScript file accessdatabase1.js (so that its methods can be invoked from the current web page). Three checkboxes are created named "cat" and each of them designates a category of product. If any of the checkboxes is selected, it will result in execution of `display()` method specified in accessdatabase1.js file.

Also, a element with id `itminfo` is defined for displaying the server response.

accessdatabase1.js

```
1. function makeRequestObject(){
2. var xmlhttp=false;
3. try {
4. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
5. } catch (e) {
6. try {
7. xmlhttp = new
8. ActiveXObject('Microsoft.XMLHTTP');
9. } catch (E) {
10. xmlhttp = false;
```

```
11. }
12. }
13. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
14. xmlhttp = new XMLHttpRequest();
15. }
16. return xmlhttp;
17. }

18. function display()
19. {
20. var xmlhttp=makeRequestObject();
21. ControlsArray=document.getElementsByTagName('input');
22. var selectedcat="";
23. for(var i=0; i< ControlsArray.length; i++)
24. {
25. if(ControlsArray[i].type=="checkbox" && ControlsArray[i].checked)
26. {
27. selectedcat+=ControlsArray[i].value + " ";
28. }
29. }
30. xmlhttp.open('GET', 'retrievedata.php?SelectedCategory='+selectedcat,
    true);
31. xmlhttp.onreadystatechange=function() {
32. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
33. var content = xmlhttp.responseText;
34. if( content ){
35. document.getElementById('itminfo').innerHTML = content;
36. }
37. }
38. }
39. xmlhttp.send(null)
40. }
```

Explanation of accessdatabase1.js program

Statements 1–17 are making an XMLHttpRequest object.

```
21. ControlsArray=document.getElementsByTagName('input');
```

The elements in the web page with tag name `input` (usually all the elements in a web page come under this category) are searched and all the items with this tag name are assigned to an array: `ControlsArray`.

```
22. var selectedcat="";
23. for(var i=0; i< ControlsArray.length; i++)
24. {
25. if(ControlsArray[i].type=="checkbox" && ControlsArray[i].checked)
26. {
27. selectedcat+=ControlsArray[i].value + " ";
28. }
29. }
```

With the help of a loop, all the elements in the `ControlsArray` are checked one by one and the values of all the elements in the array which are of type "checkbox" and are checked (selected) are stored in a variable `selectedcat` (separated by a space).

```
30. xmlhttp.open('GET', 'retrievedata.php?SelectedCategory='+selectedcat, true);
31. xmlhttp.onreadystatechange=function() {
32. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
33. var content = xmlhttp.responseText;
34. if( content ){
35. document.getElementById('itminfo').innerHTML = content;
36. }
37. }
38. }
39. xmlhttp.send(null)
40. }
```

An XMLHttp request is made to the server (by GET method) for the file retrievedata.php, sending names of the categories selected by the user to it. The state and the response to asynchronous request is checked. If the value of readyState property becomes 4 and the value of the status of the HTTP Request is 200, the response is then retrieved from the response stream and is assigned to variable content which is then applied to the innerHTML property of the element with id `itminfo` to display the response from the server (i.e. to display all the products of the selected categories).

The instruction `xmlhttp.send(null)` is used to actually send the request to the server. And if we don't want to pass any query string to the requested file, the argument passed to this method is null or else it contains the query string.

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

retrievedata.php

```
1. <?php
2. mysql_connect("localhost", "root", "mce");
3. mysql_select_db("shopping");
4. $categories = $_GET["SelectedCategory"];
5. $catArray=explode(" ", $categories);
6. $prod="";
7. for($i=0;$i<sizeof($catArray);$i++)
8. {
9. if( $i!=sizeof($catArray) -1) {
10. $prod = $prod . "'" . $catArray[$i] . "', ";
11. }
12. else
13. {
14. $prod= $prod . "'" . $catArray[$i] . "'";
15. }
16. }
17. $sql = "SELECT * FROM products WHERE Category in (" . $prod . ") order by item_code";
18. $query = mysql_query($sql) or die(mysql_error());
19. echo '<table border="1">';
20. echo '<th>Item Code</th><th>Name of Item</th><th>Description</th><th>Price</th>';
21. while ($r = mysql_fetch_array($query)) {
22. echo '<tr><td>' . $r["item_code"] . '</td><td>' . $r["item_name"] . '</td>
    <td>' . $r["description"] . '</td><td>' . $r["price"] . '</td></tr>';
23. }
24. echo '</table>';
25. ?>
```

Explanation of retrievedata.php program

This program is connecting to the MySQL server with user id "root" and password "mce". Then the database "shopping" is selected (as our `products` table is in this database). The categories of the

items selected by the user are retrieved using `$_GET` array and stored in variable `$categories` (it contains all the categories of the items selected by the user each separated by a space).

```
5. $catArray=explode(" ", $categories);
```

The `$categories` variable is exploded (split). That is, the values stored in this variable are separated on the basis of space (i.e. wherever a space occurs, it marks the beginning of another value) and stored in an array `$catArray`.

```
6. $prod="";
7. for($i=0;$i<sizeof($catArray);$i++)
8. {
9. if( $i!=sizeof($catArray) -1) {
10. $prod = $prod . "'" . $catArray[$i] . "', ";
11. }
12. else
13. {
14. $prod= $prod . "'" . $catArray[$i] . "'";
15. }
16. }
```

A `for` loop is used to extract all the elements from the array `$catArray` and stored in a variable `$prod` and each element except the last one is separated by a comma.

```
17. $sql = "SELECT * FROM products WHERE Category in (" . $prod . ") order by item_code";
18. $query = mysql_query($sql) or die(mysql_error());
```

An SQL statement is executed to fetch all the rows from the products table with the categories specified in `$prod` variable.

```
19. echo '<table border="1">';
20. echo '<th>Item Code</th><th>Name of Item</th><th>Description</th><th>Price</th>';
21. while ($r = mysql_fetch_array($query)) {
22. echo '<tr><td>' . $r["item_code"] . '</td><td>' . $r["item_name"] . '</
    td><td>' . $r["description"] . '</td><td>' . $r["price"] . '</td></tr>';
23. }
24. echo '</table>';
```

The rows retrieved from the products table are then returned in the form of a table to be displayed to the client.

Output:

The user is provided with three checkboxes each designating a category of item (Figure 6.28).

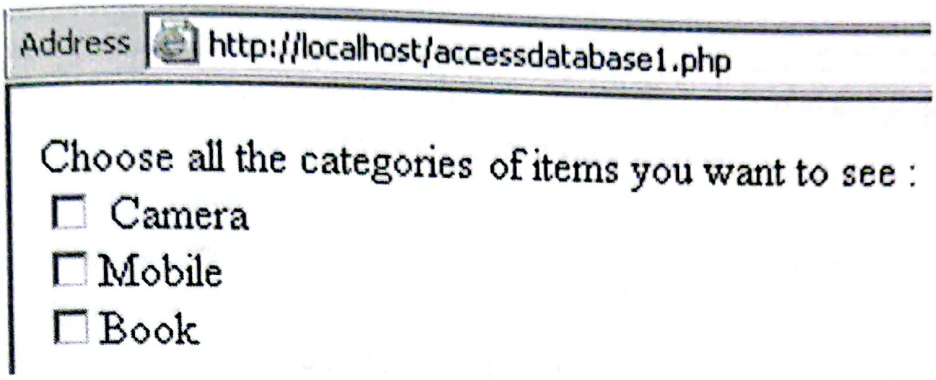
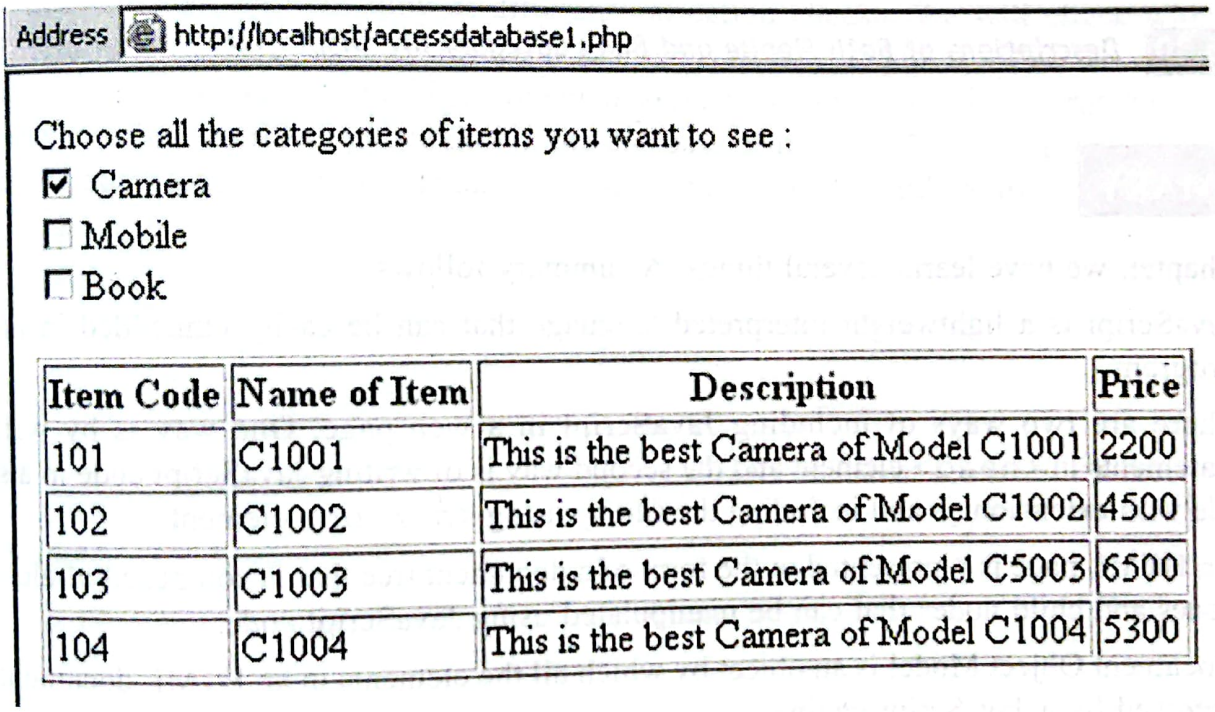


Figure 6.28 *Three Checkboxes to Select*

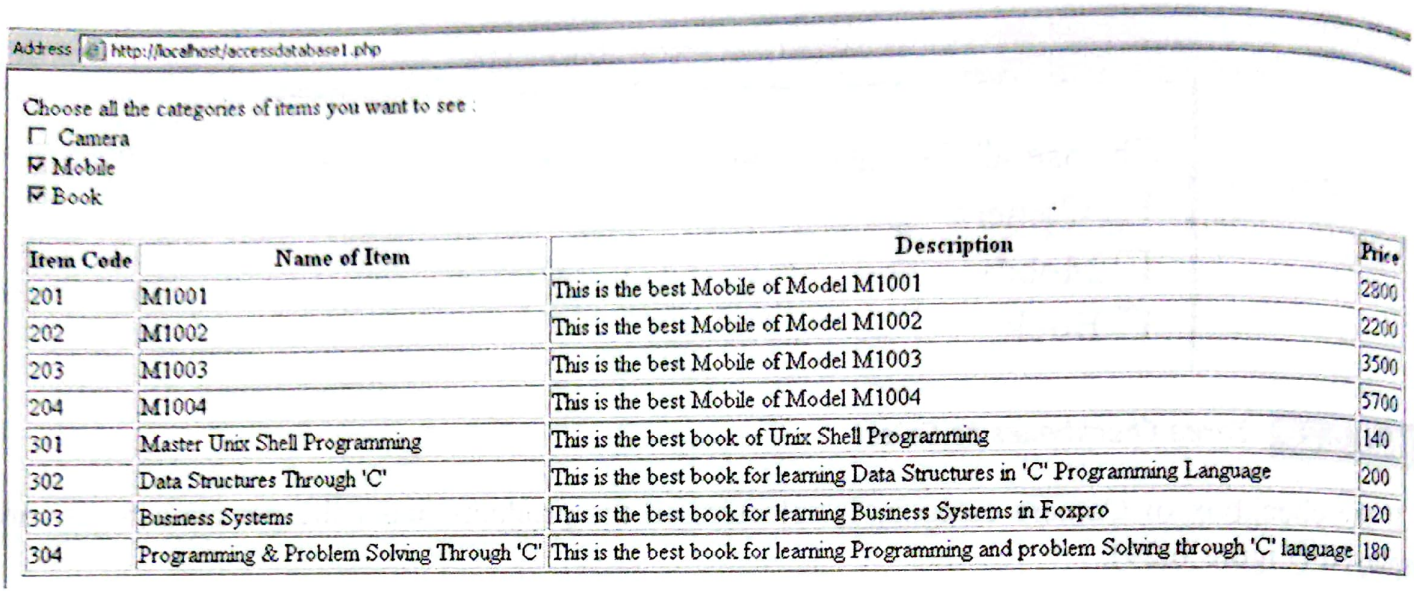
If the checkbox of Camera category is selected, all the products under the camera category are displayed (Figure 6.29).



Item Code	Name of Item	Description	Price
101	C1001	This is the best Camera of Model C1001	2200
102	C1002	This is the best Camera of Model C1002	4500
103	C1003	This is the best Camera of Model C5003	6500
104	C1004	This is the best Camera of Model C1004	5300

Figure 6.29 *Description of Camera Products if its Checkbox is Marked*

If more than one checkbox is selected, for example the checkboxes of Mobile and Book category are both selected, then all the products belonging to these two categories are displayed as shown in Figure 6.30.



Address http://localhost/accessdatabase1.php

Choose all the categories of items you want to see :

- ☐ Camera
- ☑ Mobile
- ☑ Book

Item Code	Name of Item	Description	Price
201	M1001	This is the best Mobile of Model M1001	2800
202	M1002	This is the best Mobile of Model M1002	2200
203	M1003	This is the best Mobile of Model M1003	3500
204	M1004	This is the best Mobile of Model M1004	5700
301	Master Unix Shell Programming	This is the best book of Unix Shell Programming	140
302	Data Structures Through 'C'	This is the best book for learning Data Structures in 'C' Programming Language	200
303	Business Systems	This is the best book for learning Business Systems in Foxpro	120
304	Programming & Problem Solving Through 'C'	This is the best book for learning Programming and problem Solving through 'C' language	180

Figure 6.30 ***Descriptions of Both Mobile and Book Category are Displayed***

SUMMARY

In this chapter, we have learnt several things. A summary follows:

- JavaScript is a lightweight interpreted language that can be easily embedded in an `html` program.
- There are two ways of including JavaScript in a web page. One way is by writing its statements in `<head>` element and the second way is of writing JavaScript code in a separate file with extension .js and including the file by using `<script>` element.
- An HTML page is represented in the form of a document tree that is composed of elements or nodes and child nodes that can be manipulated using JavaScript code.
- Document Object Model is an object by which all the elements in an HTML document can be accessed by a JavaScript engine.
- The root node of the web page is accessed by a global variable called `document`.
- The method used for searching a web page for an object with the specified id is `document.getElementById()`.
- The object that is used for making asynchronous calls to the web server is known as XMLHttpRequest object. It is using this object that we can make HTTP requests, receive responses and update a region of the page completely in the background.
- The keydown event fires only when a character key is pressed.
- Since in some browsers XMLHttpRequest is a native object whereas in others it is implemented as an ActiveX control, the method of making its instance is different for each browser.

- To watch for the state of the request and also response to the request (since the request made to the server is an asynchronous request), we take the help of event handler `onreadystatechange` which fires at every state change.
- To actually send the request to the server, the statement used is `xmlhttp.send(null)`
- In POST requests, the information is passed in the form of a query string.
- In the query string, the name and the value of each parameter must be URL-encoded in order to avoid data loss during transmission. This encoding is done with the help of a built-in function in JavaScript called `encodeURIComponent()`.
- The CSS style sheet is a collection of font styles stored in a separate file with extension .css and is considered as the most favourable method of applying consistent styles to the elements on a web page.

In the next chapter, we will look at different validation checks. We will check whether any essential field is left blank, whether any field has less data then required. Also, we will see three different methods of validating email ids. We will also see how a userid can be validated i.e. checking whether the specified userid already exists in the database or not. Finally, we will see how a form is validated and storing the record in the table (if data in the form is found valid).

7

Validation

In this chapter we will look at the following validation checks:

- Checking if any field is left blank
- Checking if any field has less data then required
- Three different methods of validating email id
- Validating userid—checking if it already exists
- Validating the form and storing the record in the table (if data is valid)

Validation means confirming whether the data entered by the user is correct (in terms of type and value). If not, the data must not be processed and the user must be asked to correct the mistake.

In a traditional web application, form validation is performed on the click of the submit button i.e. when all the data has been entered by the user, but on application of AJAX, every field can be validated individually and that too as and when data is entered.

There are certain fields in a web form which must be validated immediately before proceeding with the rest of the fields, like userid (checking whether the userid entered already exists or not), emailid (whether the email id is correct or not) etc.

7.1 CHECKING IF ANY FIELD IS LEFT BLANK

While asking the user to fill up a form, a common problem that is faced is that some important fields are left blank by the user. So, through AJAX, we can make sure that certain required fields are not left blank. Let's check it with the following example. We make two files named validatefield.php and checkfield.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validatefield.php

```
<html>
<head>
```

```
<script language="JavaScript" type="text/JavaScript" src="checkfield.js"></script>
</head>
<body>
Userid: <input size="20" type="text" name="userid" onblur = "validatefield (this)">
</body>
</html>
```

This program is including JavaScript file checkfield.js file in the current page so that methods defined in it can be accessed from this web page. The user is prompted to enter userid and after feeding it, the moment tab key is pressed or the user clicks anywhere else on the form (meaning if the focus of that field is lost), `validatefield()` method specified in the included JavaScript file is executed and the userid entered by the user is passed to it as argument (in order to confirm that it is not left blank).

checkfield.js

```
function validatefield(obj)
{
    if(obj.value.length ==0)
    {
    alert("This field cannot be left blank");
    obj.focus();
}
}
```

This method is checking whether the length of the object (userid) sent to it is 0 or not. If yes, it means the user has left this field blank, and an alert message is displayed to him and focus is set to the userid field so that the user can enter userid in it.

Output:

If the user keeps that field blank and presses tab key, an alert message is displayed (Figure 7.1).

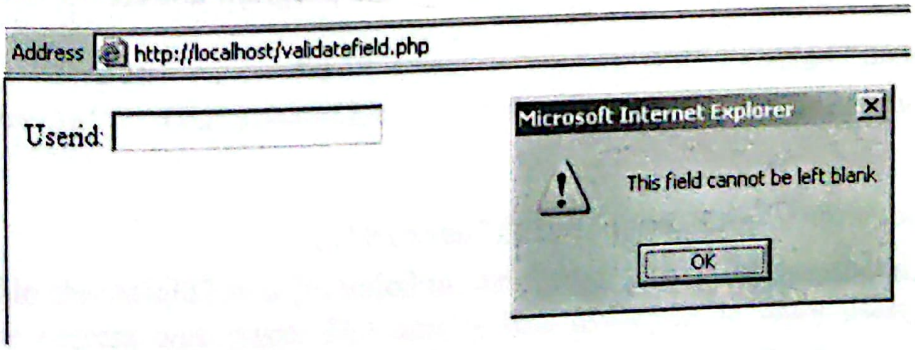


Figure 7.1 *Alert Message is Displayed if Field is Left Blank*

If the user enters something in the userid field, then no error message is displayed (Figure 7.2).

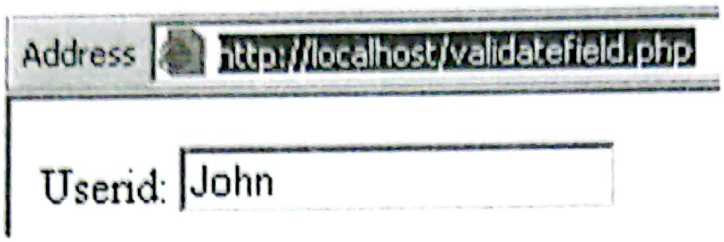


Figure 7.2 *No Error Message is Displayed if User Enters Something in the Field*

7.1.1 Second Method of the Same Program

In the following example instead of directly attaching onblur event to the web page element, we are using window.onload event handler (which is automatically called when the browser completes loading of the web page) to access the element (userid) on the web page and execute a set of statements if the onblur event has occurred on that element. Let's make two files named validatefield1.php and checkfield1.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validatefield1.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="checkfield1.js"></
script>
</head>
<body>
   Userid: <input size="20" type="text" name="userid">
</body>
</html>
```

We can see that onblur event handler is not there in the program above.

checkfield1.js

```
window.onload=function()
      {
      var obj=document.getElementById("userid");
      obj.onblur=function()
      {
      if(this.value.length ==0)
```

```
{
alert("This field cannot be left blank");
this.focus();
}
}
}
```

When the web page is loaded, window.onload event handler is automatically executed. We access the element on the document (web page) with name "userid" (with the help of `getElementById()` method) and check if onblur event has occurred on this element. If yes, then a function is executed which checks whether the length of the value entered for this element is 0 or not. If yes, then an alert message is displayed on the screen.

7.2 CHECKING IF ANY FIELD HAS LESS DATA THAN REQUIRED

Sometimes we want that the data entered in a given element should be of at least specified width (in the case of passwords, we want them to be of minimum 6 characters). In the following example, we are going to ask the user to enter userid (which cannot be left blank) and password which cannot be of less than 6 characters. We make two files named validatefields.php and checkfield2.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validatefields.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="checkfield2.js"></
script>
</head>
<body>
  Userid: <input size="20" type="text" name="userid" onblur
="validatefield(this)"><br>
  Password: <input size="20" type="password" name="password" onblur
="if(this.value !='') {validatefield(this)}">
</body>
</html>
```

A JavaScript file checkfield2.js is included in the current page so the methods defined in it can be accessed from the current web page. The user is first prompted to enter userid. The moment the user presses the tab key after entering userid, `validatefield()` method is executed with userid entered by the user passed to it as argument. Similarly, after userid, the user is prompted for

password too and the moment focus on that element is lost either by pressing tab key or by clicking mouse elsewhere on the web page, again `validatefield()` method is executed and the password entered is passed to it as argument.

Note: `validatefield()` method is executed only if password field is not left blank.

checkfield2.js

```
function validatefield(obj)
{
if(obj.name=="userid")
{
if(obj.value.length ==0)
{
alert("This field cannot be left blank");
obj.focus();
}
}

if(obj.name=="password")
{
if(obj.value.length <=5)
{
alert("The password should be of at least size 6");
obj.focus();
}
}
}
```

This method is checking if the object passed to it is userid. If it is so, then its length is checked and if it is 0, then an alert message is displayed to the user and the cursor is placed on userid field (using `focus()` method) so that the user can again feed it. And if the object passed to the method is password, and its length is less than 6 characters, then an alert message is displayed and again the cursor is set on the password field. In other words, focus is set on the password field for the user to re-enter the password of at least 6 characters.

Output:

If the user keeps userid field blank and presses the tab key, an alert message is displayed (Figure 7.3).

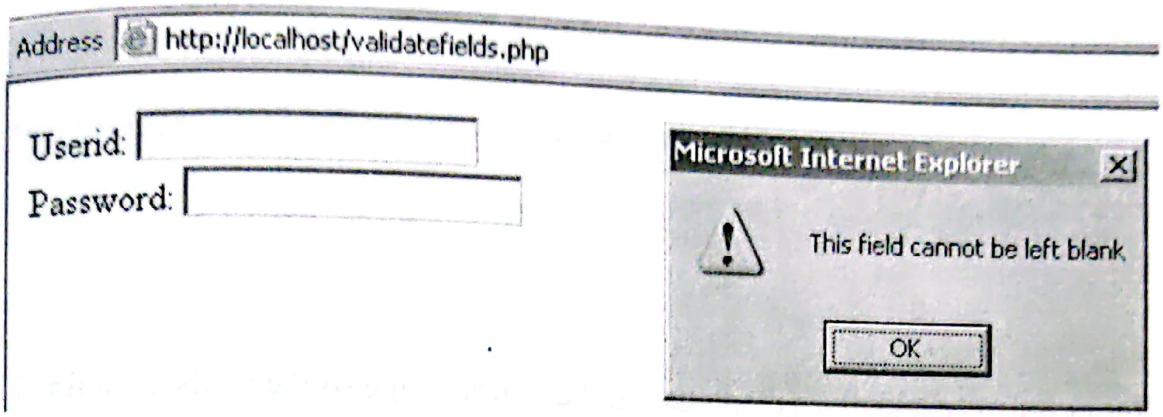


Figure 7.3 ***Alert Message is Displayed***

If the password entered is of less than 6 characters then an alert message is displayed (Figure 7.4).

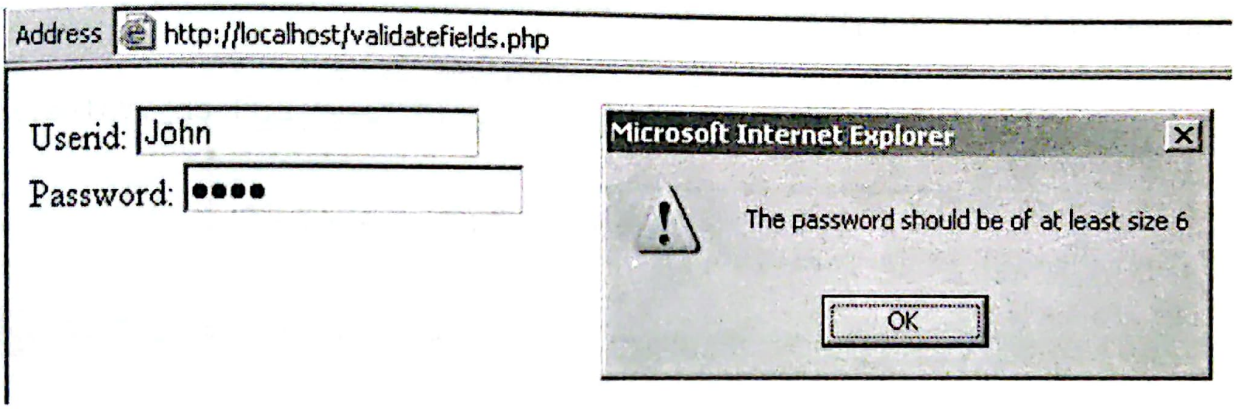


Figure 7.4 ***Alert Message is Displayed if Password is Less than 6 Characters Long***

7.3 VALIDATING EMAIL ID

The following example is for verifying whether the email id entered by user is a valid email id or not. To ensure the validity of the email id, we first check that the length of email id is not 0 (i.e. the user has not left the field blank). Then, we check for the presence of . (dot) and @ (at) in the email id entered by the user. Absence of either proves that invalid email id is entered.

We make two files named validateemail.php and checkemail.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validateemail.php

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

```
</head>
<body>
Enter email id: <input size="20" type="text" name="emailid" onblur
="validatefield(this)">
   </form>
</body>
</html>
```

A JavaScript file checkemail.js is included in the current page so the methods defined in it can be accessed from the current web page. The user is prompted to enter email id. The moment the user presses the tab key after entering email id, `validatefield()` method is executed with email id entered by the user passed to it as argument.

checkemail.js

```
function validatefield(obj)
{
      if(obj.value.length==0||obj.value.indexOf(".")==-1 || obj.value.indexOf
      ("@") ==-1)
      {
      alert("email id is invalid");
      obj.focus();
}
}
```

This method is checking if the length of the object passed to it (email id) is 0. If its length is 0 (meaning the field is left blank) or the object doesn't contain . (dot) or @ (at) symbol, then an alert message is displayed and focus is set on it for the user to enter a valid email id.

Output:

If email id doesn't contain . (dot) or @ (at) symbol, then an alert message is displayed (Figure 7.5)

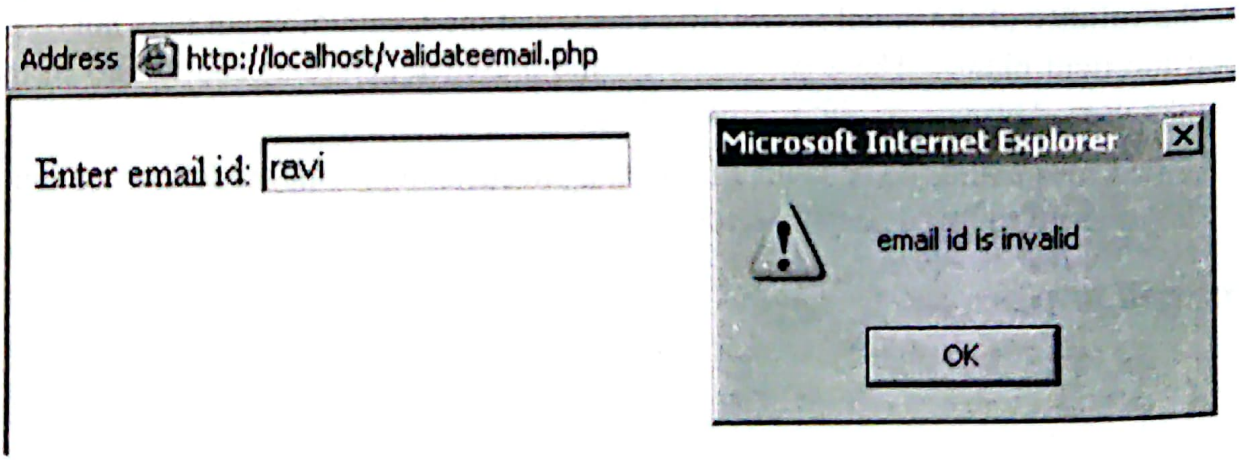


Figure 7.5 *Alert Message is Displayed if Email id is Invalid*

If email id doesn't contain . (dot) symbol, still an alert message is displayed (Figure 7.6).

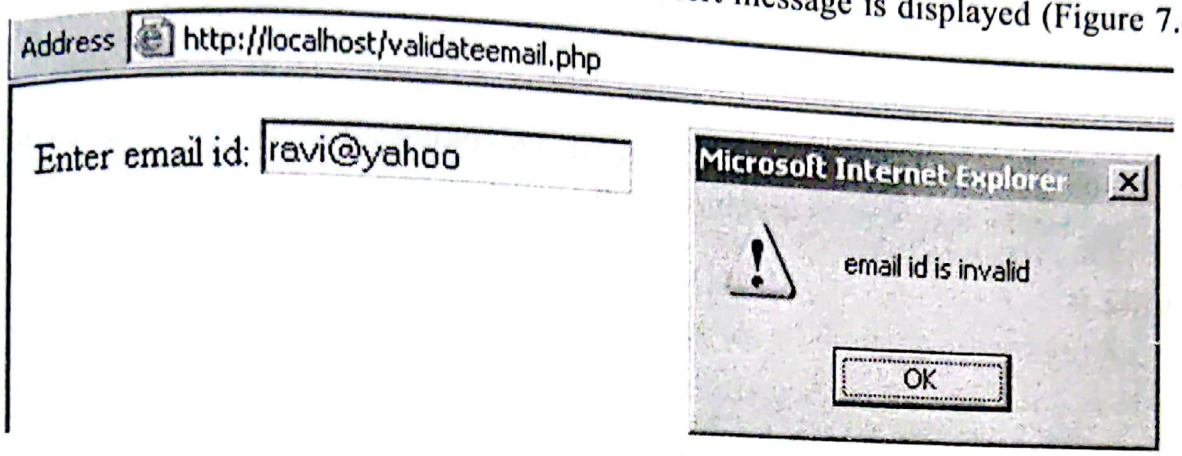


Figure 7.6 ***Alert Message is Displayed if dot(.) Not Added in the Address***

If the email id a valid email id, we find no errors (Figure 7.7).

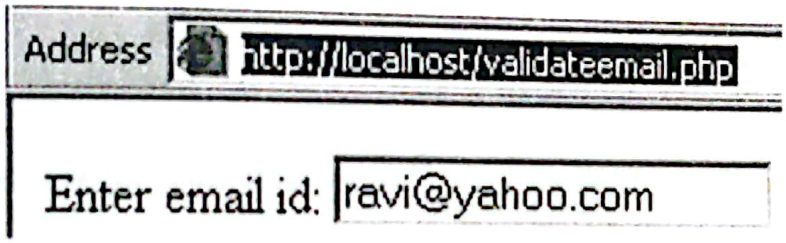


Figure 7.7 ***No Error Message is Displayed***

7.3.1 Validating email id—with Regular Expression

In this method, we check the validity of the email id with the help of regular expressions. We make two files named validateemail2.php and checkemail2.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validateemail2.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="checkemail2.js"></
   script>
</head>
<body>
Enter email id: <input size="20" type="text" name="emailid" onblur
="validatefield(this)">
   </form>
```

```
</body>
</html>
```

This program is the same as the earlier program. It is invoking `validatefield()` method with email id entered by the user sent as argument to it.

checkemail2.js

```
function validatefield(obj)
{
        regex=/(^\w{2,}\@\w{2,}\.\w{2,})/;
        match=regex.exec(obj.value);
        if(!match)
        {
        alert("email id is invalid");
        obj.focus();
}
}
```

The object (email id) passed to the function is matched with a regular expression. If the email id doesn't match the regular expressions, an alert message is displayed. The regular expression:

```
regex=/(^\w{2,}\@\w{2,}\.\w{2,})/;
```

means that an email id must have two or more alphanumerical characters followed by @ (at) symbol and again two or more alphanumerical characters followed by . (dot) symbol which is again followed by two or more alphanumerical characters.

second way of writing `validatefield()` method:

```
function validatefield(obj)
{
            regex=/(^\w+\@\w+\.\w+)/;
            match=regex.exec(obj.value);
            if(!match)
            {
            alert("email id is invalid");
            obj.valid=false;
            obj.focus();
}
}
```

In this function, the regular expression is just changed. The regular expression:

```
regex=/(^\w+\@\w+\.\w+)/;
```

means that an email id must have one or more alphanumerical characters followed by a @ (at) symbol and again one or more alphanumerical characters followed by a . (dot) symbol which is again followed by one or more alphanumerical characters.

The output is the same as in the earlier example. For an invalid email, the error message: "email id is invalid" is displayed (Figure 7.8).

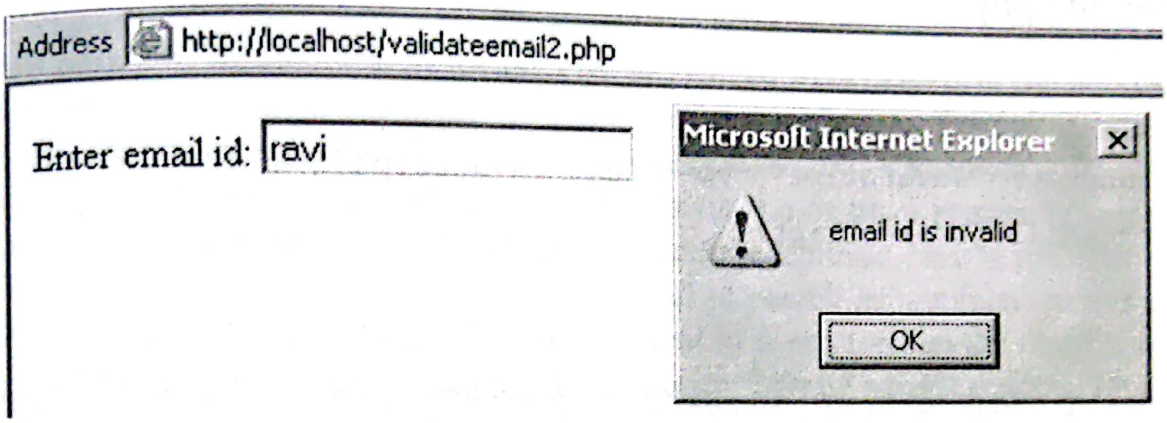


Figure 7.8 *Alert Message for Invalid Email id*

If . (dot) is missing, again the email id is considered to be an invalid email and hence, the error message: "email id is invalid" is displayed (Figure 7.9).

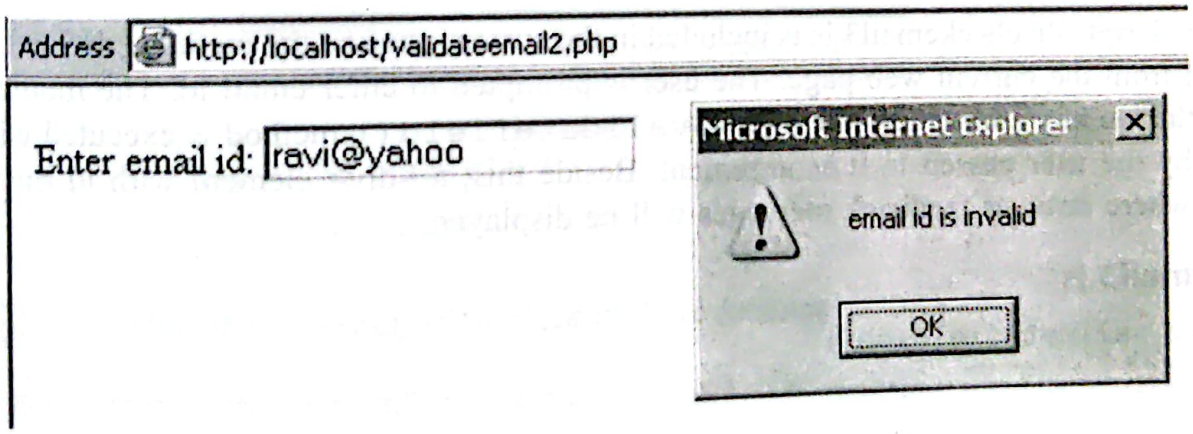


Figure 7.9 *Alert Message for Invalid Email id*

No error message is displayed for a valid email id and it is accepted (Figure 7.10).

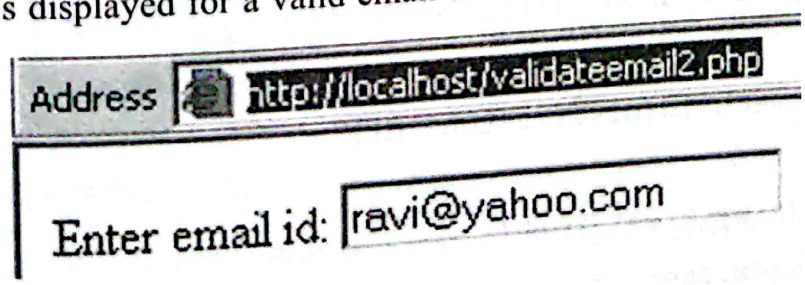


Figure 7.10 *No Error Message is Displayed*

7.3.2 Validating email id—Using DOM

In the following example, instead of displaying error in the form of an alert message, we display it as text on the web page. That is, we define a element on the web page which will be used for displaying error or other feedback messages to the user. We make two files named validateemail3.php and checkemail3.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validateemail3.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript"
src="checkemail3.js"></script>
</head>
<body>
Enter email id: <input size="20" type="text" name="emailid" onblur
="validatefield(this)"> <div id="emailmsg">
</body>
</html>
```

A JavaScript file checkemail3.js is included in the current page so the methods defined in it can be accessed from the current web page. The user is prompted to enter email id. The moment the user presses the tab key after entering email id, `validatefield()` method is executed with email id entered by the user passed to it as argument. Beside this, a element with id `emailmsg` is defined where error or feedback messages will be displayed.

checkemail3.js

```
function validatefield(obj)
{
        var div=document.getElementById("emailmsg");
        div.style.color="red";
        if(div.hasChildNodes())
        {
        div.removeChild(div.firstChild);
        }
        regex=/(^\w+\@\w+\.\w+)/;
        match=regex.exec(obj.value);
```

```
if(!match)
{
div.appendChild(document.createTextNode("Invalid Email"));
obj.focus();
}
}
```

In the above program, we first search for the element named `emailmsg` in the document (web page). The div element `emailmsg` is made for displaying error messages. Set that element's colour to "red" (error messages are usually displayed in red to draw the user's attention). Then, we check if there are already some child nodes of the current element (i.e. if there is already some message displayed at the current element). If so, then that child node is removed (that is the earlier message if any is removed). After that, the object value i.e. email id passed as argument to this method is matched with the regular expression provided. If the email id doesn't match the regular expression, then a child in the form of text node displaying the message: "Invalid email" is added (appended) to the element `emailmsg` and focus is set to the email id field so that the user can enter a valid email id.

Output:

If an invalid email id is entered, an error message is displayed at the place of `<div>` element "emailmsg" instead of in the form of an alert message (Figure 7.11).

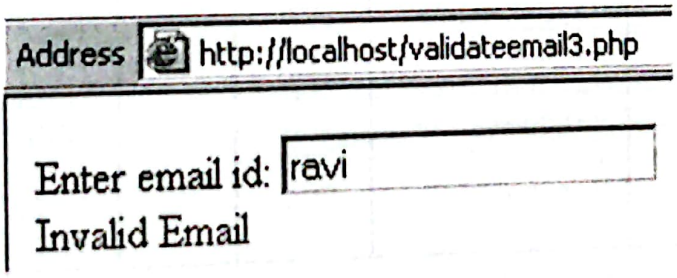


Figure 7.11 ***Error Message is Displayed in Place of Alert Message***

If . (dot) is not present in the email id, again an error message is displayed (Figure 7.12).

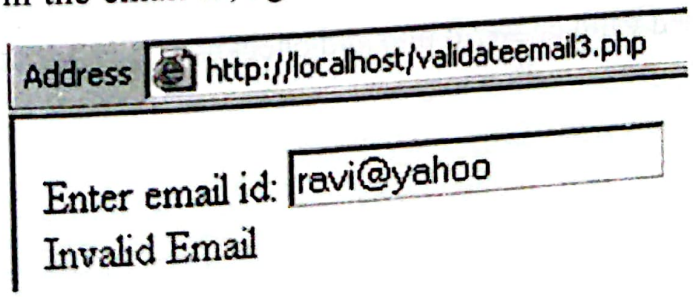


Figure 7.12 ***Error Message is Displayed***

If the email id has both @ (at) and . (dot) that means, the email id is valid and hence no error messages arise (Figure 7.13).

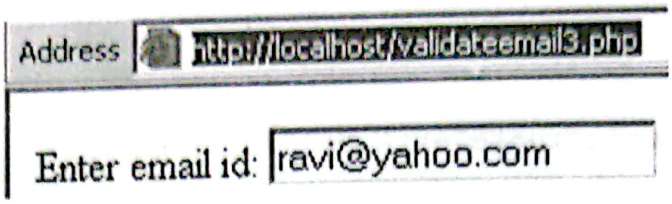


Figure 7.13 *No Error Message is Displayed*

7.4 VALIDATING USERID—CHECKING IF IT ALREADY EXISTS

In the following example, we ask the user to enter a userid and if that userid already exists (that is found in the `customers` table of shopping database), an error message is displayed and the user is asked to enter another userid. The `shopping` database and `customers` table were created in "Shopping Cart Project" in Part I of this book. For reference, the structure of the `customers` table is provided (Figure 7.14).

```
mysql> describe customers;
+--------------+---------------+------+-----+---------+-------+
| Field        | Type          | Null | Key | Default | Extra |
+--------------+---------------+------+-----+---------+-------+
| userid_code  | Varchar<50>   | NO   | PRI |         |       |
| first_name   | Varchar<50>   | YES  |     | NULL    |       |
| last_name    | Varchar<50>   | YES  |     | NULL    |       |
| add1         | varchar<255>  | YES  |     | NULL    |       |
| add2         | varchar<255>  | YES  |     | NULL    |       |
| city         | Varchar<50>   | YES  |     | NULL    |       |
| state        | Varchar<50>   | YES  |     | NULL    |       |
| zipcode      | Varchar<12>   | YES  |     | NULL    |       |
| emailid      | varchar<50>   | YES  |     | NULL    |       |
| password     | Varchar<50>   | YES  |     | NULL    |       |
| phone_no     | Varchar<50>   | YES  |     | NULL    |       |
| country      | Varchar<50>   | YES  |     | NULL    |       |
+--------------+---------------+------+-----+---------+-------+
12 rows in set <0.28 sec>
```

Figure 7.14 *Structure of the Table Customers*

We make two files named validateuserid.php and checkuserid.js with the following contents in the `htdocs` subfolder of the `apache` directory.

validateuserid.php

```
<html>
<head>
<script language="JavaScript" type="text/JavaScript" src="checkuserid.js"></script>
</head>
<body>
```

```
Enter user id: <input size="20" type="text" name="userid" onblur
="validatefield(this.value)"> <div id="uidmsg">
</body>
</html>
```

A JavaScript file checkuserid.js is included in the current page for using the methods defined in it from the current web page. The user is prompted to enter userid which is then validated with the help of `validatefield()` method that is invoked and userid entered by the user is passed to it as argument. Also, a element with id `uidmsg` is defined where error or feedback messages can be displayed.

checkuserid.js

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

18. function validatefield(user)
19. {
20. var div=document.getElementById("uidmsg");
21. div.style.color="red";
```

```
22. if(div.hasChildNodes())
23. {
24. div.removeChild(div.firstChild);
25. }
26. var xmlhttp=makeRequestObject();
27. xmlhttp.open('GET', 'ajaxcheckuser.php?userid='+user, true);
28. xmlhttp.onreadystatechange=function() {
29. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
30. var content = xmlhttp.responseText;
31. if( content ){
32. div.innerHTML = content;
33. }
34. }
35. }
36. xmlhttp.send(null);
37. }
```

Explanation of checkuserid.js program

Statements 1–17 are making an XMLHttpRequest object.

```
20. var div=document.getElementById("uidmsg");
21. div.style.color="red";
22. if(div.hasChildNodes())
23. {
24. div.removeChild(div.firstChild);
25. }
```

We first find the element in the document (web page) with id `uidmsg` (the element defined for displaying error messages). Set that element's colour to "red" (error messages are usually displayed in red to draw the user's attention). Then, we check if it has already some child nodes (i.e if there is already some message displayed at this element). If it is so, then that child node is removed (that is, the earlier message if any is removed).

```
27. xmlhttp.open('GET', 'ajaxcheckuser.php?userid='+user, true);
28. xmlhttp.onreadystatechange=function() {
```

```
29. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
30. var content = xmlhttp.responseText;
31. if( content ){
32. div.innerHTML = content;
33. }
34. }
35. }
36. xmlhttp.send(null);
37. }
```

An XMLHttp request is made to the server (by GET method) for the file ajaxcheckuser.php, sending the userid entered by the user to it. The state and the response to asynchronous request are checked. If the value of `readyState` property becomes `4` and the value of the status of the HTTP Request is `200`, the response is then retrieved from the response stream and is assigned to variable `content` which is then applied to the innerHTML property of the element with id `uidmsg` to display the response from the server (i.e. to display the error message if the userid already exists in the `customers` table).

The instruction `xmlhttp.send(null)` is used to actually send the request to the server. If we don't want to pass any query string with the requested file, the argument passed to this method is null or else it contains the query string.

Remember, the actual checking of the userid is done in the file ajaxcheckuser.php (on server). And it is this file which returns the error message to the client if the userid already exists. Let's analyse how this file works.

ajaxcheckuser.php

```
<?php
$uid = $_GET["userid"];
mysql_connect("localhost", "root", "mce");
mysql_select_db("shopping");
$sql = "SELECT * FROM customers WHERE userid like '" . $uid . "'";
$result = mysql_query($sql) or die(mysql_error());
if (mysql_num_rows($result) == 1) {
echo "User already exist. Try another userid";
}
?>
```

In the above program, the userid passed is stored in a variable `$uid`. Then, a connection with MySQL server is established by user name `root` and password `mce`. Thereafter, the "shopping" database is selected and a SQL query is executed to find all the rows in the table `customers` with userid = `$uid` (that is, the userid entered by the user). If the query results in even a single row, it means there is already a record of a user in the `customers` table with the given userid. Hence an error message is returned to the client which is then pasted at the element `uidmsg`.

Output:

If we enter a userid which already exists in the `customers` table, we get an error (Figure 7.15).

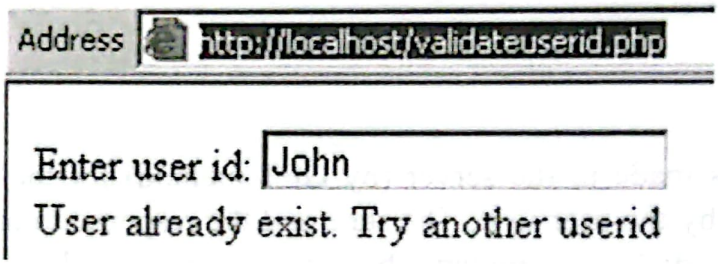


Figure 7.15 *An Error Message is Displayed if Userid Already Exists*

If the userid entered doesn't exist already, no error appears (Figure 7.16).

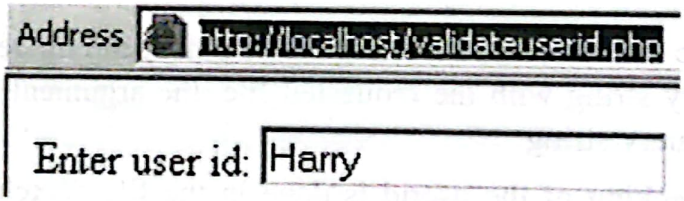


Figure 7.16 *No Error Message is Displayed if Userid Doesn't Exist Already*

7.5 VALIDATING FORM AND STORING THE RECORD IN THE TABLE (IF DATA IS VALID)

This is the best example to understand how validation is implemented in commercial web applications, because here instead of validating one element, we are validating three elements together and not only that, if the values in these elements are validated, that data will be stored in the `customers` table of shopping database (used in the earlier example).

validateform.php

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

```
</head>
<body>
  Userid: <input size="20" type="text" name="userid" onblur
="validateuid(this)"> <div id="uidmsg"></div><br>
  password: <input size="20" type="password" name="password" onblur
="if(this.value !='') {validatepwd(this)}"> <div id="pwdmsg"></div><br>
Enter email id: <input size="20" type="text" name="emailid" onblur
="if(this.value !='') {validateemail(this)}"> <div id="emailmsg"></div>
<div id="info"></div>
</body>
</html>
```

This program is including checkform.js file so as to invoke validation methods (stored in it) from this web page. This form asks the user to provided the following information: userid, password and email id. Since we don't want to display alert message for any mistake committed by the user, instead we want the error messages to be displayed in the form of text on the web page, so we define three elements named `uidmsg`, `pwdmsg` and `emailmsg` to display error message for each element and place them adjacent to their respective elements on the form. Beside this, an element with id `info` is also defined in order to display any other server response.

checkform.js

```
1. function validateuid(obj)
2. {
3. var div=document.getElementById("uidmsg");
4. div.style.color="red";
5. if(div.hasChildNodes())
6. {
7. div.removeChild(div.firstChild);
8. }
9. if(obj.name=="userid")
10. {
11. if(obj.value.length ==0)
12. {
13. div.appendChild(document.createTextNode("User id cannot be blank"));
14. obj.focus();
```

```
15. }
16. }
17. }

18. function validatepwd(obj)
19. {
20. div=document.getElementById("pwdmsg");
21. div.style.color="red";
22. if(div.hasChildNodes())
23. {
24. div.removeChild(div.firstChild);
25. }
26. if(obj.name=="password")
27. {
28. if(obj.value.length <=5)
29. {
30. div.appendChild(document.createTextNode("The password should be of at least
    size 6"));
31. obj.focus();
32. }
33. }
34. }

35. function validateemail(obj)
36. {
37. div=document.getElementById("emailmsg");
38. div.style.color="red";
39. if(div.hasChildNodes())
40. {
41. div.removeChild(div.firstChild);
42. }
43. regex=/(^\w+\@\w+\.\w+)/;
44. match=regex.exec(obj.value);
```

```
45. if(!match)
46. {
47. div.appendChild(document.createTextNode("Invalid Email"));
48. obj.focus();
49. }
50. else
51. {
52. insertrec();
53. }
54. }

55. function makeRequestObject(){
56. var xmlhttp=false;
57. try {
58. xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');
59. } catch (e) {
60. try {
61. xmlhttp = new
62. ActiveXObject('Microsoft.XMLHTTP');
63. } catch (E) {
64. xmlhttp = false;
65. }
66. }
67. if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
68. xmlhttp = new XMLHttpRequest();
69. }
70. return xmlhttp;
71. }

72. function insertrec()
73. {
74. var xmlhttp=makeRequestObject();
75. user=document.getElementById('userid').value;
```

```
76. pswd=document.getElementById('password').value;
77. eml=document.getElementById('emailid').value;
78. xmlhttp.open('GET', 'insertdata.php?uid='+user+'&pwd='+pswd+'&email='+eml,
    true);
79. xmlhttp.onreadystatechange=function() {
80. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
81. var content = xmlhttp.responseText;
82. if( content ){
83. document.getElementById('info').innerHTML = content;
84. }
85. }
86. }
87. xmlhttp.send(null)
88. }
```

Explanation of checkform.js program

Meaning of statements 1–17

These instructions are finding an element in the document (web page) with id `uidmsg` (the element defined for displaying error messages). The colour of this element is set to "red" and we check if it has already some child nodes (meaning if there is already some message displayed at this element). If it is so, it is removed (i.e. the earlier message if any is removed). Thereafter, it is checked if the object sent to this function is userid and if yes, its length is checked. If the length of userid is 0 (meaning the user hasn't entered anything in that field), then a text node containing a text message: "User id cannot be blank" is appended to this `uidmsg` element so that the text appears at this element and focus is set on the userid element so the user can re-enter a valid userid.

Meaning of statements 18–34

These instructions are finding an element in the document with id `pwdmsg` and its colour is set to "red". It is checked whether there is already any message displayed in this element (whether it has any ChildNodes). If yes, then the message is deleted (ChildNode is removed). Then it is assured if the object sent to this function is password and if it is, then its length is checked. If the length of password is less than 6 characters, then a text node containing a text message: "The password should be of at least size 6" is appended to this `pwdmsg` element so that the error message appears at this element and focus is set on password element so that the user can re-enter a valid password.

Meaning of statements 35–54

These instructions are finding an element in the document with id `emailmsg` and again its colour is set to "red". Any earlier message displayed in this element is deleted (i.e. its ChildNodes if any are removed). The email id passed to this method is matched with a regular expression. The regular expression:

```
regex=/(^\w+\@\w+\.\w+)/
```

means that an email id must have one or more alphanumerical characters followed by a @ (at) symbol and one or more alphanumerical characters followed by a . (dot) symbol which is again followed by one or more alphanumerical characters. If the email id matches with the regular expression, then `insertrec()` method is executed for inserting the valid information passed by the user in the `customers` table and if email id doesn't match with the regular expression, then a text node containing a text message "Invalid email" is appended to the `emailmsg` element so that an error message is displayed at this element and focus is set on the email id element allowing the user to enter a valid email id.

Statements 55–87 are making an XMLHttpRequest object.

```
72. function insertrec()
73. {
74. var xmlhttp=makeRequestObject();
75. user=document.getElementById('userid').value;
76. pswd=document.getElementById('password').value;
77. eml=document.getElementById('emailid').value;
78. xmlhttp.open('GET', 'insertdata.php?uid='+user+'&pwd='+pswd+'&email='+eml,
    true);
79. xmlhttp.onreadystatechange=function() {
80. if (xmlhttp.readyState==4 && xmlhttp.status == 200) {
81. var content = xmlhttp.responseText;
82. if( content ){
83. document.getElementById('info').innerHTML = content;
84. }
85. }
86. }
87. xmlhttp.send(null)
88. }
```

All the values entered by the user in form elements userid, password and emailid are retrieved and stored in variables `user`, `pswd` and `eml` respectively. Then an XMLHttp request is made to the server (by GET method) for the file insertdata.php, sending these three values to it (it is this file that will insert the information passed by the user into the `customers` table). The state and the response

to asynchronous request are checked. If the value of readyState property becomes 4 and the value of the status of the HTTP Request is 200, the response is then retrieved from the response stream and is assigned to variable `content` which is then applied to the innerHTML property of the element with id `info` to display the response from the server. The instruction `xmlhttp.send(null)` is used to actually send the request to the server and since we don't want to pass any query string to this method, its argument is set to null.

Remember, the actual inserting of user information and returning of server response is done in file insertdata.php. Let's analyse how this file works.

insertdata.php

```
<?php
mysql_connect("localhost", "root", "mce");
mysql_select_db("shopping");
$uid = $_GET["uid"];
$pwd = $_GET["pwd"];
$email = $_GET["email"];
$sql = "insert into customers(userid, password, emailid) values('" . $uid . "',
'" . $pwd . "', '" . $email . "')";
$query = mysql_query($sql) or die(mysql_error());
echo 'Record Inserted';
?>
```

This program is first establishing connection with the MySQL server and then "shopping" database is selected. The userid, password and email id passed to this file are retrieved using `$_GET` array and stored in variables `$uid`, `$pwd` and `$email` respectively. An SQL statement is executed to insert the information passed by the user in `customers` table. If the insertion is successful, the server response in the form of text "Record Inserted" is passed to the client which is then displayed on the web page at the place of element of id: `info`.

Output:

If user id is left blank, we get an error "User id cannot be blank" (Figure 7.17).

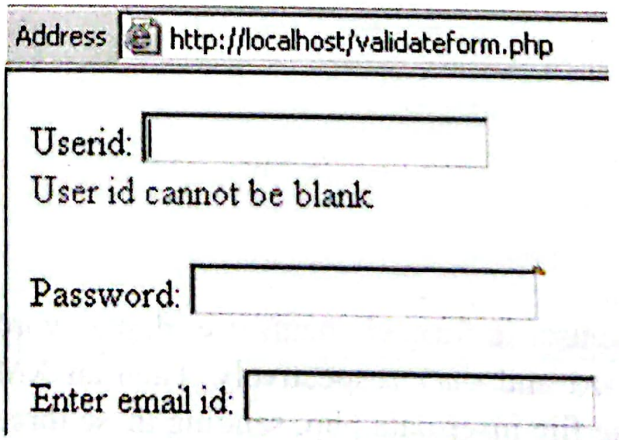


Figure 7.17 *Error Message is Displayed if Userid is Left Blank*

If password entered is of less than 6 characters, we get an error "The password should be of at least size 6" (Figure 7.18).

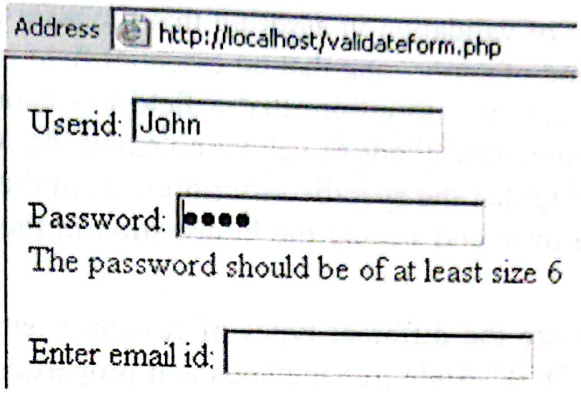


Figure 7.18 ***Error Message is Displayed if the Password is Less than 6 Characters***

If email id is of length 0 or doesn't have . (dot) or @ (at) symbol in it, an error message: "Invalid Email" is displayed (Figure 7.19).

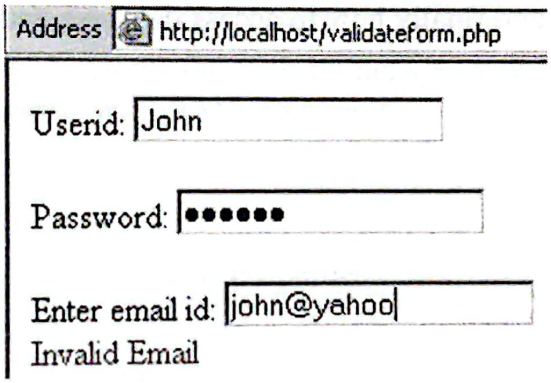


Figure 7.19 ***Error Message is Displayed if Invalid email id is Given***

If all the information passed is valid, the record is inserted in the `customers` table and the message "Record Inserted" is displayed (Figure 7.20).

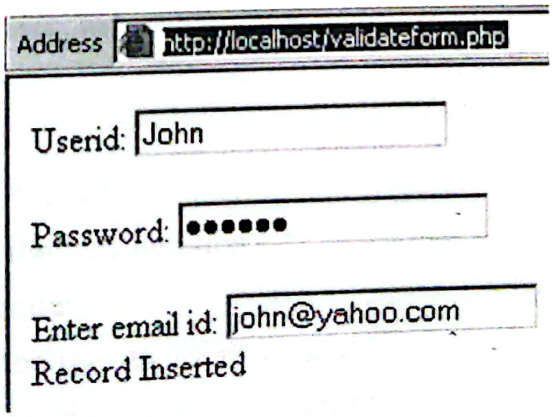


Figure 7.20 ***If No Errors then Record Inserted is Displayed***

SUMMARY

In this chapter, we saw the role of validation in assuring that the information entered by the user is correct. We have seen the method of confirming that no essential field is left blank by the user while entering information. We also saw how to ensure that a field has enough data (minimum length). Also, we have seen the three different methods of validating email ids. We also saw the technique of validating a userid i.e. confirming that the specified userid exists in the given database. Finally, we saw the method of validating a form and storing the information in the table (if data in the form is found valid).

In the next chapter we will see the different types of response data retrieved from the server. Also, we will look at different DOM Node tree methods and properties and how XML response is received. We will study several programs that include the technique to display the contents of the first child, values (text) of all the child nodes of the given node, contents of the all the child nodes along with the names of the child nodes and contents of the selected nodes. We will have a good understanding of JSON i.e. what it is and how it is installed, how to receive data in JSON format, how to access the response through iteration, how to use library for encoding, how to convert data entered by user to JSON format and finally how to access database and display its contents in JSON format.