UNIT III

3.1 Reading data using Form Controls

(Text Fields, Text Areas, Checkboxes, Radio Buttons, List Boxes, Password Controls, Hidden Controls, Image Maps, File Uploads, Buttons)

Handling forms is a multipart process. First a form is created, into which a user canenter the required details. This data is then sent to the web server, where it is interpreted,often with some error checking. If the PHP code identifies one or more fieldsthat require reentering, the form may be redisplayed with an error message. Whenthe code is satisfied with the accuracy of the input, it takes some action that usuallyinvolves the database, such as entering details about a purchase.

To build a form, you must have at least the following elements:

• An opening <form> and closing </form> tag
• A submission type specifying either a Get or Post method
• One or more input fields
• The destination URL to which the form data is to be submitted

```
<?php // formtest.php
echo <<<_END
<html>
<head>
<title>Form Test</title>
</head>
<body>
<form method="post" action="formtest.php">
What is your name?
<input type="text" name="name">
<input type="submit">
</form>
</body>
</html>
_END;
?>
```

o/p

What is your name? Submit Query

using the <<<_END..._END; heredoc construct, youdon't have to add \n linefeed characters to send a linefeed—just press Return and start a new line. Also, unlike either a doublequote-or single-quote-delimited string, you are free to use all thesingle and double quotes you like within a heredoc, without escapingthem by preceding them with a slash (\).

```
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
```

o/p

What is your name? Name: E-mail:

Inside of this multiline output is some standard code for commencing an HTMLdocument, displaying its title, and starting the body of the document. This is followedby the form, which is set to send its data using the Post method to the PHP program*formtest.php*, which is the name of the program itself.

The rest of the program just closes all the items it opened: the form, the body of theHTML document, and the PHP echo <<<_END statement.

Input Types

HTML forms are very versatile and allow you to submit a wide range of input types,from text boxes and text areas to checkboxes, radio buttons, and more.

Text boxes

The input type you will probably use most often is the text box. It accepts a widerange of alphanumeric text and other characters in a single-line box. The general formatof a text box input is as follows:

<input type="text" name="*name*" size="*size*" maxlength="*length*" value="*value*">

The size attribute specifies the width of the box (in charactersof the current font) as it should appear on the screen, and maxlength specifiesthe maximum number of characters that a user is allowed to enter into the field.
The only required attributes are type, which tells the web browser what type of inputto expect, and name, for giving the input a name that will be used to process the fieldupon receipt of the submitted form.

Text areas

When you need to accept input of more than a short line of text, use a text area. Thisis similar to a text box, but, because it allows multiple lines, it has some differentattributes. Its general format looks like this:

<textarea name="*name*" cols="*width*" rows="*height*" wrap="*type*">
</textarea>

The first thing to notice is that <textarea> has its own tag and is not a subtype of the<input> tag. It therefore requires a closing </textarea> to end input.Instead of a default attribute, if you have default text to display, you must put it beforethe closing </textarea>, and it will then be displayed and be editable by the user:

<textarea name="*name*" cols="*width*" rows="*height*" wrap="*type*">
This is some default text.
</textarea>

To control the width and height, use the cols and rows attributes. Both use the characterspacing of the current font to determine the size of the area. If you omit thesevalues, a default input box will be created that will vary in dimensions depending onthe browser used, so you should always define them to be certain about how yourform will appear.Last, you can control how the text entered into the box will wrap (and how any suchwrapping will be sent to the server) using the wrap attribute. Table 11-1 shows thewrap types available. If you leave out the wrap attribute, soft wrapping is used.

TYPE	**ACTION**
off	Text does not wrap, and lines appear exactly as the user types them.
soft	Text wraps but is sent to the server as one long string without carriage returns and line feeds.
hard	Text wraps and is sent to the server in wrapped format with soft returns and line feeds.

Checkboxes

When you want to offer a number of different options to a user, from which he canselect one or more items, checkboxes are the way to go. Here is the format to use:

<input type="checkbox" name="*name*" value="*value*" checked="checked">

If you include the checked attribute, the box is already checked when the browser isdisplayed. The string you assign to the attribute should be either a double quote orthe value "checked", or there should be no value assigned. If you don’t include theattribute, the box is shown unchecked. Here is an example of creating an unchecked
box: I Agree <input type="checkbox" name="agree">

If the user doesn't check the box, no value will be submitted. But if he does, a value of"on" will be submitted for the field named agree. If you prefer to have your ownvalue submitted instead of the word *on* (such as the number 1), you could use thefollowing syntax:

```
I Agree <input type="checkbox" name="agree" value="1">
```

On the other hand, if you wish to offer a newsletter to your readers when submittinga form, you might want to have the checkbox already checked as the default value:

```
Subscribe? <input type="checkbox" name="news" checked="checked">
```

If you want to allow groups of items to be selected at one time, assign them all thesame name. However, only the last item checked will be submitted, unless you passan array as the name.

Example 11-4. Offering multiple checkbox choices

```
Vanilla <input type="checkbox" name="ice" value="Vanilla">
Chocolate <input type="checkbox" name="ice" value="Chocolate">
Strawberry <input type="checkbox" name="ice" value="Strawberry">
```

If only one of the checkboxes is selected, such as the second one, only that item willbe submitted (the field named ice would be assigned the value "Chocolate"). But iftwo or more are selected, only the last value will be submitted, with prior values beingignored.

If you *want* exclusive behavior—so that only one item can be submitted—then youshould use *radio buttons* but to allow multiple submissions,you have to slightly alter the HTML, as in Example 11-5 (note the addition of the square brackets, [], following the values of ice).

Example 11-5. Submitting multiple values with an array

```
Vanilla <input type="checkbox" name="ice[]" value="Vanilla">
Chocolate <input type="checkbox" name="ice[]" value="Chocolate">
Strawberry <input type="checkbox" name="ice[]" value="Strawberry">
```

Now, when the form is submitted, if any of these items have been checked, an arraycalled ice will be submitted that contains any and all values. In each case, you canextract either the single submitted value, or the array of values, to a variable like this:

```
$ice = $_POST['ice'];
```

If the field ice has been posted as a single value, $ice will be a single string, such as"Strawberry". But if ice was defined in the form as an array (like Example 11-5),$ice will be an array, and its number of elements will be the number of values submitted.Table 11-2 shows the seven possible sets of values that could be submitted bythis HTML for one, two, or all three selections. In each case, an array of one, two, orthree items is created.

Table 11-2. The seven possible sets of values for the array $ice

One Value Subitted	**One Value Subitted**	**One Value Subitted**
$ice[0] => Vanilla	$ice[0] => Vanilla	$ice[0] => Vanilla
$ice[1] => Chocolate	$ice[1] => Chocolate	
$ice[2] => Strawberry		
$ice[0] => Chocolate	$ice[0] => Vanilla	
$ice[1] => Strawberry		
$ice[0] => Strawberry	$ice[0] => Chocolate	
$ice[1] => Strawberry		

If $ice is an array, the PHP code to display its contents is quite simple and might looklike this:

```
foreach($ice as $item) echo "$item<br>";
```

This uses the standard PHP foreach construct to iterate through the array $ice andpass each element's value into the variable $item, which is then displayed via the echocommand. The <br> is just an HTML formatting device to force a new line after eachflavor in the display. By default, checkboxes are square.

Radio buttons

Radio buttons are named after the push-in preset buttons found on many olderradios, where any previously depressed button pops back up when another is pressed.They are used when you want only a single value to be returned from a selection oftwo or more options. All the buttons in a group must use the same name and,
because only a single value is returned, you do not have to pass an array.For example, if your website offers a choice of delivery times for items purchasedfrom your store, you might use HTML like that in Example 11-6 (see Figure 11-5 tosee how it displays).

Example 11-6. Using radio buttons

```
8am-Noon<input type="radio" name="time" value="1">
Noon-4pm<input type="radio" name="time" value="2" checked="checked">
4pm-8pm<input type="radio" name="time" value="3">
```

Here, the second option of Noon–4pm has been selected by default. This defaultchoice ensures that at least one delivery time will be chosen by the user, which shecan change to one of the other two options if she prefers. Had one of the items notbeen already checked, the user might forget to select an option, and no value would
be submitted at all for the delivery time. By default, radio buttons are round.

Hidden fields

Sometimes it is convenient to have hidden form fields so that you can keep track ofthe state of form entry. For example, you might wish to know whether a form hasalready been submitted. You can achieve this by adding some HTML in your PHPcode, such as the following:

```
echo '<input type="hidden" name="submitted" value="yes">'
```

This is a simple PHP echo statement that adds an input field to the HTML form. Let'sassume the form was created outside the program and displayed to the user. The firsttime the PHP program receives the input, this line of code has not run, so there willbe no field named submitted. The PHP program re-creates the form, adding theinput field. So when the visitor resubmits the form, the PHP program receives it withthe submitted field set to "yes". The code can simply check whether the field ispresent:

```
if (isset($_POST['submitted']))
{...
```

Hidden fields can also be useful for storing other details, such as a session ID stringthat you might create to identify a user, and so on.

<select>

The <select> tag lets you create a drop-down list of options, offering either single ormultiple selections. It conforms to the following syntax:

<select name="*name*" size="*size*" multiple="multiple">

The attribute size is the number of lines to display. Clicking on the display causes alist to drop down, showing all the options. If you use the multiple attribute, a usercan select multiple options from the list by pressing the Ctrl key when clicking. So toask a user for his favorite vegetable from a choice of five, you might use HTML as in

Example 11-7, which offers a single selection.

```
Vegetables
<select name="veg" size="1">
<option value="Peas">Peas</option>
<option value="Beans">Beans</option>
<option value="Carrots">Carrots</option>
<option value="Cabbage">Cabbage</option>
<option value="Broccoli">Broccoli</option>
</select>
```

This HTML offers five choices, with the first one, *Peas*, preselected (due to it being thefirst item).
Figure 11-6 shows the output where the list has been clicked to drop it down,and the option *Carrots* has been highlighted. If you want to have a different default optionoffered first (such as *Beans*), use the <selected> tag, like this:

<option selected="selected" value="Beans">Beans</option>

You can also allow users to select more than one item, as in Example 11-8.

Example 11-8. Using select with the multiple attribute

```
Vegetables
<select name="veg" size="5" multiple="multiple">
<option value="Peas">Peas</option>
<option value="Beans">Beans</option>
<option value="Carrots">Carrots</option>
<option value="Cabbage">Cabbage</option>
<option value="Broccoli">Broccoli</option>
</select>
```

Labels

You can provide an even better user experience by utilizing the <label> tag. With it,you can surround a form element, making it selectable by clicking any visible partcontained between the opening and closing <label> tags.
For example, going back to the example of choosing a delivery time, you could allowthe user to click the radio button itself *and* the associated text, like this:

<label>8am-Noon<input type="radio" name="time" value="1"></label>

The text will not be underlined like a hyperlink when you do this, but as the mousepasses over, it will change to an arrow instead of a text cursor, indicating that thewhole item is clickable.

The submit button

To match the type of form being submitted, you can change the text of the submitbutton to anything you like by using the value attribute, like this:

<input type="submit" value="Search">

You can also replace the standard text button with a graphic image of your choice,using HTML such as this:

<input type="image" name="submit" src="image.gif">

Default Values

Sometimes it’s convenient to offer your site visitors a default value in a web form. Forexample, suppose you put up a loan repayment calculator widget on a real estate website.It could make sense to enter default values of, say, 25 years and 6 percent interest,so that the user can simply type either the principle sum to borrow or the amountthat she can afford to pay each month.
In this case, the HTML for those two values would be something like Example 11-3.

Example 11-3. Setting default values

```
<form method="post" action="calc.php"><pre>
Loan Amount <input type="text" name="principle">
Monthly Repayment <input type="text" name="monthly">
Number of Years <input type="text" name="years" value="25">
Interest Rate <input type="text" name="rate" value="6">
<input type="submit">
</pre></form>
```

3.2 Submitting form values, using $_*Get* and $_*Post* Methods, $_REQUEST

Superglobal variables

Starting with PHP 4.1.0, several predefined variables are available. These are knownas *superglobal variables*, which means that they are provided by the PHP environmentbut are global within the program, accessible absolutely everywhere.These superglobals contain lots of useful information about the currently running
program and its environment

$GLOBALS	All variables that are currently defined in the global scope of the script. The variable names are the keysof the array.
$_SERVER	Information such as headers, paths, and script locations. The entries in this array are created by the webserver, and there is no guarantee that every web server will provide any or all of these.
$_GET	Variables passed to the current script via the HTTP Get method.
$_POST	Variables passed to the current script via the HTTP Post method.
$_FILES	Items uploaded to the current script via the HTTP Post method.
$_COOKIE	Variables passed to the current script via HTTP cookies.
$_SESSION	Session variables available to the current script.
$_REQUEST	Contents of information passed from the browser; by default, $_GET, $_POST, and $_COOKIE.
$_ENV	Variables passed to the current script via the environment method.

All of the superglobals (except for $GLOBALS) are named with a single initial underscoreand only capital letters; therefore, you should avoid naming your own variablesin this manner to avoid potential confusion.

Retrieving Submitted Data
Example 11-1 is only one part of the multipart form-handling process. If you enter aname and click the Submit Query button, absolutely nothing will happen other thanthe form being redisplayed. So now it's time to add some PHP code to process thedata submitted by the form.
Example 11-2 expands on the previous program to include data processing. Type it ormodify *formtest.php* by adding in the new lines, save it as *formtest2.php*, and try theprogram for yourself. The result of running this program and entering a name isshown in Figure 11-2.
Example 11-2. Updated version of formtest.php

```
<?php // formtest2.php
if (isset($_POST['name'])) $name = $_POST['name'];
else $name = "(Not entered)";
echo <<<_END
<html><head>        <title>Form Test</title>        </head>        <body>
Your name is: $name<br>
<form method="post" action="formtest2.php">
What is your name?
<input type="text" name="name">
<input type="submit">
</form></body></html>_END;?>
```

The only changes are a couple of lines at the start that check the $_POST associative array for the field *name* having been submitted. Chapter 10 introduced the $_POST associative array, which contains an element for each field in an

HTML form. In Example 11-2, the input name used was *name* and the form method was Post, so element name of the $_POST array contains the value in $_POST['name'].
The PHP isset function is used to test whether $_POST['name'] has been assigned a value. If nothing was posted, the program assigns the value (Not entered); otherwise, it stores the value that was entered. Then a single line has been added after the<body> statement to display that value, which is stored in $name.

Sometimes it can be convenient to turn the key/value pairs from an array into PHP variables. One such time might be when you are processing the $_GET or $_POST variables as sent to a PHP script by a form.When a form is submitted over the Web, the web server unpacks the variables into a global array for the PHP script. If the variables were sent using the Get method, they will be placed in an associative array called $_GET; if they were sent using Post, they will be placed in an associative array called $_POST.

The $_POST Array
I mentioned in an earlier chapter that a browser sends user input through either a Get request or a Post request. The Post request is usually preferred (because it avoids placing unsightly data in the browser's address bar), and so we use it here. The web server bundles up all of the user input (even if the form was filled out with a hundred fields) and puts in into an array named $_POST.
$_POST is an associative array, which you encountered in Chapter 6. Depending on whether a form has been set to use the Post or the Get method, either the $_POST orthe $_GET associative array will be populated with the form data. They can both be read in exactly the same way.
Each field has an element in the array named after that field. So, if a form contained afield named isbn, the $_POST array contains an element keyed by the word isbn. ThePHP program can read that field by referring to either $_POST['isbn'] or$_POST["isbn"] (single and double quotes have the same effect in this case).
If the $_POST syntax still seems complex to you, rest assured that you can just use the convention I've shown in Example 10-6, copy the user's input to other variables, and forget about $_POST after that. This is normal in PHP programs: they retrieve all the fields from $_POST at the beginning of the program and then ignore it.

The forms can easily be changed to use the Get method, as long as values are fetched from the $_GET array insteadof the $_POST array.

$_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

```
<html><body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = htmlspecialchars($_REQUEST['fname']);
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;    }}          ?>       </body>         </html>
```

$_POST (PREVIOUS EXAMPLE USING POST)

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.
The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

$_GET (PREVIOUS EXAMPLE USING GET)

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".
$_GET can also collect data sent in the URL.
Assume we have an HTML page that contains a hyperlink with parameters:

```
<!DOCTYPE html>
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

3.3 Accessing form inputs with *Get/Post* functions

There is no reason to write to an element in the $_POST array. Its only purpose is to communicate information from the browser tothe program, and you're better off copying data to your own variables
before altering it.
So, back to the get_post function, which passes each item it retrieves through the real_escape_string method of the connection object to strip out any characters that a hacker may have inserted in order to break into or alter your database, like this:

```
function get_post($conn, $var){
return $conn->real_escape_string($_POST[$var]);   }
```

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.
$_POST is an array of variables passed to the current script via the HTTP POST method.

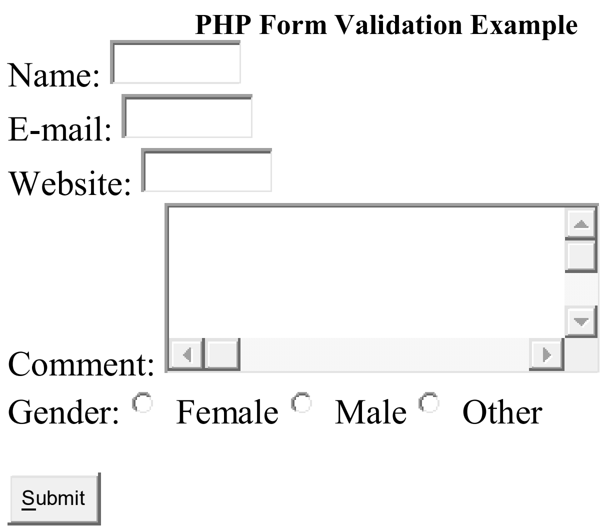
When to use GET?
Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
GET may be used for sending non-sensitive data.
Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?
Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.
Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
Developers prefer POST for sending form data.

3.4 Combining HTML and PHP codes together on single page, Redirecting the user

Data Validation: Checking data with built in PHP functions(Stripping unwanted characters, Testing network names, Dates, Escaping Shell commands, Escaping SQL queries)

Stripping unwanted characters



PHP Form Validation Example

Name:
E-mail:
Website:
Comment:
Gender: Female Male Other

Submit

Your Input:

```
<html><head>          </head>          <body>          <?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;}           ?>

<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">    <br><br>
  E-mail: <input type="text" name="email">   <br><br>
  Website: <input type="text" name="website">        <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>   <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other   <br><br>
  <input type="submit" name="submit" value="Submit">    </form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;        echo "<br>";   echo $gender;  ?></body>                </html>
```

Validate Form Data With PHP
The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:
<script>location.href('http://www.hacked.com')</script>
- this would not be executed, because it would be saved as HTML escaped code, like this:
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
The code is now safe to be displayed on a page or inside an e-mail.
We will also do two more things when the user submits the form:
Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).
We will name the function test_input().
Now, we can check each $_POST variable with the test_input() function,

The Form Element
The HTML code of the form looks like this:
```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```
When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!
If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

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

So far, so good.
However, consider that a user enters the following URL in the address bar:
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?
$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.
The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!
Required Fields:
PHP - Display The Error Messages
Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

```
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>

if (empty($_POST["name"])) {
   $nameErr = "Name is required";
  } else {
   $name = test_input($_POST["name"]);  }
```

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.
In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format"; }
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL"; }
```

Uploading Files

Uploading files to a web server is a subject that seems daunting to many people, but itactually couldn't be much easier. All you need to do to upload a file from a form ischoose a special type of encoding called *multipart/form-data*, and your browser willhandle the rest.

Example 7-15. Image uploader upload.php

```
<?php // upload.php
echo <<<_END
<html><head><title>PHP Form Upload</title></head><body>
<form method='post' action='upload.php' enctype='multipart/form-data'>
Select File: <input type='file' name='filename' size='10'>
<input type='submit' value='Upload'>
</form>
_END;

if ($_FILES)
{
$name = $_FILES['filename']['name'];
move_uploaded_file($_FILES['filename']['tmp_name'], $name);
echo "Uploaded image '$name'<br><img src='$name'>";
}
echo "</body></html>";
?>
```

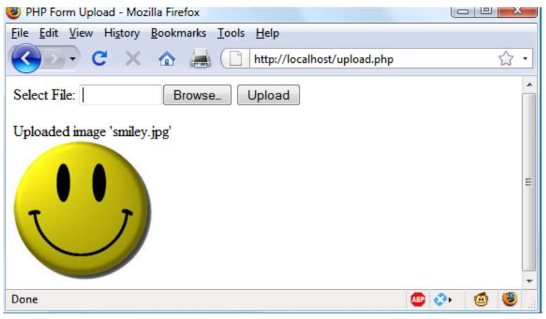
enctype='multipart/form-data': tells the web browser that the data posted should be encoded via the content MIME type of multipart/form-data.

Select File:<input type='file' name='filename' size='10'>

The first request is for a file; it uses an input type of file, a name of filename, and an input field with a width of 10 characters.

if ($_FILES)
all uploaded filesare placed into the associative system array $_FILES. Therefore, a quick check to seewhether $_FILES contains anything is sufficient to determine whether the user hasuploaded a file.
The first time the user visits the page, before uploading a file, $_FILES is empty, so theprogram skips this block of code. When the user uploads a file, the program runsagain and discovers an element in the $_FILES array.

Once the program realizes that a file was uploaded, the actual name, as read from theuploading computer, is retrieved and placed into the variable $name. Now all that'snecessary is to move the file from the temporary location in which PHP stored theuploaded file to a more permanent one. We do this using the move_uploaded_file function, passing it the original name of the file, with which it is saved to the currentdirectory.
Finally, the uploaded image is displayed within an IMG tag, and the result should lookLike


PHP Form Upload - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/upload.php
Select File:
Browse...
Upload
Done

Uploaded image 'smiley.jpg'

Using $_FILES
Five things are stored in the $_FILES array when a file is uploaded, as shown inTable 7-6 (where *file* is the file upload field name supplied by the submitting form).*Table 7-6. The contents of the $_FILES array*

Array element	**Contents**
$_FILES['*file*']['*name*']	The name of the uploaded file (e.g., *smiley.jpg*)
$_FILES['*file*']['*type*']	The content type of the file (e.g., *image/jpeg*)
$_FILES['*file*']['*size*']	The file's size in bytes
$_FILES['*file*']['*tmp_name*']	The name of the temporary file stored on the server
$_FILES['*file*']['*error*']	The error code resulting from the file upload

Content types used to be known as *MIME (Multipurpose Internet Mail Extension)*types, but because their use later expanded to the whole Internet, now they are oftencalled *Internet media types*.

Image Map

The <map> tag is used to define a client-side image-map. An image-map is an image with clickable areas.
The required name attribute of the <map> element is associated with the <img>'s usemap attribute and creates a relationship between the image and the map.
The <map> element contains a number of <area> elements, that defines the clickable areas in the image map.
The basic idea behind an image map is that you combine two different components:

- A map of defined linked areas
- An image

The map is overlaid on the image, and the clickable areas coincide with portions of the image. In HTML the image and the clickable areas are coded separately. However, from the visitor's perspective, it appears that portions of the image itself are linked to different destination.

HTML Elements Used to Create Image Maps

There are three HTML elements used to create image maps:
img: specifies the location of the image to be included in the map.
map: is used to create the map of clickable areas.
area: is used within the map element to define the clickable areas.

Step 1: Determine the size of our image
Our image is 1000 pixels wide by 664 pixels tall. However, in this example, we're going to use HTML to cause the image to display half that size: 500 by 332 pixels. When you create an image map it's important to remember that if you change the size of the image you will also have to change the area coordinates. This is because the area coordinates are tied to the original size and scale of the image. In order to render our image in the size we've selected

```
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">
```

Step 2: Create a map to overlay the image
The map code is quite simple. It looks like this:<map name="planetmap"></map>
Step 3: Define the coordinates for the map shapes
We can create that shape, or area, in an HTML map by using the following code:

```
<area href="https://facebook.com" alt="Facebook" target="_blank"  shape=poly coords="30,100, 140,50, 290,220, 180,280">
```

```
<html>          <body>
<p>Click on the sun or on one of the planets to watch it closer:</p>
<img src="planets.gif" width="145" height="126" alt="Planets" usemap="#planetmap">
<map name="planetmap">
<area shape="rect" coords="0,0,82,126" alt="Sun" href="sun.htm">
<area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm">
<area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm">
</map>          </body>         </html>
```

Password Controls

Testing network names, Dates, Escaping Shell commands, Escaping SQL queries)

UNIT IV
4.1 Setting a cookie with PHP, Deleting a cookie
4.2 Creating session cookie
4.3 Working with the query string Creating query string
4.4 Session
4.5 Working with session variables , Passing session IDs
4.6Session ID propagation

UNIT V

5.1 Concepts and Installation of MySQL
5.2 MySQL structure and syntax
5.3 Types of MySQL tables and Storage engines
5.4 MySQL commands
5.5 Integration of PHP with MySQL
5.6 Connection to the MySQL Database
5.7 Creating and Deleting MySQL database using PHP
5.8 Updating, Inserting, Deleting records in the MySQL database

UNIT VI

6.1 Creating and deleting a file
6.2 Reading and writing text files
6.3 Working with directories in PHP
6.4 Checking for existence of file
6.5 Opening a file for writing, reading, or appending
6.6 Writing Data to the file
6.7 Reading character