

3

Chapter 14 Head
First HTML & CSS
Using Forms

HTML Forms and PHP

The previous chapter provided a brief introduction to the topic of variables. Although you'll often create your own variables, you'll also commonly use variables in conjunction with HTML forms. Forms are a fundamental unit of today's Web sites, enabling such features as registration and login systems, search capability, and online shopping. Even the simplest site will find logical reasons to incorporate HTML forms. And with PHP, it's stunningly simple to receive and handle data generated by them.

With that in mind, this chapter will cover the basics of creating HTML forms and explain how the submitted form data is available to a PHP script. This chapter will also introduce several key concepts of real PHP programming, including how to manage errors in your scripts.

In This Chapter

Creating a Simple Form	50
Choosing a Form Method	54
Receiving Form Data in PHP	57
Displaying Errors	61
Error Reporting	64
Manually Sending Data to a Page	67
Review and Pursue	72

Array Keys are taken from the `name=""` attribute of each form element.

Creating a Simple Form

For the HTML form example in this chapter, you'll create a feedback page that takes the user's salutation, name, email address, response, and comments **A**. The code that generates a form goes between opening and closing **form** tags:

```
<form>
```

form elements

This is HTML

```
</form>
```

The **form** tags dictate where a form begins and ends. Every element of the form must be entered between these two tags. The opening **form** tag also contains an **action** attribute. It indicates the page to which the form data should be submitted. This value is one of the most important considerations when you're creating a form. In this book, the **action** attributes will always point to PHP scripts:

```
<form action="somepage.php"> using a script that checks the data then handles it
```

~~Before creating this next form, let's briefly revisit the topic of XHTML. As stated in the first chapter, XHTML has some rules that result in a significantly different syntax than HTML. For starters, the code needs to be in all lowercase letters, and every tag attribute must be enclosed in quotes. Further, every tag must be closed; those that don't have formal closing tags, like **input**, are closed by adding a blank space and a slash at the end. Thus, in HTML you write~~

```
<input type="text"
name="address" size="40">
```

The text `<input>` element is for entering one line of text. The `<input>` element is a void element, so there's no content after it. Use the type attribute to indicate you want "text" input.

Please complete this form to submit your feedback:

Name:

Email Address:

Response: This is... ☐ excellent ☐ okay ☐ boring

Comments:

A The HTML form that will be used in this chapter's examples.

action attribute - where to send the data for processing

Script 3.1 This HTML page has a form with several different input types.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
6     content="text/html; charset=utf-8"/>
7   <title>Feedback Form</title>
8 </head>
9 <body>
10 <!-- Script 3.1 - feedback.php -->
11 <div><p>Please complete this form to
  submit your feedback:</p>
12 <form action="handle_form.php">
13
14   <p>Name: <select name="title">
15     <option value="Mr.">Mr.</option>
16     <option value="Mrs.">Mrs.</option>
17     <option value="Ms.">Ms.</option>
18 </select> <input type="text"
  name="name" size="20"></p>
19
20 <p>Email Address: <input
  type="text" name="email"
  size="20" /></p>
21
22 <p>Response: This is...
23 <input type="radio"
  name="response" value="excellent">
  excellent
24 <input type="radio"
  name="response" value="okay">
  okay
25 <input type="radio"
  name="response" value="boring">
  boring</p>
26
27 <p>Comments: <textarea
  name="comments" rows="3"
  cols="30"></textarea></p>
28
29 <input type="submit"
  name="submit" value="Send My
  Feedback">
30 </form>
31 </div>
32 </body>
33 </html>
```

Hopefully this quick explanation will help you understand the XHTML in the following script.

Finally, in both HTML and XHTML, each form element needs to have its own unique name. Stick to a consistent naming convention when naming elements, using only letters, numbers, and the underscore (_). The result should be names that are also logical and descriptive.

To create a basic HTML form:

1. Begin a new document in your text editor or IDE, to be named **feedback.php** (Script 3.1): **See Tip #1 p. 53**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/
  → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
  <title>Feedback Form</title>
</head>
<body>
<!-- Script 3.1 - feedback.php -->
<div><p>Please complete this form
→ to submit your feedback:</p>
```

2. Add the opening **form** tag:

```
<form action="handle_form.php">
```

The **form** tag indicates that this form will be submitted to the page **handle_form.php**, found within the same directory as this HTML page. ~~You can use a full URL to the PHP script, if you'd prefer to be explicit (e.g., http://www.example.com/handle_form.php).~~ **Not recommended**

continues on next page

3. Add a select menu plus a text input for the person's name:

```
<p>Name: <select name="title">
<option value="Mr.">Mr.</option>
<option value="Mrs.">Mrs.</option>
<option value="Ms.">Ms.</option>
</select> <input type="text"
name="name" size="20"></p>
```

The inputs for the person's name will consist of two elements **A**. The first is a drop-down menu of common titles: *Mr.*, *Mrs.*, and *Ms.* Each option listed between the **select** tags is an answer the user can choose **B**. The second element is a basic text box for the person's full name.

4. Add a text input for the user's email address:

```
<p>Email Address: <input
type="text" name="email"
size="20"></p>
```

5. Add radio buttons for a response:

```
<p>Response: This is...
<input type="radio" name="response" →
value="excellent"> excellent
<input type="radio" name="response" →
value="okay"> okay
<input type="radio" name="response" →
value="boring"> boring</p>
```

This HTML code creates three radio buttons (clickable circles, **A**). Because they all have the same **name** value, only one of the three can be selected at a time. Per XHTML rules, the code is in lowercase except for the values, ~~and an extra space and slash are added to the end of each input to close the tag.~~

B The **select** element creates a drop-down menu of options.

The **<select>** element creates a menu control in the web page. The menu provides a way to choose between a set of choices. The **<select>** element works in combination with the **<option>** element below to create a menu. The **<select>** element goes around all the menu option to group them into one menu. Give the **<select>** element a unique name using the name attribute.

See Tip #4 on pg. 53 type="email"

http://www.w3schools.com/html/html5_form_attributes.asp

All radio buttons associated with a given set of choices must have the same name value. But each choice has a different value (i.e. boring)

6. Add a **textarea** to record the comments:


```
<p>Comments: <textarea  
→ name="comments" rows="3"  
→ cols="30"></textarea></p>
```

A **textarea** gives users more space to enter their comments than a text input would. However, the text input lets you limit how much information users can enter, which you can't do with the **textarea** (not without using JavaScript, that is). When you're creating a form, choose input types appropriate to the information you wish to retrieve from the user.

Note that a **textarea** *does* have a closing tag.

7. Add the submit button:

```
<input type="submit" name="submit"  
→ value="Send My Feedback" />
```

The **value** attribute of a submit element is what appears on the button in the Web browser . You could also use *Go!* or *Submit*, for example.

8. Close the form:

```
</form>
```

9. Complete the page:

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

10. Save the page as **feedback.html** and view it in your browser.

Because this is an HTML page, not a PHP script, you could view it in your Web browser directly from your computer.

TIP Note that **feedback.html** uses the HTML extension because it's a standard HTML page (not a PHP script). You could use the **.php** extension without a problem, even though there's no actual PHP code. (Remember that in a PHP page, anything not within the PHP tags—**<?php** and **?>**—is assumed to be HTML.)

TIP Be certain that your **action** attribute correctly points to an existing file on the server, or your form won't be processed properly. In this case, the form will be submitted to **handle_form.php**, to be located in the same directory as the **feedback.html** page.

TIP In this example, an HTML form is created by hand-coding the HTML, but you can do this in a Web page application (such as Adobe Dreamweaver) if you're more comfortable with that approach.

TIP One welcome addition in the forthcoming HTML 5 specification are new form elements, such as **email**, **url**, and **number**.

Form processing scripts rely on the "name" attribute. The "name" attribute must be used for each element that you want processed. The array keys are taken from the **name=""** attribute for each form element.

Choosing a Form Method

The experienced HTML developer will notice that the feedback form just created is missing one thing: The initial **form** tag has no **method** attribute. The **method** attribute tells the server how to transmit the data from the form to the handling script.

You have two choices with **method**: GET and POST. With respect to forms, the difference between using GET and POST is squarely in how the information is passed from the form to the processing script. The GET method sends all the gathered information along as part of the URL. The POST method transmits the information invisibly to the user. For example, upon submitting a form, if you use the GET method, the resulting URL will be something like

http://www.example.com/page.php?
⇒ **some_var=some_value&age=20&...** ←

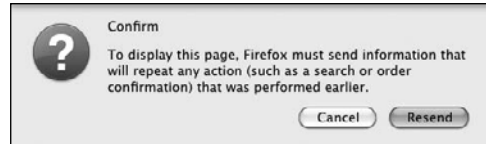
Following the name of the script, **page.php**, is a question mark, followed by one *name=value* pair for each piece of data submitted.

When using the POST method, the end user will only see

http://www.example.com/page.php

When deciding which method to use, keep in mind these four factors:

- With the GET method, a limited amount of information can be passed.
- The GET method sends the data to the handling script publicly (which means, for example, that a password entered in a form would be viewable by anyone within eyesight of the Web browser, creating a larger security risk).



A If users refresh a PHP script that data has been sent to via the POST method, they will be asked to confirm the action (the specific message will differ using other browsers).

You see this type of URL when using a search engine. Everything after the ? is a query string.

Script 3.2 The **method** attribute with a value of **post** is added to complete the form.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="Content-Type"
6     content="text/html; charset=utf-8"/>
7   <title>Feedback Form</title>
8 </head>
9 <body>
10 <!-- Script 3.2 - feedback.php -->
11 <div><p>Please complete this form to
  submit your feedback:</p>
12 <form action="handle_form.php"
  method="post">
13
14   <p>Name: <select name="title">
15     <option value="Mr.">Mr.</option>
16     <option value="Mrs.">Mrs.</option>
17     <option value="Ms.">Ms.</option>
18 </select> <input type="text"
  name="name" size="20"></p>
19
20   <p>Email Address: <input
  type="text" name="email"
21   size="20"></p>
22   <p>Response: This is...
23   <input type="radio" name="response"
  value="excellent"> excellent
24   <input type="radio" name="response"
  value="okay"> okay
25   <input type="radio"
  name="response" value="boring">
  boring</p>
26   <p>Comments: <textarea name="comments"
  rows="3" cols="30"></textarea></p>
27
28   <input type="submit"
29   name="submit" value="Send My
  Feedback">
30 </form>
31 </div>
32 </body>
33 </html>
```

- A page generated by a form that used the GET method can be bookmarked, but one based on POST can't be.
- Users will be prompted if they attempt to reload a page accessed via POST **A**, but will not be prompted for pages accessed via GET.

Generally speaking, GET requests are used when asking for information from the server. Search pages almost always use GET (check out the URLs the next time you use a search engine), as do sites that paginate results (like the ability to browse categories of products). Conversely, POST is normally used to trigger a server-based action. This might be the submission of a contact form (result: an email gets sent) or the submission of a blog's comment form (result: a comment is added to the database and therefore the page).

This book uses POST almost exclusively for handling forms, although you'll also see a useful technique involving the GET method (see "Manually Sending Data to a Page" at the end of this chapter).

To add a method to a form:

1. Open **feedback.php** (Script 3.1) in your text editor or IDE, if it is not already open.
2. Within the initial **form** tag, add **method="post"** (Script 3.2, line 12).

The form's **method** attribute tells the browser how to send the form data to the receiving script. Because there may be a lot of data in the form's submission (including the comments), and because it wouldn't make sense for the user to bookmark the resulting page, POST is the logical method to use.

continues on next page

3. Save the script and reload it in your Web browser.

It's important that you get in the habit of reloading pages in the Web browser after you make changes. It's quite easy to forget the reloading step and find yourself flummoxed when your changes are not being reflected.

4. View the source of the page to make sure all the required elements are present and have the correct attributes **B**.

TIP In the discussion of the methods, **GET** and **POST** are written in capital letters to make them stand out. However, the form in the script uses *post* for XHTML compliance. Don't worry about this inconsistency (if you caught it at all)—the method will work regardless of case.

The `print_r()` function allows you to inspect the contents of arrays. `$_POST` is a superglobal, which is an associative array

The `<pre>` tags simply makes the output easier to read.

```
<pre>
<?php if ($_GET) {
    echo 'Contents of the $_GET
array: <br>'; //single quotes added
for literal interpretation of variable
    print_r($_GET);
} elseif ($_POST) {
    echo 'Contents of the $_POST
array: <br>';
    print_r($_POST); }
?>
</pre>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
"http://www.w3.org/TR/xhtml1/DTD/
<html xmlns="http://www.w3.org/1999/xhtml
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Feedback Form</title>
</head>
<body>
<!-- Script 3.2 - feedback.html -->
<div><p>Please complete this form to submit your feedback:</p>

<form action="handle_form.php" method="post">

    <p>Name: <select name="title">
    <option value="Mr.">Mr.</option>
    <option value="Mrs.">Mrs.</option>
    <option value="Ms.">Ms.</option>
    </select> <input type="text" name="name" size="20" /></p>

    <p>Email Address: <input type="text" name="email" size="20" /></p>

    <p>Response: This is...
    <input type="radio" name="response" value="excellent" /> excellent
    <input type="radio" name="response" value="okay" /> okay
    <input type="radio" name="response" value="boring" /> boring</p>

    <p>Comments: <textarea name="comments" rows="3" cols="30"></textarea></p>

    <input type="submit" name="submit" value="Send My Feedback" />

</form>
</div>
</body>
</html>
```

B With forms, much of the important information, such as the **action** and **method** values or element names, can only be seen within the HTML source code.

Receiving Form Data in PHP

Now that you've created a basic HTML form capable of taking input from a user, you need to write the PHP script that will receive and process the submitted form data. For this example, the PHP script will simply repeat what the user entered into the form. In later chapters, you'll learn how to take this information and store it in a database, send it in an email, write it to a file, and so forth.

To access the submitted form data, you need to refer to a particular *predefined variable*. Chapter 2, "Variables," already introduced one predefined variable: `$_SERVER`. When it comes to handling form data, the specific variable the PHP script would refer to is either `$_GET` or `$_POST`. If an HTML form uses the GET method, the submitted form data will be found in `$_GET`. When an HTML form uses the POST method, the submitted form data will be found in `$_POST`.

`$_GET` and `$_POST`, besides being predefined variables (i.e., ones you don't need to create), are *arrays*, a special variable type (`$_SERVER` is also an array). This means that both `$_GET` and `$_POST` may contain numerous values, making the printing of those values more challenging. You cannot treat arrays like so (also see Figure B under "Variable Values" in Chapter 2):

```
print $_POST; // Will not work!
```

Instead, to access a specific value, you must refer to the array's *index* or *key*.

Chapter 7, "Using Arrays," goes into this subject in detail, but the premise is simple.

Start with a form element whose *name* attribute has a value of *something*:

```
<input type="text" name="something" />
```

Then, assuming that the form uses the POST method, the value entered into that form element would be available in `$_POST['something']`:

```
print $_POST['something'];
```

Unfortunately, there is one little hitch here: When used within double quotation marks, the single quotation marks around the key will cause parse errors **A**:

```
print "Thanks for saying:
→ $_POST['something']";
```

There are a couple of ways you can avoid this problem. This chapter will use the solution that's syntactically the simplest: just assign the particular `$_POST` element to another variable first: **AKA a shorthand variable.**

```
$something = $_POST['something'];
print "Thanks for saying: $something";
```

Two final notes before implementing this information in a new PHP script: First, as with all variables in PHP, `$_POST` is case-sensitive: it must be typed exactly as you see it here (a dollar sign, one underscore, then all capital letters). Second, the indexes in `$_POST`—*something* in the preceding example—must exactly match the *name* attributes values in the corresponding form element.

Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in /Users/larryullman/Sites/phpvqs4/handle_form.php on line 19

A This ugly parse error is created by attempting to use `$_POST['something']` within double quotation marks.

To handle an HTML form:

1. Begin a new document in your text editor or IDE, to be named **handle_form.php** (Script 3.3):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/
    → xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>Your Feedback</title>
</head>
<body>
```

2. Add the opening PHP tag and any comments:

```
<?php // Script 3.3 handle_form.php
// This page receives the data
→ from feedback.html.
// It will receive: title, name,
→ email, response, comments, and
→ submit in $_POST.
```

Comments are added to the script to make the script's purpose clear. Even though the **feedback.php** page indicates where the data is sent (via the **action** attribute), a comment here indicates the reverse (where this script is getting its data). It also helps to spell out the exact form element names, in a case-sensitive manner.

3. Assign the received data to new variables:

```
$title = $_POST['title'];
$name = $_POST['name'];
$response = $_POST['response'];
$comments = $_POST['comments'];
```

Script 3.3 This script displays the form data submitted to it by referencing the associated **\$_POST** variables.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
        xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
6      <title>Your Feedback</title>
7  </head>
8  <body>
9      <?php // Script 3.3 handle_form.php
10
11      // This page receives the data from
        feedback.html.
12      // It will receive: title, name, email,
        response, comments, and submit in $_POST.
13
14      // Create shorthand versions of the
        variables:
15      $title = $_POST['title'];
16      $name = $_POST['name'];
17      $response = $_POST['response'];
18      $comments = $_POST['comments'];
19
20      // Print the received data:
21      print "<p>Thank you, $title $name,
        for your comments.</p>";
22      <p>You stated that you found this
        example to be '$response' and
        added:<br />$comments</p>;
23
24  ?>
25  </body>
26  </html>
```

Magic Quotes

Earlier versions of PHP had a feature called *Magic Quotes*, which has since been **deprecated** (meaning you shouldn't use it and it will be removed from the language in time). Magic Quotes—when enabled—automatically escapes single and double quotation marks found in submitted form data. So the string *I'd like more information* would be turned into *I\'d like more information*.

The escaping of potentially problematic characters can be useful and even necessary in some situations. But if the Magic Quotes feature is enabled on your PHP installation, you'll see these backslashes when the PHP script prints out the form data. You can undo its effect using the **stripslashes()** function. To apply it to the **handle_form.php** script, you would do this, for example:

```
$comments = stripslashes  
→ ($_POST['comments']);
```

instead of just this:

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

That will have the effect of converting an escaped submitted string back to its original, non-escaped value.

If you're not seeing extraneous slashes added to submitted form data, you don't need to worry about Magic Quotes.

Again, since the form uses the POST method, the submitted data can be found in the **\$_POST** array. The individual values are accessed using the syntax **\$_POST['name_attribute_value']**. This works regardless of the form element's type (input, select, checkbox, etc.).

To make it easier to use these values in a **print** statement in Step 4, each value is assigned to a new variable in this step. Neither **\$_POST['email']** nor **\$_POST['submit']** is being addressed, but you can create variables for those values if you'd like.

4. Print out the user information:

```
print "<p>Thank you, $title $name,  
→ for your comments.</p>  
<p>You stated that you found this  
→ example to be '$response' and  
→ added:<br />$comments</p>;
```

This one **print** statement uses the four variables within a context to show the user what data the script received.

5. Close the PHP section and complete the HTML page:

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

6. Save the script as **handle_form.php**.

Note that the name of this file must exactly match the value of the **action** attribute in the form.

7. Upload the script to the server (or store it in the proper directory on your computer if you've installed PHP), making sure it's saved in the same directory as **feedback.php**.

continues on next page

8. Load **feedback.php** in your Web browser through a URL (*http://something*).

You must load the HTML form through a URL so that when it's submitted to the PHP script, that PHP script is also run through a URL. *PHP scripts must always be run through a URL!*

Failure to load a form through a URL is a common beginner's mistake.

9. Fill out **B**, and then submit the form **C**.

If you see a blank page, read the next section of the chapter for how to display the errors that presumably occurred.

If you see an error notice **D** or see that a variable does not have a value when printed, you likely misspelled either the form element's **name** value or the **\$_POST** array's index (or you filled out the form incompletely).

TIP If you want to pass a preset value along to a PHP script, use the *hidden* type of input within your HTML form. For example, the line

```
<input type="hidden" name="form_page" value="feedback.html">
```

inserted between the form tags will create a variable in the handling script called **\$_POST['form_page']** with the value **feedback.php**.

TIP Notice that the value of radio button and certain menu variables is based on the **value** attribute of the selected item (for example, *excellent* from the radio button). This is also true for checkboxes. For text boxes, the value of the variable is what the user typed.

TIP If the **handle_form.php** script displays extra slashes in submitted strings, see the "Magic Quotes" sidebar for an explanation and solution.

TIP You can also access form data, regardless of the form's method, in the **\$_REQUEST** predefined variable. **\$_GET** and **\$_POST** are more precise, however, and therefore preferable.

Please complete this form to submit your feedback:

Name:

Email Address:

Response: This is... ☒ excellent ☐ okay ☐ boring

Comments:

B Whatever the user enters into the HTML form should be printed out to the Web browser by the **handle_form.php** script **C**.

Thank you, Mr. Larry Ullman, for your comments.

You stated that you found this example to be 'excellent' and added:
No problems so far!

C This is another application of the **print** statement discussed in Chapter 1, but it constitutes your first dynamically generated Web page.

Notice: Undefined index: Name in /Users/larryullman/Sites/phpvqs4/handle_form.php on line 16

Thank you, Mr. , for your comments.

D Notices like these occur when a script refers to a variable that doesn't exist. In this particular case, the cause is erroneously referring to **\$_POST['Name']** when it should be **\$_POST['name']**.

Displaying Errors

One of the very first issues that arise when it comes to debugging PHP scripts is that you may or may not even see the errors that occur. After you install PHP on a Web server, it will run under a default configuration with respect to security, performance, how it handles data, and so forth. One of the default settings is to not display any errors. In other words, the **display_errors** setting will be off **A**. When that's the case, what you might see when a script has an error is a blank page. (This is the norm on fresh installations of PHP; most hosting companies will enable **display_errors**.)

The reason that errors should not be displayed on a live site is that it's a security risk. Simply put, PHP's errors often give away too much information for the public at large to see (not to mention showing PHP errors looks unprofessional). But you, the developer, *do* need to see these errors in order to fix them!

To have PHP display errors, you can do one of the following:

- Turn **display_errors** back on for PHP as a whole. (See the “Configuring PHP” section of Appendix A, “Installation and Configuration,” for more information.)
- Turn **display_errors** back on for an individual script.

continues on next page

display_errors	Off	Off
display_startup_errors	On	On
doc_root	<i>no value</i>	<i>no value</i>
docref_ext	<i>no value</i>	<i>no value</i>
docref_root	<i>no value</i>	<i>no value</i>
enable_dl	On	On
error_append_string	<i>no value</i>	<i>no value</i>
error_log	/Applications/MAMP/logs /php_error.log	/Applications/MAMP/logs /php_error.log
error_prepend_string	<i>no value</i>	<i>no value</i>
error_reporting	32767	32767

A Run a **phpinfo()** script (e.g., Script 1.2) to view your server's **display_errors** setting.

To find your php.ini file on a MAC:
Applications/
XAMPP/etc/php.ini
To find your php.ini file on a PC;
C:\XAMPP\php
\php.ini - page 423

While developing a site, the first option is by far preferred. However, it's only a possibility for those with administrative control over the server. Anyone can use the second option by including this line in a script:

```
ini_set ('display_errors', 1);
```

The `ini_set()` function allows a script to temporarily override a setting in PHP's configuration file (many, but not all, settings can be altered this way). The previous example changes the `display_errors` setting to *on*, which is represented by the number 1.

Although this second method can be implemented by anyone, the downside is that if your script contains certain kinds of errors (discussed next), the script cannot be executed. In that situation, this line of code won't be executed, and the particular error—or any that prevents a script from running at all—still results in a blank page.

To display errors in a script:

1. Open `handle_form.php` in your text editor or IDE, if it is not already open.
2. As the first line of PHP code, enter the following (Script 3.4):

```
ini_set ('display_errors', 1);
```

Again, this line tells PHP you'd like to see any errors that occur. You should call it first thing in your PHP section so the rest of the PHP code will abide by this new setting.

3. Save the file as `handle_form.php`.

Script 3.4 This addition to the PHP script turns on the `display_errors` directive so that errors will be shown.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
        xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
6      <title>Your Feedback</title>
7  </head>
8  <body>
9  <?php // Script 3.4 - handle_form.php #2
10
11  ini_set ('display_errors', 1);
    // Let me learn from my mistakes!
12
13  // This page receives the data from
        feedback.html.
14  // It will receive: title, name, email,
        response, comments, and submit in
        $_POST.
15
16  // Create shorthand versions of the
        variables:
17  $title = $_POST['title'];
18  $name = $_POST['name'];
19  $response = $_POST['response'];
20  $comments = $_POST['comments'];
21
22  // Print the received data:
23  print "<p>Thank you, $title $name, for
        your comments.</p>";
24  <p>You stated that you found this
        example to be '$response' and added:
        <br />$comments</p>";
25
26  ?>
27 </body>
28 </html>
```

Please complete this form to submit your feedback:

Name:

Email Address:

Response: This is... ☐ excellent ☐ okay ☐ boring

Comments:

B Incompletely filling out the form...

Notice: Undefined index: response in /Users/larryullman/Sites/phpvqs4/handle_form.php on line 19

Thank you, Mr. Larry Ullman, for your comments.

You stated that you found this example to be " and added:

C ...results in error messages. These notices are generated by references to form elements for which there are no values.

4. Upload the file to your Web server and test it in your Web browser (**B** and **C**).

If the resulting page has no errors in it, then the script will run as it did before. If you saw a blank page when you ran the form earlier, you should now see messages like those in **C**. Again, if you see such errors, you likely misspelled the name of a form element, misspelled the index in the `$_POST` array, or didn't fill out the form completely.

TIP Make sure `display_errors` is enabled any time you're having difficulties debugging a script. If you installed PHP on your computer, I *highly* recommend enabling it in your PHP configuration while you learn (again, see Appendix A).

TIP If you see a blank page when running a PHP script, also check the HTML source code for errors or other problems.

TIP Remember that the `display_errors` directive only controls whether error messages are sent to the Web browser. It doesn't create errors or prevent them from occurring in any way.

TIP Failure to use an equals sign after *name* in a form element will also cause problems:

```
<input name"something" />
```

Error Reporting

Another PHP configuration issue you should be aware of, along with **display_errors**, is *error reporting*. There are eleven different types of errors in PHP, plus four user-defined types (which aren't covered in this book). **Table 3.1** lists the four most important general error types, along with a description and example of each.

You can set what errors PHP reports on in two ways. First, you can adjust

the **error_reporting** level in PHP's configuration file (again, see Appendix A). If you are running your own PHP server, you'll probably want to adjust that global setting while developing your scripts.

The second option is to use the **error_reporting()** function in a script. The function takes either a number or one or more *constants* (nonquoted strings with predetermined meanings) to adjust the levels. The most important of these constants are listed in **Table 3.2**.

TABLE 3.1 PHP Error Types

Type	Description	Example
Notice	Nonfatal error that may or may not be indicative of a problem	Referring to a variable that has no value
Warning	Nonfatal error that is most likely problematic	Misusing a function
Parse error	Fatal error caused by a syntactical mistake	Omission of a semicolon or an imbalance of quotation marks, braces, or parentheses
Error	A general fatal error	Memory allocation problem

TABLE 3.2 Error Reporting Constants

Name
E_NOTICE
E_WARNING
E_PARSE
E_ERROR
E_ALL
E_STRICT
E_DEPRECATED

Script 3.5 Adjust a script's level of error reporting to give you more or less feedback on potential and existing problems. In my opinion, more is always better.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2     "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5     <meta http-equiv="Content-Type"
6         content="text/html; charset=utf-8"/>
7     <title>Your Feedback</title>
8 </head>
9 <body>
10 <?php // Script 3.5 - handle_form.php #3
11
12 ini_set ('display_errors', 1);
13 // Let me learn from my mistakes!
14 error_reporting (E_ALL | E_STRICT);
15 // Show all possible problems!
16
17 // This page receives the data from
18 feedback.html.
19 // It will receive: title, name, email,
20 response, comments, and submit in
21 $_POST.
22
23 // Create shorthand versions of the
24 variables:
25 $title = $_POST['title'];
26 $name = $_POST['name'];
27 $response = $_POST['response'];
28 $comments = $_POST['comments'];
29
30 // Print the received data:
31 print "<p>Thank you, $title $name, for
32 your comments.</p>";
33 <p>You stated that you found this
34 example to be '$response' and added:
35 <br />$comments</p>";
36
37 ?>
38 </body>
39 </html>
```

Using this information, you could add any of the following to a script:

```
error_reporting (0);
error_reporting (E_ALL);
error_reporting (E_ALL & ~E_NOTICE);
```

The first line says that no errors should be reported. The second requests that all errors be reported. The last example states that you want to see all error messages except notices (the `& ~` means *and not*). Keep in mind that adjusting this setting doesn't prevent or create errors; it just affects whether or not errors are reported.

It's generally best to develop and test PHP scripts using the highest level of error reporting possible. To accomplish that, declare that you want *all errors plus strict* error reporting:

```
error_reporting (E_ALL | E_STRICT);
```

The ~~E_ALL~~ setting does not include ~~E_STRICT~~, which is why that line says that all errors should be shown *or* (the vertical bar, called the *pipe*) strict errors should be shown. This latter setting takes reporting a step further, also raising notices for things that could be a problem in future versions of PHP. Let's apply this setting to the `handle_form.php` page.

To adjust error reporting in a script:

1. Open `handle_form.php` (Script 3.4) in your text editor or IDE, if it is not already.
2. After the `ini_set()` line, add the following (Script 3.5):

```
error_reporting (E_ALL | E_STRICT);
```
3. Save the file as `handle_form.php`.

continues on next page

4. Place the file in the proper directory for your PHP-enabled server and test it in your Web browser by submitting the form (A and B).

At this point, if the form is filled out completely and the `$_POST` indexes exactly match the names of the form elements, you shouldn't see any errors (as in the figures). If any problems exist, including any potential problems (thanks to `E_STRICT`), they should be displayed and reported.

TIP The PHP manual lists all the error-reporting levels, but those listed here are the most important.

TIP The code in this book was tested using the highest level of error reporting: `E_ALL` | `E_STRICT`.

Please complete this form to submit your feedback:

Name:

Email Address:

Response: This is... ☐ excellent ☐ okay ☒ boring

Comments:

A Try the form one more time...

Thank you, Ms. Blankenship, for your comments.

You stated that you found this example to be 'boring' and added:
Enough already!

B ...and here's the result (if filled out completely and without any programmer errors).

Manually Sending Data to a Page

The last example for this chapter is a slight tangent to the other topics but plays off the idea of handling form data with PHP. As discussed in the section “Choosing a Form Method,” if a form uses the GET method, the resulting URL is something like

`http://www.example.com/page.php?`
→ `some_var=some_value&age=20&...`

The receiving page (here, `page.php`) is sent a series of *name=value* pairs, each of which is separated by an ampersand (&). The whole sequence is preceded by a question mark (immediately after the handling script’s name).

To access the values passed to the page in this way, turn to the `$_GET` variable. Just as you would when using `$_POST`, refer to the specific name as an index in `$_GET`. In that example, `page.php` receives a `$_GET['some_var']` variable with a value of *some_value*, a `$_GET['age']` variable with a value of *20*, and so forth.

You can pass data to a PHP script in this way by creating an HTML form that uses the GET method. But you can also use this same idea to send data to a PHP page *without* the use of the form. Normally you’d do so by creating links:

```
<a href="page.php?id=22">Some Link</a>
```

That link, which could be dynamically generated by PHP, will pass the value *22* to `page.php`, accessible in `$_GET['id']`.

To try this for yourself, the next pair of scripts will easily demonstrate this concept, using a hard-coded HTML page.

To create the HTML page:

1. Begin a new document in your text editor or IDE, to be named **hello.html** (Script 3.6):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>Greetings!</title>
</head>
<body>
<!-- Script 3.6 - hello.html --
> <div><p>Click a link to say
→ hello:</p>
```

2. Create links to a PHP script, passing values along in the URL:

```
<ul>
    <li><a href="hello.php?
    → name=Michael">Michael</a></li>
    <li><a href="hello.php?
    → name=Celia">Celia</a></li>
    <li><a href="hello.php?
    → name=Jude">Jude</a></li>
    <li><a href="hello.php?
    → name=Sophie">Sophie</a></li>
</ul>
```

Script 3.6 This HTML page uses links to pass values to a PHP script in the URL (thereby emulating a form that uses the GET method).

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
        xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
6      <title>Greetings!</title>
7  </head>
8  <body>
9      <!-- Script 3.6 - hello.html -->
10     <div><p>Click a link to say hello:</p>
11
12     <ul>
13         <li><a href="hello.php?
            name=Michael">Michael</a></li>
14         <li><a href="hello.php?
            name=Celia">Celia</a></li>
15         <li><a href="hello.php?
            name=Jude">Jude</a></li>
16         <li><a href="hello.php?
            name=Sophie">Sophie</a></li>
17     </ul>
18
19     </div>
20 </body>
21 </html>
```

Click a link to say hello:

- [Michael](#)
- [Celia](#)
- [Jude](#)
- [Sophie](#)

A The simple HTML page, with four links to the PHP script.

The premise here is that the user will see a list of links, each associated with a specific name **A**. When the user clicks a link, that name is passed to **hello.html** in the URL **B**.

If you want to use different names, that's fine, but stick to one-word names without spaces or punctuation (or else they won't be passed to the PHP script properly, for reasons that will be explained in time).

3. Complete the HTML page:

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

4. Save the script as **hello.html** and place it within the proper directory on your PHP-enabled server.

5. Load the HTML page through a URL in your Web browser.

Although you can view HTML pages without going through a URL, you'll click links in this page to access the PHP script, so you'll need to start off using a URL here. Don't click any of the links yet, as the PHP script doesn't exist!

```
<!-- Script 3.6 - hello.html -->
<div><p>Click a link to say hello:</p>

<ul>
    <li><a href="hello.php?name=Michael">Michael</a></li>
    <li><a href="hello.php?name=Celia">Celia</a></li>
    <li><a href="hello.php?name=Jude">Jude</a></li>
    <li><a href="hello.php?name=Sophie">Sophie</a></li>
</ul>

</div>
```

B The HTML source of the page shows how values are being passed along in the URL for the four links.

To create the PHP script:

1. Begin a new document in your text editor or IDE, to be named **hello.php** (Script 3.7):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="Content-Type"
    → content="text/html;
    → charset=utf-8"/>
    <title>Greetings!</title>
</head>
<body>
```

2. Begin the PHP code:

```
<?php // Script 3.7 - hello.php
```

3. Address the error management, if desired:

```
ini_set ('display_errors', 1);
error_reporting (E_ALL | E_STRICT);
```

These two lines, which configure how PHP responds to errors, are explained in the pages leading up to this section. They may or may not be necessary for your situation but can be helpful.

4. Use the **name** value passed in the URL to create a greeting:

```
$name = $_GET['name'];
print "<p>Hello, <span
style=\"font-weight:
bold;\">$name</span>!</p>";
```

The **name** variable is sent to the page through the URL (see Script 3.6). To access that value, refer to **\$_GET['name']**. Again, you would use **\$_GET** (as opposed to **\$_POST**) because the value is coming from a GET request.

Script 3.7 This PHP page refers to the **name** value passed in the URL in order to print a greeting.

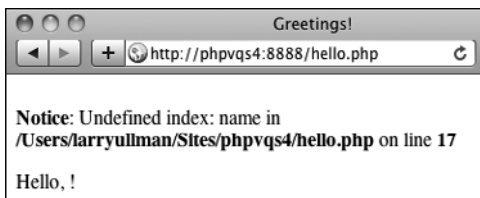
```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/
        xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/
        xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8"/>
6      <title>Greetings!</title>
7  </head>
8  <body>
9      <?php // Script 3.7 - hello.php
10
11      ini_set ('display_errors', 1);
        // Let me learn from my mistakes!
12      error_reporting (E_ALL | E_STRICT);
        // Show all possible problems!
13
14      // This page should receive a name value
        in the URL.
15
16      // Say "Hello":
17      $name = $_GET['name'];
18      print "<p>Hello, <span
        style=\"font-weight: bold;\">$name
        </span>!</p>";
19
20      ?>
21  </body>
22  </html>
```



C By clicking the first link, *Michael* is passed along in the URL and is greeted by name.



D By clicking the second link, *Celia* is sent along in the URL and is also greeted by name.



E If the `$_GET['name']` variable isn't assigned a value, the browser prints out this awkward message, along with the error notice.



F Any value assigned to `name` (lowercase) in the URL is used by the PHP script.

As with earlier PHP scripts, the value in the predefined variable (`$_GET`) is first assigned to another variable, to simplify the syntax in the `print` statement.

5. Complete the PHP code and the HTML page:

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

6. Save the script as **hello.php** and place it within the proper directory on your PHP-enabled server.

It should be saved in the same directory as **hello.html** (Script 3.6).

7. Click the links in **hello.html** to view the result **C** and **D**.

TIP If you run **hello.php** directly (i.e., without clicking any links), you'll get an error notice because no name value would be passed along in the URL **E**.

TIP Because **hello.php** reads a value from the URL, it actually works independently of **hello.html**. For example, you can directly edit the **hello.php** URL to greet anyone, even if **hello.html** does not have a link for that name **F**.

TIP If you want to use a link to send multiple values to a script, separate the `name=value` pairs (for example, `first_name=Larry`) with the ampersand (&). So, another link may be `hello.php? first_name=Larry&last_name=Ullman`. You should continue to use only single words, without punctuation or spaces, however (until you later learn about the `urlencode()` function).

TIP Although the example here—setting the value of a person's name—may not be very practical, this basic technique is useful on many occasions. For example, a PHP script might constitute a template, and the content of the resulting Web page would differ based on the values the page received in the URL.

Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum (www.LarryUllman.com/forum/).

Review

- What is the significance of a form's **action** attribute?
- What is the significance of a form's **method** attribute? Is it more secure to use GET or POST? Which method type can be bookmarked in the browser?
- What predefined variable will contain the data from a form submission? Note: There are multiple answers.
- Why must an HTML page that contains a form that's being submitted to a PHP script be loaded through a URL?
- Under what circumstances will attempts to enable **display_errors** in a script not succeed? Why is it not secure to enable **display_errors** on live sites?

Pursue

- Load **feedback.php** in your Web browser without going through a URL (i.e., the address bar would likely start with *file:///*). Fill out and submit the form. Observe the result so that you can recognize this problem, and understand its cause, in case you see similar results in the future.

- If you have not already, and if you can, make sure that **display_errors** is enabled on your development environment.
- If you have not already, and if you can, make sure that **error_reporting** is set to **E_ALL** | ~~**E_STRICT**~~ on your development environment.
- Try introducing different errors in a PHP script—by improperly balancing quotation marks, failing to use semicolons, referring to variables improperly, and so on—to see the result.
- Experiment with the **hello.html** and the **hello.php** pages to send different values, including numbers, to the PHP script through the URL.
- Create a variation on **hello.html** that sends multiple *name=value* pairs to a PHP script. Have the PHP script then print all the received values.
- If you're the inquisitive type, and don't mind waiting for answers, try passing more complicated values to a page through the URL. Try using spaces and punctuation to see what happens.
- Create a new HTML form that performs a task you envision yourself needing (or a lighter-weight version of that functionality). Then create the PHP script that handles the form, printing just the received data.