

Apress™

Books for Professionals by Professionals

Chapter Ten: “Forms”

A Programmer’s Introduction to PHP 4.0

by William Jason Gilmore

ISBN # 1-893115-85-2

Copyright ©2001 William J. Gilmore. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

info@apress.com

CHAPTER 10

Forms

The ability to retrieve and process user-provided information has become an integral part of most successful Web sites. The ability to collect statistics, poll users, store preferential information, and offer document searches certainly adds a whole new dimension to what would be an otherwise only minimally interactive medium.

Information retrieval is largely implemented through the use of HTML forms. Certainly you are already familiar with the look and feel of an HTML form. General practice is that you enter one or more pieces of data (for example, your name and email address), press a submit button of sorts, and are then greeted with a response message.

You may be thinking that the process of collecting user data via HTML forms is a complicated and tedious process. If so, you will be surprised to learn that it is actually quite easy.

An Introduction to Forms

There are a number of different forms you can use to input information. Some require users to enter information using their keyboard, while others require the users to select one or more choices by clicking with a mouse. Yet others simply involve a hidden form value that is embedded in the form itself and is not intended to be modified by the user.

It is possible to have multiple forms on the same page, so there must be some way to distinguish one form from the other. Furthermore, there must be a way to tell the form where to go once the user initiates the form action (usually by clicking a button). Both of these needs are taken care of by enclosing the form entities in the following HTML tags:

```
<form action="some_action" method="post">  
  ... form entities ...  
</form>
```

As you can see, two important elements make up this enclosure: the action and the method. The *action* specifies what script should process the form, while the *method* specifies how the form data will be sent to the script. There are two possible methods:

Chapter 10

- The get method sends all of the form information at the end of the URL. This method is rarely used, due to various language and length restrictions.
- The post method sends all of the form information in the request body. This method is usually preferred over get.

NOTE *This introduction is intended to be a brief primer regarding the basic syntax of HTML forms. For a more complete introduction, I suggest checking out Special Edition Using HTML 4 by Molly E. Holzschlag (QUE; ISBN 0789722674, December 1999).*

Keyboard-Oriented Form Entities

Now you're ready to begin building forms. The first step is to learn the keyboard-oriented form entities. There are only two: the text box and the text area box.

The Text Box

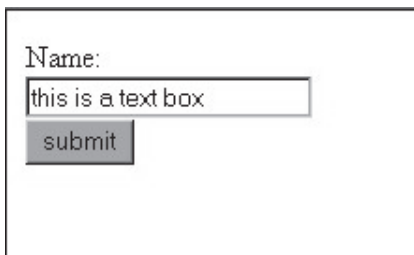
The text box is typically used for short text entries, such as an email address, postal address, or name. Its syntax is:

```
<input type="text" name="variable_name" size="N" maxlength="N" value="">
```

There are five text box components:

- **type:** Type of form input, in this case text.
- **name:** Variable name used to store the data.
- **size:** Total size of the text box as it will display in the browser.
- **maxlength:** Total number of characters that can be input into the text box.
- **value:** Default value that will display in the text box.

A sample text box is shown in Figure 10-1.



Name:

Figure 10-1. A text box

A variation on the text box is the password text box, which operates exactly like the text box, except that the data is hidden with asterisks as it is entered in the text field. To change the text box to a password text box, just use `type = "password"` instead of `type = "text"`.

The Text Area Box

The text area box is useful when you would like the reader to be a bit more verbose than just entering a name or email address. Its syntax is:

```
<textarea name="variable_name" rows="N" cols="N"></textarea>
```

There are three textarea components:

- name: Variable name used to store the data.
- rows: Number of rows comprising textarea.
- cols: Number of columns comprising textarea.

A sample text area box is shown in Figure 10-2.

Chapter 10

Message:

This is a text area box

submit

Figure 10-2. A text area box

Mouse-Oriented Form Entities

There are several other form entities that are controlled by the user selecting a predefined value with a mouse. I will limit the introduction to checkboxes, radio buttons, and pull-down menus.

The Checkbox

Checkboxes are convenient when you would like to present users with one or more choices to check, much like making a checkmark on some form with a pencil. The syntax is:

```
<input type="checkbox" name="variable_name" value="variable_value">
```

There are three checkbox components:

- **type:** Type of form input, in this case a checkbox.
- **name:** Variable name used to store the data, in this case the entity value.
- **value:** Default value that will be assigned to the variable name. Note that if the checkbox is checked, *this* is the value that is assigned to variable name. If it is not checked, then this variable will *not* be passed.

A sample checkbox is shown in Figure 10-3.

Choose your favorite soup
(check all that apply):

☐ vegetable

☐ wedding

☐ chicken

Figure 10-3. Checkboxes

The Radio Button

The radio button is a variation of the checkbox, similar in all aspects except that only one button can be checked. The syntax is:

```
<input type="radio" name="variable_name" value="variable_value">
```

As you can see, its syntax is exactly like that of the checkbox. There are three radio button components:

- type: Type of form input, in this case a radio.
- name: Variable name used to store the data, in this case the entity value.
- value: Default value that will display in the text box. Note that if the radio button is selected, *this* is the value that is assigned to variable name. If it is not selected, then this variable will *not* be passed.

Sample radio buttons are shown in Figure 10-4.

Choose your favorite soup (only one):

☐ vegetable

☐ wedding

☐ chicken

Figure 10-4. Radio buttons

Chapter 10

The Pull-Down Menu

Pull-down menus are particularly convenient when you have a long list of data from which you would like users to select a value. Pull-down menus are commonly used for large data sets, a list of American states or countries, for example. The syntax is:

```
<select name="variable_name">
<option value="variable_value1">
<option value="variable_value2">
<option value="variable_value3">
. . .
<option value="variable_valueN">
</select>
```

There are two pull-down menu components:

- **name:** Variable name used to store the data, in this case the variable name that will store the chosen value.
- **value:** Default value that will display in the text box. Note that if the checkbox is checked, *this* is the value that is assigned to variable name.

A sample pull-down menu is shown in Figure 10-5.

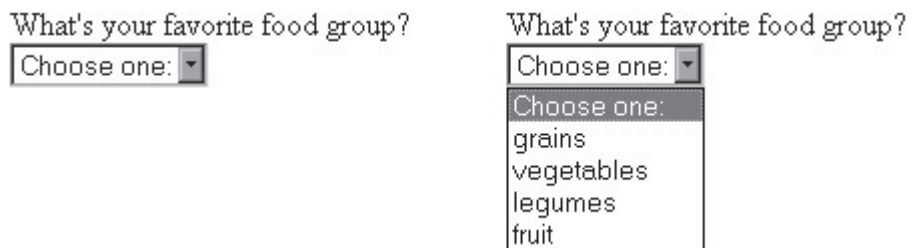


Figure 10-5. A pull-down menu

Hidden Values

Hidden form values are embedded in the form itself and are generally used as a means to persist data from one script to the other. While there is nothing wrong with doing so, PHP offers a much more convenient method for keeping persistent data: session tracking, which is the subject of Chapter 13. Regardless, hidden values have their uses, so I will proceed with the introduction.

The syntax of the hidden form value is exactly like that of the text box, save for the differing type value. Since the hidden value is hidden from the user, there is no way to show a sample. The syntax is:

```
<input type="hidden" name="variable_name" value="variable_value">
```

There are three hidden value components:

- type: Type of form input. In this case it's hidden.
- name: Variable name used to store the hidden data.
- value: Default value that will display in the text box.

Keep in mind that perhaps the title of this form entity is a misnomer. While the hidden value does not display to the browser, the user could simply perform a View Source and view whatever hidden values are in the form.

The Submit Button

The submit button actuates the action specified in the *action* component of the *form* enclosure. Its syntax is:

```
<input type="submit" value="button_name">
```

There are two submit button components:

- type: Type of form input, in this case submit.
- value: Default value that will display in the text box.

A sample submit button is shown in Figure 10-6.

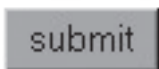


Figure 10-6. A submit button

Chapter 10

The Reset Button

The reset button will erase all information entered into the form. This is generally a pretty useless feature, but has become so commonly used in forms on the Web that I thought I should include it. The syntax is:

```
<input type="reset" name="reset" value="button_name">
```

There are two reset button components:

- **type:** Type of form input, in this case reset.
- **value:** Name shown on top of the button.

A reset button looks exactly like a submit button (illustrated in Figure 10-6), except that its type and value are set to “reset”.

NOTE *Jakob Nielsen, a noted Web-usability expert, recently wrote a rather interesting article about the drawbacks to the reset button. It is at <http://www.useit.com/alertbox/20000416.html>.*

Putting It Together: A Sample Form

Now that you have been introduced to the basic form components, you can create one that will accept user information. Suppose you wanted to create a form that would pose various questions to users about what they think about your new site. I'll create this form in Listing 10-1.

Listing 10-1: Sample user feedback form

```
<form action = "process.php" method = "post">
<b>Please take a moment to tell us what you think about our site:</b><p>
<b>Name:</b><br>
<input type="text" name="name" size="15" maxlength="25" value=""><br>
<b>Email:</b><br>
<input type="text" name="email" size="15" maxlength="45" value=""><br>
<b>How frequently do you visit our site?:</b><br>
<select name="frequency">
<option value="">Site frequency:
<option value="0">This is my first time
<option value="1">&lt; 1 time a month
<option value="2">Roughly once a month
<option value="3">Several times a week
```

```

<option value="4">Every day
<option value="5">I'm addicted
</select><br>
<b>I frequently purchase the following products from our site:</b><br>
<input type="checkbox" name="software" value="software">Software<br>
<input type="checkbox" name="cookware" value="cookware">Cookware<br>
<input type="checkbox" name="hats" value="hats">Chef's Hats<br>
<b>Our site's greatest asset is:</b><br>
<input type="radio" name="asset" value="products">Product selection<br>
<input type="radio" name="asset" value="design">Cool design<br>
<input type="radio" name="asset" value="service">Customer Service<br>
<b>Comments:</b><br>
<textarea name="comments" rows="3" cols="40"></textarea><br>
<input type="submit" value="Submit!">
</form>

```

The form as displayed in the browser is illustrated in Figure 10-7.

Please take a moment to tell us what you think about our site:

Name:

Email:

How frequently do you visit our site?:
 Site frequency:

I frequently purchase the following products from our site:

☐ Software
☐ Cookware
☐ Chef's Hats

Our site's greatest asset is:

☐ Product selection
☐ Cool design
☐ Customer Service

Comments:

Figure 10-7. A sample user input form

Chapter 10

Pretty straightforward, right? But the question now arises of how you take the user input and do something useful with it. That is the subject of the next section, “Forms and PHP.”

Keep in mind that this introduction to forms should be considered just that; an introduction. It is by no means a comprehensive summary of all options offered to the various form components. Check out any of the many forms-related tutorials on the Web and recently released HTML books for further information.

Having completed this introduction to HTML forms, I now proceed to the really interesting part of this chapter; that is, how PHP can be used to process and interact with user information input via these forms.

Forms and PHP

How PHP handles form information is really not all that different from how PHP handles variable data passed along with the URL, a subject I discussed in detail in the previous chapter.

Introductory Examples

To facilitate rapid learning of the various ways you can use PHP to manipulate form information, I present a series of scenarios. Each scenario illustrates a different way you can take advantage of this technology to add interactivity to your site.

Scenario 1: Passing Form Information from One Script to Another

This is perhaps the most basic of examples, in which user input is simply gathered on one page and displayed on another. Listing 10-2 contains the form that will prompt a user for a name and email address. When the user clicks the submit button, entitled “go!” the form will request listing10-3.php. Listing 10-3 will in turn display the \$name and \$email variables that were passed along with the page request.

Listing 10-2: A simple form

```
<html>
<head>
<title>Listing 10-2</title>
</head>
```

```

<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">

<form action="listing10-3.php" method="post">
<b>Give us some information!</b><br>
Your Name:<br>
<input type="text" name="name" size="20" maxlength="20" value=""><br>
Your Email:<br>
<input type="text" name="email" size="20" maxlength="40" value=""><br>
<input type="submit" value="go!">
</form>

</body>
</html>

```

Listing 10-3: Displaying the data collected in Listing 10-1

```

<html>
<head>
<title>Listing 10-3</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?
// output the user's name and email address.
print "Hi, $name!. Your email address is $email";
?>
</body>
</html>

```

In summary, the user fills out the form fields, clicks the submit button, and is directed to Listing 10-3, which then formats and displays the data. Simple as that.

An alternative method to carry out form processing uses just one script. The disadvantage of this method is that the script becomes longer and therefore relatively more difficult to view and understand; the advantage is that you minimize the total number of files to handle. Furthermore, you eliminate code redundancy when error checking, a subject I discuss later in this chapter. Of course, there will be times when one script is not advantageous, but it's always nice to keep in mind that it's easily done. In Scenario 2, I reconsider Scenario 1, this time using just one script.

Scenario 2: Alternative (One-Script) Form Processing

Processing forms using one script is fairly simple, making use of an *if* conditional statement to tell us whether or not the form variables have been set. If they have been set, then they will be processed (in this example, displayed). Otherwise, the form will be displayed. Verifying whether or not the variables have been set is accomplished with the `strcmp()` function, described in Chapter 8, “Strings and Regular Expressions.” Listing 10-4 provides an example of one-script processing. Notice how the form action calls the page in which it resides. A conditional *if* statement is used to check for the transmission of a hidden variable named *seenform*. If *seenform* doesn't exist, the form will be displayed. If *seenform* does exist, this implies that the form has been filled out by the user, and the information is processed, in this case output to the browser.

Listing 10-4: One-script form processing

```
<html>
<head>
<title>Listing 10-4</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?
// all double quotations in $form must be escaped,
// otherwise a parse error will occur
$form = "
<form action=\"listing10-4.php\" method=\"post\">
<input type=\"hidden\" name=\"seenform\" value=\"y\">
<b>Give us some information!</b><br>
Your Name:<br>
<input type=\"text\" name=\"name\" size=\"20\" maxlength=\"20\" value=\"\"><br>
Your Email:<br>
<input type=\"text\" name=\"email\" size=\"20\" maxlength=\"40\" value=\"\"><br>
<input type=\"submit\" value=\"subscribe!\">
</form>;
// If we haven't already seen the form ($seenform passed by hidden
// form value), show the form.
if ($seenform != "y"):
    print "$form";
else :
    print "Hi, $name!. Your email address is $email";
endif;
?>
</body>
</html>
```

Keep in mind that this is not the most user-friendly format, as it does not specifically inform readers that they have not correctly filled in the form on subsequent reloads of the page. The issue of error verification is discussed later in this chapter. For the moment, it's just important to realize how one script can be used to perform these operations.

Now that you're getting an idea just how easy it is to process form information, I'll proceed with an interesting example that will automatically mail the user information to an address that you specify. This is illustrated in Scenario 3.

Scenario 3: Sending the Information to an Email Address

While the idea of simply displaying the entered user information to the browser is appealing, it doesn't do too much for us in the sense of actually processing the user input in a meaningful way. One way to process this information could be to send it to a particular email address, for example, the site administrator's. While the `mailto:` hyperlink can spurn an email message from the browser, keep in mind that not all computers are configured with an external email application. Therefore, using a Web-based form to send email is one foolproof way to make sure the user's message gets to you.

In the next section, "mail()," I create a short form that prompts the user to enter some information and comments about the site. This data is then formatted accordingly and fed to PHP's predefined `mail()` function. Before delving into this sample form, take a moment to brush up on the `mail()` syntax.

mail()

The `mail()` function, as you may have surmised, is used to mail information to a given recipient. Its syntax is:

```
boolean mail(string recipient, string subject, string message [, string  
addl_headers])
```

The *subject* is, of course, the subject of the email. The *message* is the textual body of the email, and the optional input parameter *addl_headers* is used to supply any additional header information (such as HTML formatting) that is sent along with the email.

NOTE *The function `mail()` uses `sendmail` on UNIX-based machines. For Windows, this function will not work unless you have a mail server installed or you point `mail()` to a working SMTP server. This is accomplished by modifying the SMTP variable, in the `php.ini` file.*

Assuming you have heeded the preceding note, and your `mail()` function is operating properly, go ahead and execute the following code (of course first changing `youraddress@yourserver.com` to your valid email address):

```
$email = "youraddress@yourserver.com";
$subject = "This is the subject";
$message = "This is the message";
$headers = "From: somebody@somesite.com ";

mail($email, $subject, $message, $headers);
```

Although for large volumes of email you are definitely better off going with a robust mailer application such as `majordomo` (<http://www.greatcircle.com/majordomo/>), PHP's `mail()` function works great when the need arises.

Now that you've been brought up to speed regarding the `mail()` function, you are ready to implement it. Check out Listing 10-5, which gathers user information and sends it to an address specified by the script administrator.

Listing 10-5: Using `mail()` to redirect user information

```
<html>
<head>
<title>Listing 10-5</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">

<?
$form = "
<form action=\"listing10-5.php\" method=\"post\">
<input type=\"hidden\" name=\"seenform\" value=\"y\">
<b>Send us your comments!</b><br>
Your Name:<br>
<input type=\"text\" name=\"name\" size=\"20\" maxlength=\"20\" value=\"\"><br>
```

```

Your Email:<br>
<input type="text" name="email" size="20" maxlength="40" value=""><br>
Your Comments:<br>
<textarea name="comments" rows="3" cols="30"></textarea><br>
<input type="submit" value="submit!">
</form>
";

// If we haven't already seen the form ($seenform passed by hidden
// form value), show the form.
if ($seenform != "y") :
    print "$form";
else :
    // change $recipient to be the recipient of the form information
    $recipient = "yourname@youreemail.com";
    // email subject
    $subject = "User Comments ($name)";
    // extra email headers
    $headers = "From: $email";
    // send the email or produce an error
    mail($recipient, $subject, $comments, $headers) or die("Could not send
    email!");
    // send the user an appropriate message
    print "Thank you $name for taking a moment to send us your comments!";

endif;
?>
</body>
</html>

```

Pretty slick, isn't it? Listing 10-5 works like Listing 10-4, first checking whether or not the form has already been viewed by the user. Assuming that it has, the `mail()` function is invoked, and the user information is sent to the recipient address as designated by `$recipient`. The user is then greeted with a thank you message, displayed to the browser.

An easy add-on to this example is a second `mail()` call that formats a short thank you that is subsequently sent to the user. The next example builds on this premise, providing the user with a choice of newsletters. The user will be sent the corresponding newsletters based on the selections made.

Chapter 10

Scenario 4: Sending the User Information via Email

In this scenario, I use a series of checkboxes, each specifying a different informational brochure. Users can click one, two, or three of these checkboxes, enter their email address, and they will be sent the selected brochures. Note the usage of an array in the checkboxes. This will facilitate the verification of which checkboxes have been selected, in addition to improving the code organization.

Each informational email should be kept in a separate file. For purposes of this scenario, there are three text files:

- Site architecture: site.txt is the information about the site.
- Development team: team.txt is the information about our talented team.
- Upcoming events: events.txt invites you to join a wild event.

Now check out Listing 10-6.

Listing 10-6: Sending user-requested information

```
<html>
<head>
<title>Listing 10-6</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?

$form = "
<form action=\"listing10-6.php\" method=\"post\">
<input type=\"hidden\" name=\"seenform\" value=\"y\">
<b>Receive information about our site!</b><br>
Your Email:<br>
<input type=\"text\" name=\"email\" size=\"20\" maxlength=\"40\" value=\"\"><br>
<input type=\"checkbox\" name=\"information[site]\" value=\"y\">Site
Architecture<br>
<input type=\"checkbox\" name=\"information[team]\" value=\"y\">Development
Team<br>
<input type=\"checkbox\" name=\"information[events]\" value=\"y\">Upcoming
Events<br>
<input type=\"submit\" value=\"send it to me!\">
</form>;
```

```

if ($seenform != "y") :
    print "$form";
else :
    $headers = "From: devteam@yoursite.com";
    // cycle through each array key/value element
    while ( list($key, $val) = each ($information) ) :
        // verify if the current value is "y"
        if ($val == "y") :

            // create filename that corresponds with current key
            $filename = "$key.txt";
            $subject = "Requested $key information";

            // open the filename
            $fd = fopen ($filename, "r");
            // read the entire file into a variable
            $contents = fread ($fd, filesize ($filename));

            // send email.
            mail($email, $subject, $contents, $headers) or die("Can't send
            // email!");
            fclose($fd);
        endif;
    endwhile;
    // Notify user of success
    print sizeof($information)." informational newsletters have been sent to
$email!";
endif;
?>
</body>
</html>

```

Listing 10-6 uses a convenient while loop to cycle through each array key/value pair, sending out only those informational newsletters that correspond to those key/value pairs in which the value has been set to y. It is important to remember that the text files *must* be named in accordance with each key in the array (site.txt, team.txt, and events.txt). I dynamically create the filename using each key and then open that filename, in turn reading the file contents to a variable (*\$contents*). The *\$contents* variable is then used as the message parameter of the *mail()* function.

The next scenario deals with storing the user information in a more organized format, using a text file to do so.

Scenario 5: Adding the User Information to a Text File

This text file information can later be analyzed for statistical meaning, searched, or whatever your heart desires. Listing 10-7 is a one-script form processor like those in the previous scenarios. The user is prompted to complete four items: name, email address, preferred language, and occupation. This information is then appended to a text file aptly named `user_information.txt`. Information elements are separated by a vertical bar (|) to distinguish them from each other.

Listing 10-7: Storing user information in a text file

```
<html>
<head>
<title>Listing 10-7</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?
// create the form
$form = "
<form action=\"listing10-7.php\" method=\"post\">
<input type=\"hidden\" name=\"seenform\" value=\"y\">
<b>Give us your personal info!</b><br>
Your Name:<br>
<input type=\"text\" name=\"name\" size=\"20\" maxlength=\"20\" value=\"\"><br>
Your Email:<br>
<input type=\"text\" name=\"email\" size=\"20\" maxlength=\"20\" value=\"\"><br>
Your Preferred Language:<br>
<select name=\"language\">
<option value=\"\">Choose a language:
<option value=\"English\">English
<option value=\"Spanish\">Spanish
<option value=\"Italian\">Italian
<option value=\"French\">French
<option value=\"Japanese\">Japanese
<option value=\"newyork\">NewYork-ese
</select><br>
Your Occupation:<br>
<select name=\"job\">
<option value=\"\">What do you do?:
<option value=\"student\">Student
<option value=\"programmer\">Programmer
<option value=\"manager\">Project Manager
```

```

<option value=\"slacker\">Slacker
<option value=\"chef\">Gourmet Chef
</select><br>
<input type=\"submit\" value=\"submit!\">
</form>";

// has the form already been filled in?
if ($seenform != "y") :
    print "$form";
else :
    $fd = fopen("user_information.txt", "a");

    // make sure user has not entered a vertical bar in the input
    $name = str_replace("|", "", $name);
    $email = str_replace("|", "", $email);

    // assemble user information
    $user_row = $name."|".$email."|".$language."|".$job."\n";
    fwrite($fd, $user_row) or die("Could not write to file!");
    fclose($fd);
    print "Thank you for taking a moment to fill out our brief questionnaire!";

endif;
?>
</body>
</html>

```

One important item is the section of code that ensures that the user did not insert any vertical bars (|) in the name or email data. The function `str_replace()` strips those out, replacing them with nothing. If this were not done, any user-inserted vertical bars would act to corrupt the data file, making it difficult, if not impossible, to correctly parse that file.

Keep in mind that when dealing with relatively large amounts of information, a text file works just fine. However, when you are dealing with many users, or large amounts of information, you may wish to make use of a database to store and manipulate forms-entered data. This matter is covered in detail in Chapter 11.

Up to this point, the forms that I have covered have generally assumed that the user will enter correct and nonmalicious data. This is a very unwise assumption! In the next section I complement the scenarios discussed in this section, showing how you can verify the integrity of the forms information. In addition to weeding out any information that may be malicious or insufficient, error checking ultimately results in a more efficient and user-friendly interface.

Error Checking

Processing user information could be rather useless if it were constantly filled with misleading data. While there is no way to ensure that users are not lying, you can verify the information's integrity (for example, you can verify whether or not an email address is valid). While JavaScript is a popular technology for form verification, browser incompatibilities can render this useless. Since PHP executes on the server side, you can rest assured that the form data will always be correctly verified (provided that your code is correct).

When you find errors in user data, you need to inform users what errors were found and suggest how they could fix these errors. Some possible methods of informing users include simply displaying proper error messages and suggesting alternative variations of the entered information (as may be the case when the user enters a username that is already taken). In this section, I cover verification and message display, saving input information for a larger example in the project at the end of this chapter.

Scenario 6: Displaying Missing or Erred Form Fields

The last thing you want as a site developer is for users to become frustrated due to lack of feedback as to what they are doing wrong when filling out a form, particularly when requesting further product information or even making a purchase! One efficient way to ensure that users understand which form fields are missing or not being accepted is to display explicit error messages.

The idea behind this form of error checking is to check each field separately, ensuring that it is not empty and that illegal data has not been entered. If the field is OK, move on to the next field. Otherwise, display an appropriate error message, set a flag that will later be used as a means for triggering the form redisplay, and move on to the next field, repeating this process until you've checked the entire form. This method is illustrated in Listing 10-8.

Listing 10-8: Verifying form information and displaying appropriate messages

```
<html>
<head>
<title>Listing 10-8</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?
// create the form
$form = "
```

```

<form action=\"listing10-8.php\" method=\"post\">
<input type=\"hidden\" name=\"seenform\" value=\"y\">
<b>Give us some information!</b><br>
Your Name:<br>
<input type=\"text\" name=\"name\" size=\"20\" maxlength=\"20\"
value=\"$name\"><br>
Your Email:<br>
<input type=\"text\" name=\"email\" size=\"20\" maxlength=\"40\"
value=\"$email\"><br>
<input type=\"submit\" value=\"subscribe!\">
</form>;

// has the form already been filled in?
if ($seenform != "y"):
    print "$form";

// The user has filled out the form. Now verify the information
else :
    $error_flag = "n";
    // ensure that the name variable is not empty
    if ($name == "") :
        print "<font color=\"red\">* You forgot to enter your name!</font>
        // <br>";
        $error_flag = "y";
    endif;
    // ensure that the email variable is not empty
    if ($email == "") :

        print "<font color=\"red\">* You forgot to enter your email!</font>
        <br>";
        $error_flag = "y";

    else :
        // convert all email alphabetical characters to lowercase
        $email = strtolower(trim($email));
        // ensure the email address is of valid syntax
        if (! @ereg('^[0-9a-z]+'
            '@'
            '([0-9a-z-]+\.)+'
            '([0-9a-z]){2,4}$', $email)) :

```

Chapter 10

```

        print "<font color=\"red\">* You entered an invalid email
        address!</font> <br>";
        $error_flag = "y";
    endif;
endif;

// If the $error_flag has been set, redisplay the form
if ($error_flag == "y") :
    print "$form";
else :
    // do something with that user information
    print "You entered valid form information!";
endif;
endif;
?>
</body>
</html>

```

Listing 10-8 ensures that the form fields, in this case name and email address, is not empty and verifies that the supplied email address is valid. If any of the data checks out as invalid, the appropriate error messages are displayed, and the form is again displayed, with all of the previously entered data *already* filled in. The fact that the information is already filled in will make users more willing to attempt to correct the incorrect information. An empty form invites users to go elsewhere for the services they need.

Dynamic Forms Construction

To this point, I have been coding each form manually, toiling away as I bask in the irradiated glow of my computer screen. In the programming world, we all know that doing anything manually is bad, as it introduces the chance for errors, not to mention that it cuts into our game-playing time.

Therefore, in this section, I provide a scenario in which you use array data to dynamically build a pull-down menu. It's a rather simple process, but can cut quite a bit of time from the initial coding and later code maintenance.

Scenario 7: Generating a Pull-Down Menu

Assume that you have a list of your favorite sites that you would like to provide to your site readers as examples of cool design. Rather than hard coding each, you could build an array and then use that array to populate your form.

Listing 10-9 makes use of the one-script processor, first checking whether or not the `$site` variable has been assigned a value. If it has, the `header()` function is called, and the `$site` variable is appended to the string "Location: http://". This command tells the `header()` function to redirect the browser to whatever URL is specified in that command. If `$site` has not been assigned a value, then the form is displayed. The pull-down menu is created by looping x times based on the size of the `$favsites` array. This array contains five of my favorite sites. Of course, you can add as many sites as you wish.

CAUTION *It is extremely important that you understand that the `header()` function must be called before any output is displayed to the browser. You cannot just call `header()` whenever you wish in a PHP script. Calling `header()` at the wrong time has caused such grief among novice PHP developers that I suggest you repeat this rule five times.*

Listing 10-9: Generating a dynamic pull-down menu

```
<?
if ($site != "") :
    header("Location: http://$site");
    exit;
else :
?>

<html>
<head>
<title>Listing 10-9</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#cbda74" vlink="#808040"
alink="#808040">
<?
$favsites = array ("www.k10k.com",
                    "www.yahoo.com",
                    "www.drudgereport.com",
                    "www.phprecipes.com",
                    "www.frogdesign.com");

// now build the form
?>
<form action = "listing10-9.php" method="post">
<select name="site">
<option value = "">Choose a site:
<?

```


Chapter 10

```

$x = 0;

while ( $x < sizeof ($favsites) ) :
    print "<option value='$favsites[$x]'$>$favsites[$x]";
    $x++;
endwhile;
?>
</select>
<input type="submit" value="go!">
</form>
<?
endif;
?>

```

Generating dynamic forms is particularly useful when you are dealing with a large amount of data that could change at a given time, thus causing any hard-coded form information to become outdated. However, I do recommend hard coding any information that you would deem to be static (for example, a listing of the U.S. states), as this could result in a slight gain in performance.

Project: Create a Guestbook

Ever since the dawn of the World Wide Web, site builders have been interested in providing ways for visitors to post comments and thoughts onsite. Typically, this feature is known as a *guestbook*. Using HTML forms, PHP's form-processing capabilities, and a text file, I'll show you just how easily you can create a customizable guestbook.

First, you want to create an initialization file that holds certain global variables and application functions. This file, entitled "init.inc", is shown in Listing 10-10.

Listing 10-10: The init.inc file used to create your guestbook

```

<?
// file: init.inc
// purpose: provides global variables and functions for use in guestbook project

// Default page title
$title = "My Guestbook";

// Background color
$bg_color = "white";

```

```

// Font face
$font_face = "Arial, Verdana, Times New Roman";

// Font color
$font_color = "black";

// Posting date
$post_date = date("M d y");

// Guestbook data
$guest_file = "comments.txt";

// This function retrieves the guestbook information
// and displays it to the browser.
function view_guest($guest_file) {

    GLOBAL $font_face, $font_color;

    print "Return to <a href=\"index.php\">index</a>.<br><br>";

    // If there is data in the guestbook data file...

    if (filesize($guest_file) > 0) :

        // open the guestbook data file
        $fh = fopen($guest_file, "r") or die("Couldn't open $guest_file");

        print "<table border=1 cellpadding=2 cellspacing=0 width=\"600\">";

        // while not the end of the file
        while (! feof($fh)) :

            // get the next line
            $line = fgets($fh, 4096);

            // split the line apart, and assign each component to a variable
            list($date, $name, $email, $comments) = explode("|", $line);

            // If there is a name, print it
            if ($name != "") :

                print "<tr>";
                print "<td><font color=\"$font_color\"";
                print "face=\"$font_face\">Date:</font></td>";
            }
        }
    }
}

```

Chapter 10

```

        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">\$date</font></td>";
        print "</tr>";

        print "<tr>";
        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">Name:</font></td>";
        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">\$name</font></td>";
        print "</tr>";

        print "<tr>";
        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">Email:</font></td>";
        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">\$email</font></td>";
        print "</tr>";

        print "<tr>";
        print "<td valign=\"top\"><font color=\"\$font_color\"
        // face=\"\$font_face\">Message:</font></td>";
        print "<td><font color=\"\$font_color\"
        // face=\"\$font_face\">\$comments</font></td>";
        print "</tr>";

        print "<tr><td colspan=\"2\">&nbsp;</td></tr>";
    endif;

endwhile;

print "</table>";

// close the file
fclose(\$fh);

else :

    print "Currently there are no entries in the guestbook!";

endif;

} // view_guest

```

```
// This function adds new information to the datafile.
function add_guest($name, $email, $comments) {

    GLOBAL $post_date, $guest_file;

    // format the data for file input
    $contents = "$post_date|$name|$email|$comments\n";

    // open the data file
    $fh = fopen($guest_file, "a") or die("Could not open $guest_file!");

    // write the data contents to the file
    $wr = fwrite($fh, $contents) or die("Could not write to $guest_file!");

    // close the file
    fclose($fh);

} // add_guest
?>
```

Next, you want to create three files: an index file containing the “add” and “view” links for the guestbook, a file that will display the guestbook information, and a file that will prompt users for new guestbook information. The index file in Listing 10-11 simply displays two links to the functions of the guestbook: adding and viewing data. You could easily incorporate these links into a more comprehensive site.

Listing 10-11: An index file containing add and view links for your guestbook

```
<html>
<?
INCLUDE("init.inc");
?>
<head>
<title><?=$page_title;?></title>
</head>
<body bgcolor="<?=$bg_color;?>" text="#000000" link="#808040" vlink="#808040"
alink="#808040">
<a href="view_guest.php">View the guestbook!</a><br>
<a href="add_guest.php">Sign the guestbook!</a><br>
</body>
</html>
```

Chapter 10

The `view_guest.php` file in Listing 10-12 displays all guestbook information in the data file.

Listing 10-12: The `view_guest.php` file

```
<html>
<?
INCLUDE("init.inc");
?>
<head>
<title><?=$page_title;?></title>
</head>
<body bgcolor="<?=$bg_color;?>" text="#000000" link="#808040" vlink="#808040"
alink="#808040">
<?
view_guest($guest_file);
?>
```

The `add_guest.php` file in Listing 10-13 prompts users for new guestbook data. Subsequently, it is used to add the information to the data file.

Listing 10-13: The `add_guest.php` file

```
<html>
<?
INCLUDE("init.inc");
?>
<head>
<title><?=$page_title;?></title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#808040" vlink="#808040"
alink="#808040">

<?
// If the form data has not yet been passed, display the form.
if (! $seenform) :
?>

<form action="add_guest.php" method="post">
<input type="hidden" name="seenform" value="y">

Name:<br>
<input type="text" name="name" size="15" maxlength="30" value=""><br>
```

```

Email:<br>
<input type="text" name="email" size="15" maxlength="35" value=""><br>

Comment:<br>
<textarea name="comment" rows="3" cols="40"></textarea><br>
<input type="submit" value="submit">
</form>
<?
// The form data has been passed. Therefore, add it to the text file
else :

    add_guest($name, $email, $comment);

    print "Your comments have been added to the guestbook. <a
    href=\"index.php\">Click here</a> to return to the index.";

endif;

?>

```

One of the greatest advantages of developing an application using well-defined functions is ease of portability. For example, you may be interested in storing guestbook data in a database instead of a text file. No problem: just change the contents of the `add_guest()` and `view_guest()` files for database operation, and you've successfully ported the guestbook application for database use.

Figure 10-8 illustrates how the guestbook would be viewed after a few entries have been inserted:

Return to [index](#).

Date:	Oct 29 00
Name:	Michele
Email:	michelle@latorre.com
Message:	I love cheese!
Date:	Oct 29 00
Name:	Nino
Email:	nino@latorre.com
Message:	Great site!

Figure 10-8. *view_guest.php*

Chapter 10

The datafile would store the same information shown in Figure 10-8 as follows:

```
Oct 29 00|Michele|michelle@latorre.com|I love cheese!  
Oct 29 00|Nino|nino@latorre.com|Great site!
```

What's Next?

Processing form information is arguably one of PHP's greatest features, as it introduces ease and reliability to an already extremely valuable site feature, that is, user interactivity. In this chapter we covered quite a bit of material regarding forms and PHP's role in forms processing, namely:

- An introduction to form syntax
- Passing form info from one PHP script to another
- One-script forms processing
- The `mail()` function
- Sending forms information to an email address
- Automating user information requests via email
- Adding user information to a text file
- Error checking
- Dynamic forms construction

Incorporating a database into your site infrastructure is one of the first steps you should consider taking if you intend to provide any substantial amount of information over the Web. This is the subject of the next chapter.