CHAPTER

Reading Data in Web Pages

This is the chapter a lot of developers have been waiting for—the chapter that covers connecting HTML controls in Web pages like text fields, radio buttons, check boxes and so on, to PHP back on the server.

You have the basics of PHP down at this point, which means you can handle the code necessary to work with PHP and HTML controls. So we're going to get connected to HTML in this chapter, seeing how to read data that the user has entered into HTML controls in Web pages.

Setting Up Web Pages to Communicate with PHP

To connect to PHP on the server, or any code on the server, you've got to set up your Web pages a particular way. You have to enclose all your HTML controls in an HTML form, and you have to indicate in that form where the data in those controls will be sent. For example, the user might enter their name in a text field, and you have to let the browser know where to send that name when the user clicks the Submit button.

Here's what an HTML form might look like:

```
<form>
        .
        .
        .
</form>
```

You need to specify the method with which your data will be sent—the two most common methods are "get" and "post" (more on the difference between them later—both methods will get your data to your PHP script), and you might choose the get method here:

```
<form method="get">
        .
        .
        .
</form>
```

Alright, this tells the browser how it's supposed to send the data in the Web page—and you also have to tell the browser where to send that data with an URL. You assign that URL to the <form> element's action attribute, like this:

```
<form method="get" action="http://www.phpisgreat.com/phpreader.php">
      .
      .
      .
</form>
```

This <form> element will send its data to the URL http://www.phpisgreat.com/phpreader.php when the Submit button (coming up in a moment) is clicked.

You can also assign relative URLs to the action attribute. If, for example, this HTML page resided on the server in a specific directory, and the PHP script phpreader.php resided in the *same* directory on the server, you could shorten the URL to this:

```
<form method="get" action="phpreader.php">
      .
      .
      .
</form>
```

This version of the <form> element assumes that the HTML page currently in the browser came from the same directory as the PHP script is in. For example, if this HTML page was http://www.phpisgreat.com/input.html, then the PHP script specified by simply assigning "phpreader.php" to the action attribute would send the data in the HTML page to the PHP script at http://www.phpisgreat.com/phpreader.php.

There's even another version—you can omit the action element altogether:

```
<form method="get">
      .
      .
      .
</form>
```

In this case, the form's data is sent back to the same URL that the current document is at. For example, if you have a PHP script that can display HTML controls, phpcomplete.php, then when you navigate to that script in your browser, you'll see those HTML controls. If there's no action attribute, the data in the form will be sent back to the exact same script when the user clicks the Submit button.

That's a common thing to do—have a PHP script handle both the display of the HTML controls, and then read the data in those HTML controls when the user clicks the Submit button. You'll see how this works in this book.

Besides the HTML controls like text fields and check boxes, you'll also need a submit button in your form, because the data in the form is sent to your PHP script when that submit button is clicked. The submit button need not have the caption "Submit"—you can set its caption to anything you like by assigning that caption to the submit button's value attribute.

Here's how to create a submit button—note it has to be inside the HTML <form> element—with the caption Send, and that you use an <input> element with the type attribute set to "submit" to create a submit button:

```
<html>
  <head>
    <title>
      Connecting to PHP
    </title>
  </head>
  <body>
    <h1>
      Connecting to PHP
    </h1>
    <form method="get" action="phpreader.php">
          .
          .
          .
      <input type="submit" value="Send">
      <input type="reset" value="Reset">
    </form>
  </body>
</html>
```

Note that there's also a (optional) reset button here; when clicked, this button resets the data in all the HTML controls in the form back to their default values.

Say you wanted to use a text field (that is, a <input type = "text"> HTML control) to ask for the user's name; you could do that like this in the HTML form:

```
<html>
  <head>
    <title>
      Connecting to PHP
    </title>
  </head>
  <body>
    <h1>
      Connecting to PHP
    </h1>
    <form method="get" action="phpreader.php">
      What's your name?

      <input name="data" type="text">
          .
          .
      <input type="submit" value="Send">
      <input type="reset" value="Reset">
    </form>
  </body>
</html>
```

Note that this HTML gives the name "data" to the text field. How can you access the name entered into this text field on the server, in your PHP script?

If you've used the POST method, you can find that data in the $_POST array, as we're going to start seeing in the next chunk on retrieving data from text fields. If you've used the GET method, you use the $_GET array. These arrays are "superglobal" arrays, which means that they're available to you without having to use the global keyword. Also, the $_REQUEST array holds data from both $_GET and $_POST.

That means that to recover the name the user entered into the text field named "data", you can use the expression $_REQUEST["data"] in your PHP script.

Let's put all this into practice with real text fields, coming up next.

Handling Text Fields

Say you've put together a Web page, phptext.html, that has a text field and a submit button in a form, as you see here:

```
<html>
  <head>
    <title>
      Entering data into text fields
    </title>
  </head>
  <body>
    <h1>
      Entering data into text fields
    </h1>
    <form method="get" action="phptext.php">
      What's your name?

      <input name="data" type="text">

      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

You can see this HTML page in Figure 5-1, waiting for you to enter your name.

Okay, so how do you read the data the user entered in this Web page—that is, their name in a text field we've named "data"—from PHP on the server?

This HTML page is set up to send its data to the PHP script phptext.php (note that because you've assigned a relative URL to the action attribute—just the name of the PHP script—that PHP script must be stored in the same directory on the server as phptext.html):

```
<form method="get" action="phptext.php">
  What's your name?

  <input name="data" type="text">

  <input type="submit" value="Send">
</form>
```


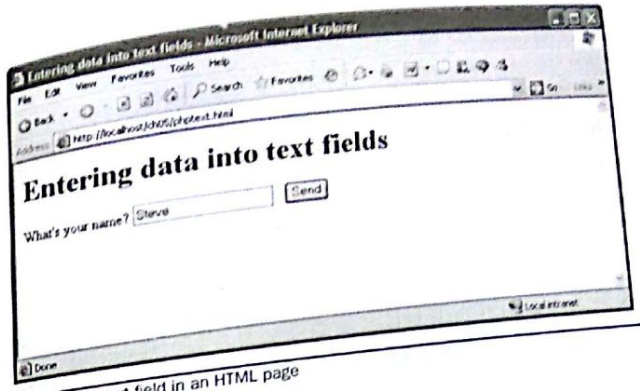

FIGURE 5-1 Using a text field in an HTML page

In phptext.php, you can access the text in the text field named "data" as $_REQUEST["data"]. So here's phptext.php:

```
<html>
  <head>
    <title>
      Reading data from text fields
    </title>
  </head>
  <body>
    <h1>
      Reading data from text fields
    </h1>
    Thanks for answering,
    <?php
      echo $_REQUEST["data"];
    ?>
  </body>
</html>
```

All this PHP script, phptext.php, does is to echo the name the user entered into the data text field. Make sure that phptext.php goes in the same directory as phptext.html on your server, and give this example a try. When you click the submit button in phptext.html, you see the results, something like what you see in Figure 5-2, where the user's name appears.

Note the URL in Figure 5-2: http://localhost/ch05/phptext.php?data=Steve. That URL includes the data the user entered into the text field, placed there following a question mark (?). The data sent with the "get" method is always URL-encoded (spaces are replaced with + signs, different controls name/data pairs are separated with an &), and placed into the URL.

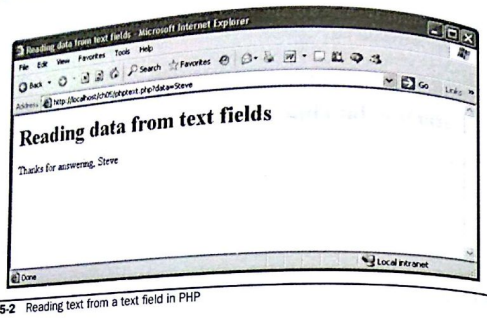

FIGURE 5-2 Reading text from a text field in PHP

The get method works, but as you can see, it can create less-than-professional results. If you don't want the user's data to appear in the URL they access the server with, use the "post" method instead, like this in phptext.html:

```
<html>
  <head>
    <title>
      Entering data into text fields
    </title>
  </head>
  <body>
    <h1>
      Entering data into text fields
    </h1>
    <form method="post" action="phptext.php">
      What's your name?

      <input name="data" type="text">

      <input type="submit" value="Send">
  form>
  y>
```

ou use this new version of phptext.html, the user's data is sent in the HTTP the browser sends to the server, instead of the URL. That results in a cleaner can see in the URL text area in the browser in Figure 5-3. Because posted data is TTP headers and not in the URL, data that you send with the post method is more secure.

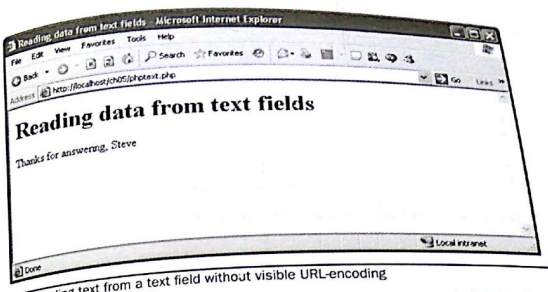

FIGURE 5-3 Reading text from a text field without visible URL-encoding

Excellent, you've been able to read data from a text field. A similar HTML control, text areas, is coming up next.

Handling Text Areas

Text fields are fine if you only want a single line of text, but if you want multiline text input, you have to go with text areas.

Here's an example, phptextarea.html, that presents the user with a text area, and asks what pizza toppings they want. It starts like this:

```
<html>
  <head>
    <title>
      Entering data into text areas
    </title>
  </head>
  <body>
    <h1>
      Entering data into text areas
    </h1>
    <form method="post" action="phptextarea.php">
      Enter the pizza toppings you want: <br>
        .
        .
        .
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

Then it adds a text area like this, where the numbers 1–4 will appear in the text area:

```
<html>
  <head>
    <title>
      Entering data into text areas
    </title>
  </head>
  <body>
    <h1>
      Entering data into text areas
    </h1>
    <form method="post" action="phptextarea.php">
      Enter the pizza toppings you want: <br>
      <textarea name="data" cols="50" rows="5">
1.
2.
3.
4.
      </textarea>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

You can see this page in Figure 5-4, waiting for you to enter your pizza toppings.

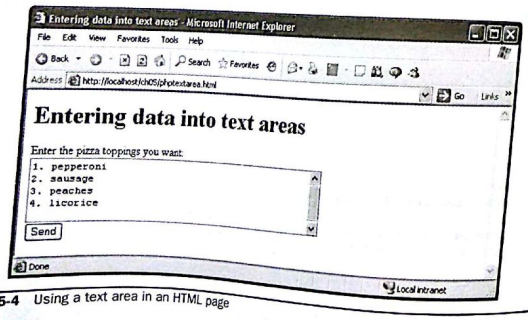

FIGURE 5-4 Using a text area in an HTML page

So how do you read the data the user entered in this Web page—in the text area named "data"—from PHP on the server? The data the user entered will be sent to phptextarea.php on the server, and you can access the text from the text area as $_REQUEST["data"] in that script:

```
<html>
  <head>
    <title>
      Reading data from text areas
    </title>
  </head>
  <body>
    <h1>
      Reading data from text areas
    </h1>
    You ordered a pizza with: <br>
    <?php
      $text = $_REQUEST["data"];
      .
      .
      .
    ?>
  </body>
</html>
```

Note that because you're dealing with a text area, multiline text will be filled with newline characters, \n. When you display that text, the browser is going to ignore the newlines, so you might replace them with
 elements instead, like this, where we display the user's pizza toppings in phptextarea.php:

```
<html>
  <head>
    <title>
      Reading data from text areas
    </title>
  </head>
  <body>
    <h1>
      Reading data from text areas
    </h1>
    You ordered a pizza with: <br>
    <?php
      $text = $_REQUEST["data"];
      echo str_replace("\n", "<br>", $text);
    ?>
  </body>
</html>
```

And you can see the results in Figure 5-5, where the application has not only echoed the text the user entered, but also preserved the multiline nature of that text. Nice.

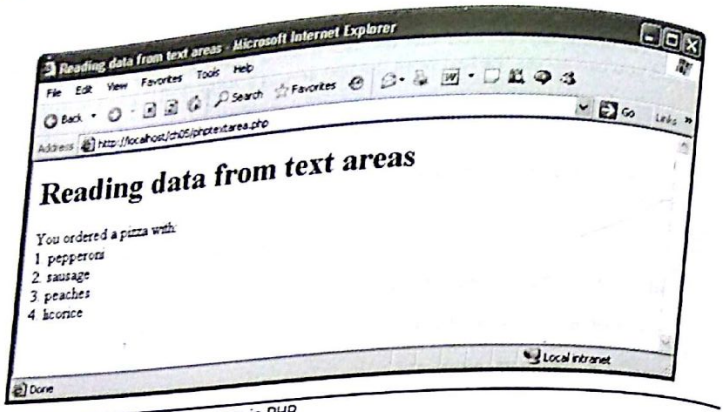

FIGURE 5-5 Reading text from a text area in PHP

Handling Check Boxes

The next step up in controls are check boxes—those square controls that you can select or de-select with the mouse.

You create check boxes with the <input> element like this in a Web page, where you're asking the user if they want fries in phpcheckbox.html:

```
<html>
  <head>
    <title>
      Entering data into check boxes
    </title>
  </head>
  <body>
    <h1>
      Entering data into check boxes
    </h1>
    <form method="post" action="phpcheckbox.php">
      Do you want fries with that?
      <input name="check1" type="checkbox" value="yes">
      Yes
      <input name="check2" type="checkbox" value="no">
      No
      <br>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

Note that the value of the first check box is "yes" and the value of the second is "no"—those are the values that will be sent to your script on the server.

And you can see the results in Figure 5-6, where the application is asking the user if they want fries.

How do you read the data from the check boxes, check1 and check2? You might think you could do that simply like this, where you simply use expressions like $_REQUEST["check1"]:

```
<html>
  <head>
    <title>
      Reading data from check boxes
    </title>
  </head>
  <body>
    <h1>
      You selected:
      <?php
        echo $_REQUEST["check1"], "<br>";
        echo $_REQUEST["check2"], "<br>";
      ?>
    </body>
</html>
```

Unfortunately, that's not right—the user may not have checked a check box, so attempting to display the data from that check box would give you an error in PHP. For example, if the user has not check check1, then echo $_REQUEST["check1"] will give you an error (because the array $_REQUEST doesn't have an element with the index "check1").

In this case, you have to first check if there is any data waiting for you from a particular check box before you attempt to display that data. You can check if an array has an element with a certain index with the isset function, so before echoing $_REQUEST["check1"], check

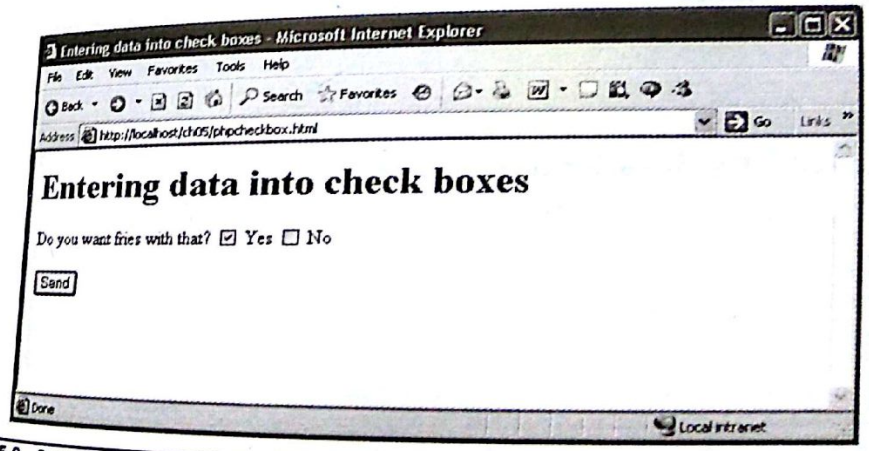

FIGURE 5-6 Setting check boxes

if that array element exists with isset($_REQUEST["check1"]). Here's how that works in phpcheckbox.php:

```
<html>
  <head>
    <title>
      Reading data from text fields
    </title>
  </head>
  <body>
    <h1>
      Reading data from check boxes
    </h1>
    You selected:
    <?php
      if (isset($_REQUEST["check1"])) {
        echo $_REQUEST["check1"], "<br>";
      }
      if (isset($_REQUEST["check2"])) {
        echo $_REQUEST["check2"], "<br>";
      }
    ?>
  </body>
</html>
```

And you can see the results in Figure 5-7, where you're reading what check boxes the user selected.

Note that the user could click both check boxes here—which would mean that they both wanted fries and didn't want fries. A better choice for the controls here are radio buttons, where only one can be selected at a time, and they're coming up next.

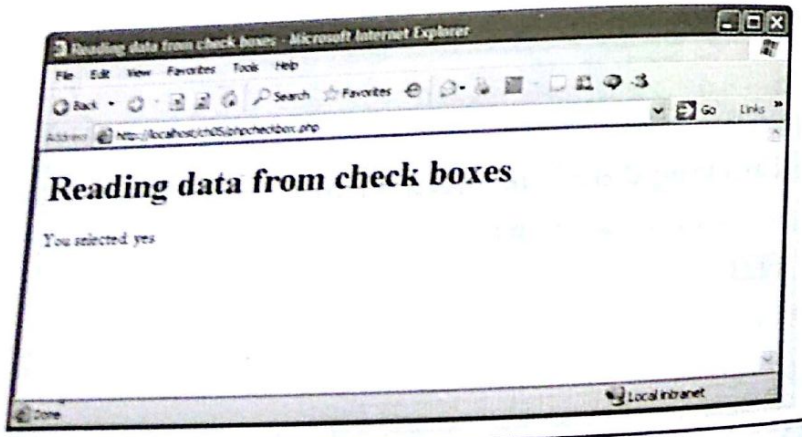

Figure 5-7 Reading check box data in PHP

Handling Radio Buttons

Check boxes are good if you want the user to be able to select multiple items from a number of choices. But if you want to let the user select only one item from a number of choices, you should use radio buttons instead, because they only allow the user to make one selection at a time.

Here's an example, phpradiobutton.html, that, like the preceding example, asks the user if they want fries—but unlike the previous example, this time, the user can only select yes or no, not both (note that we're giving both radiobuttons the same name here, "radios"):

```
<html>
  <head>
    <title>
      Entering data with radio buttons
    </title>
  </head>
  <body>
    <h1>
      Entering data with radio buttons
    </h1>
    <form method="post" action="phpradiobutton.php">
      Do you want fries with that?
      <input name="radios" type="radio" value="yes">
      Yes
      <input name="radios" type="radio" value="no">
      No
      <br>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

You can see this page in Figure 5-8, waiting for your answer.

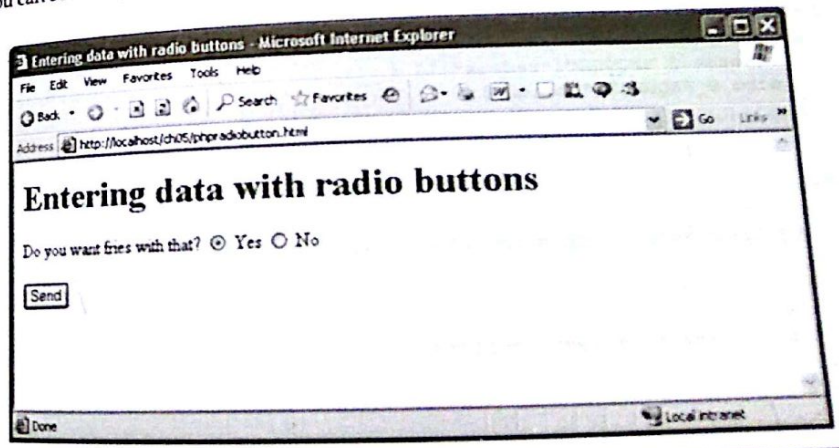

Figure 5-8 Using radio buttons in an HTML page

How do you read data from radio buttons? You can use $_REQUEST like this in phpradiobutton.php:

```
<html>
  <head>
    <title>
      Reading data from radio buttons
    </title>
  </head>
  <body>
    <h1>
      Reading data from radio buttons
    </h1>
    You selected
    <?php
      echo $_REQUEST["radios"];
    ?>
  </body>
</html>
```

But you'd run into the same problem as with the check boxes—the user may not have selected either radio button. On the other hand, there will only be one entry in the $_REQUEST array under the index "radios", so you only need one statement like this in phpradiobutton.php:

```
<html>
  <head>
    <title>
      Reading data from radio buttons
    </title>
  </head>
  <body>
    <h1>
      Reading data from radio buttons
    </h1>
    You selected
    <?php
      if (isset($_REQUEST["radios"])) {
        echo $_REQUEST["radios"];
      }
    ?>
  </body>
</html>
```

In fact, you can indicate that no radio button was selected if that was the case:

```
<head>
  <title>
    Reading data from radio buttons
  </title>
</head>
```


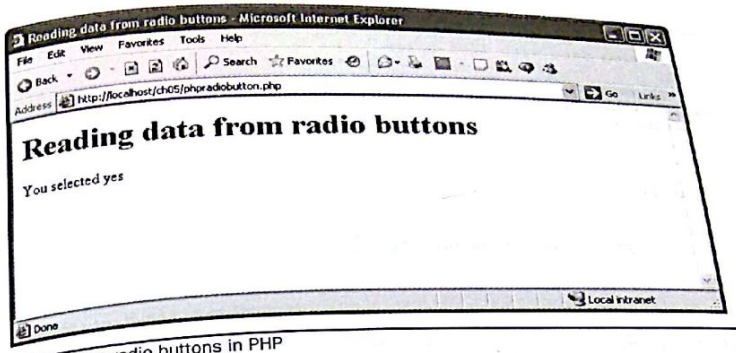

FIGURE 5-9 Reading radio buttons in PHP

```
<body>
  <h1>
    Reading data from radio buttons
  </h1>
  You selected
  <?php
    if (isset($_REQUEST["radios"])) {
      echo $_REQUEST["radios"];
    }
    else {
      echo "No radio button was selected. <br>";
    }
  ?>
</body>
</html>
```

You can see the results in Figure 5-9, where the application has accurately determined which radio button was clicked.

Handling List Boxes

List boxes are also a common HTML control, and they take a little special handling. Say that you wanted to let the user select their favorite flavors of ice cream; you might start like this in phplistbox.html, where you're creating the list box with a <select> HTML control:

```
<html>
  <head>
    <title>
      Entering data with list boxes
    </title>
  </head>
```

```
    <body>
      <h1>
        Entering data with list boxes
      </h1>
      Select your favorite ice cream flavors:
      <form method="post" action="phplistbox.php">
        <select>
          .
          .
          .
        </select>
        <br>
        <br>
        <input type="submit" value="Send">
      </form>
    </body>
</html>
```

To let the user select multiple ice cream flavors, we're going to make this a multiple select control, which you do in HTML with the stand-alone attribute multiple in the <select> element. And here's the trick that will let this multiple-selection control work with PHP—you give the control the name of an array, not just a name. For example, if this control were a single-selection control, you might call it "ice_cream"—but because it's a multiple-selection control, you call it "ice_cream[]", which tips PHP off that this is a control that allows multiple selections:

```
<html>
  <head>
    <title>
      Entering data with list boxes
    </title>
  </head>
  <body>
    <h1>
      Entering data with list boxes
    </h1>
    Select your favorite ice cream flavors:
    <form method="post" action="phplistbox.php">
      <select name="ice_cream[]" multiple>
        .
        .
        .
      </select>
      <br>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```


Now you're free to add the ice cream flavors as <option> elements inside the <select> control (the items in <select> controls are given as <option> elements in HTML):

```
<html>
  <head>
    <title>
      Entering data with list boxes
    </title>
  </head>
  <body>
    <h1>
      Entering data with list boxes
    </h1>
    Select your favorite ice cream flavors:
    <form method="post" action="phplistbox.php">
      <select name="ice_cream[]" multiple>
        <option>vanilla</option>
        <option>strawberry</option>
        <option>chocolate</option>
        <option>herring</option>
      </select>
      <br>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

You can see this page in Figure 5-10, waiting for your ice cream selections.

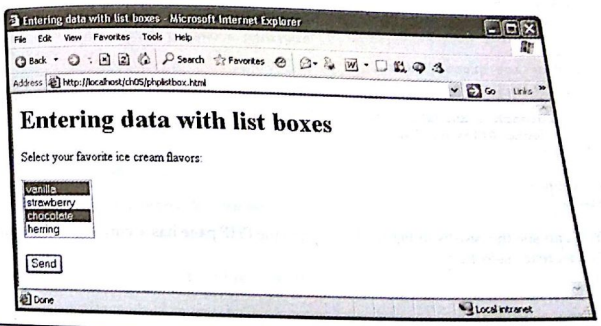

FIGURE 5-10 Using list boxes in an HTML page

Alright, what about reading the selections the user made? Up until now, you would have handled their selections like this, simply echoing $_REQUEST["ice_cream"] to the browser:

```
<head>
  <title>
    Reading data from list boxes
  </title>
</head>
<body>
  <h1>
    Reading data from list boxes
  </h1>
  Your ice cream flavors:
  <BR>
  <?php
    echo $_REQUEST["ice_cream"];
  ?>
</body>
</html>
```

But that won't work here, because ice_cream is an array, not a single variable. So you might use a foreach loop to display the user's ice cream selections like this in phplistbox.php:

```
<head>
  <title>
    Reading data from list boxes
  </title>
</head>
<body>
  <h1>
    Reading data from list boxes
  </h1>
  Your ice cream flavors:
  <BR>
  <?php
    foreach($_REQUEST["ice_cream"] as $flavor){
      echo $flavor, "<br>";
    }
  ?>
</body>
</html>
```

You can see the results in Figure 5-11, where the PHP page has accurately reported the user's ice cream selections.

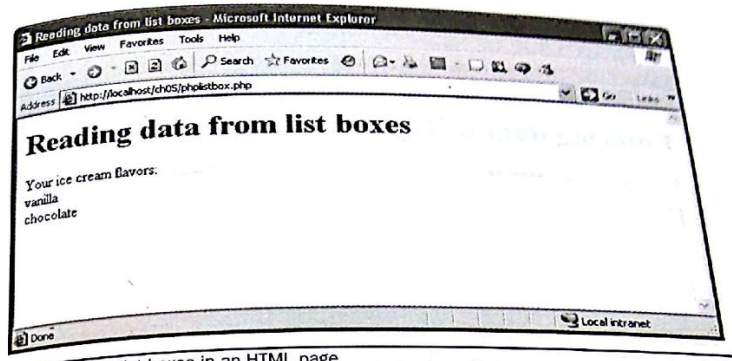

FIGURE 5-11 Reading list boxes in an HTML page

Handling Password Controls

A common use of PHP is to check passwords on the server, giving the user access to a resource if they have the right password, and you can use password controls for that.

Here's an example that asks the user for their password, phppassword.html:

```
<html>
  <head>
    <title>
      Entering data with password controls
    </title>
  </head>
  <body>
    <h1>
      Entering data with password controls
    </h1>
    <form method="post" action="phppassword.php">
      Enter your password:

      <input name="password" type="password">

      <br>
      <br>
      <input type="submit" value="Send">
    </form>
  </body>
</html>
```

You can see this page in Figure 5-12, where the user has entered their password.

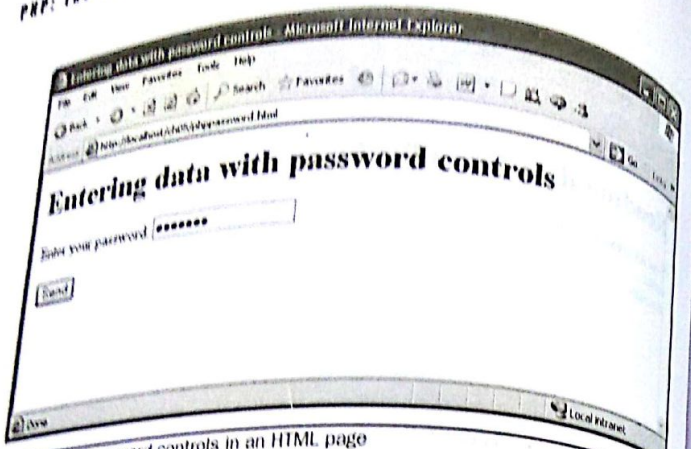

FIGURE 5-12 Using password controls in an HTML page

You can store the password on the server—which is the charm of using PHP for password verification. In this case, you check $_REQUEST["password"] against the password, which is "letmein" in this case, and if it matches, display a welcome page to the user in phppassword.php:

```
<head>
  <title>
    Reading data from password controls
  </title>
</head>
<body>
  <h1>
    Reading data from password controls
  </h1>
  <?php
    if ($_REQUEST["password"] == "letmein"){
  ?>
    <h2>
     Password accepted
    </h2>
    OK, you're in.<br>
    Please act responsibly.
      .
      .
      .
  ?>
</body>
</html>
```


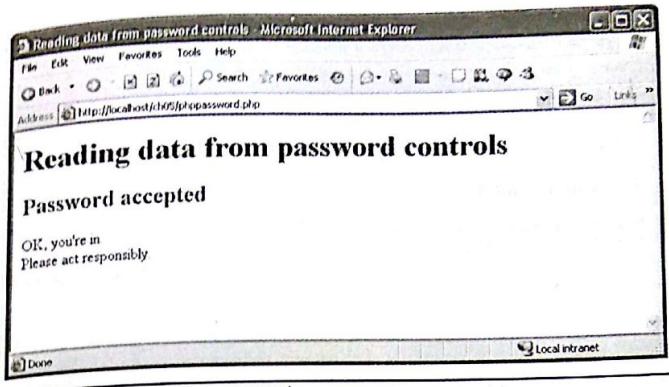

FIGURE 5-13 Gaining access with a password

You can see this result in Figure 5-13.

On the other hand, if the password the user entered is not right, you can display an error page—note how this page mixes HTML and PHP:

```
<head>
  <title>
    Reading data from password controls
  </title>
</head>
<body>
  <h1>
    Reading data from password controls
  </h1>
  <?php
    if ($_REQUEST["password"] == "letmein"){
  ?>
    <h2>
     Password accepted
    </h2>
    OK, you're in.<br>
    Please act responsibly.
  <?php
    }
    else {
  ?>
    <h2>
     Password denied
    </h2>
```

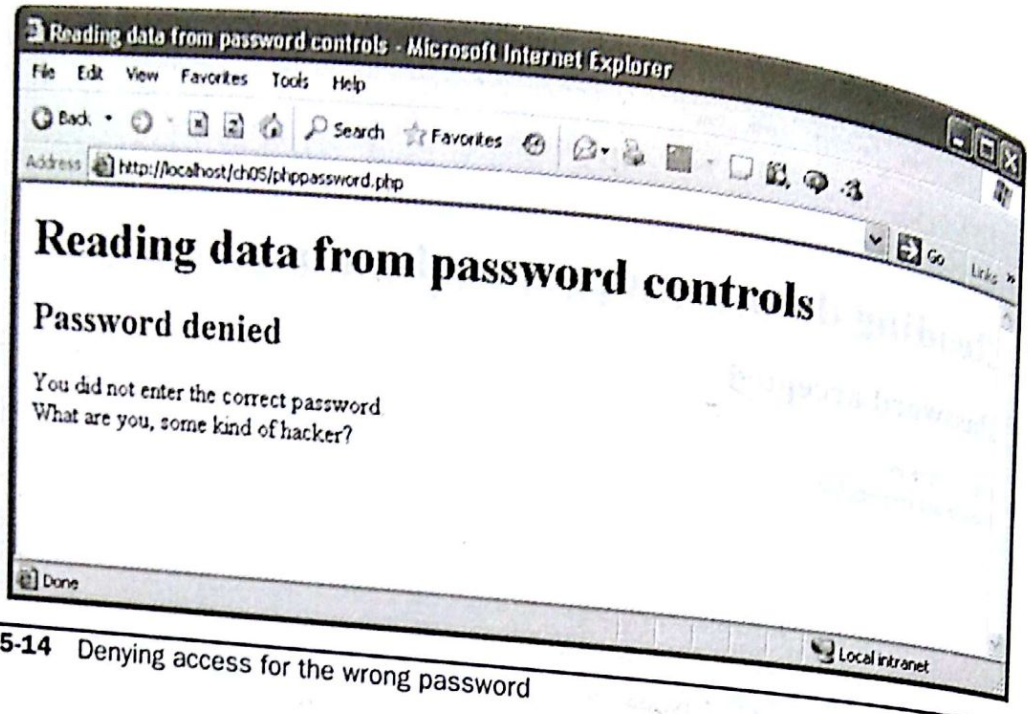

FIGURE 5-14 Denying access for the wrong password

```
      You did not enter the correct password.<br>
      What are you, some kind of hacker?
    <?php
      }
    ?>
  </body>
</html>
```

And you can see the error result in Figure 5-14.

Handling Hidden Controls

HTML hidden controls are a good match for PHP scripts, because they let you store data in Web pages that the user doesn't usually see, and that you can make use of on the server (the user can see hidden data if they look at a Web page's source).

Here's an example that uses a hidden control named customer_type to store what we think of the customer, behind the scenes. In this case, phphidden.php, customer_type is set to "good":

```
<html>
  <head>
    <title>
      Storing data with hidden controls
    </title>
  </head>
  <body>
    <h1>
      Storing data with hidden controls
    </h1>
```

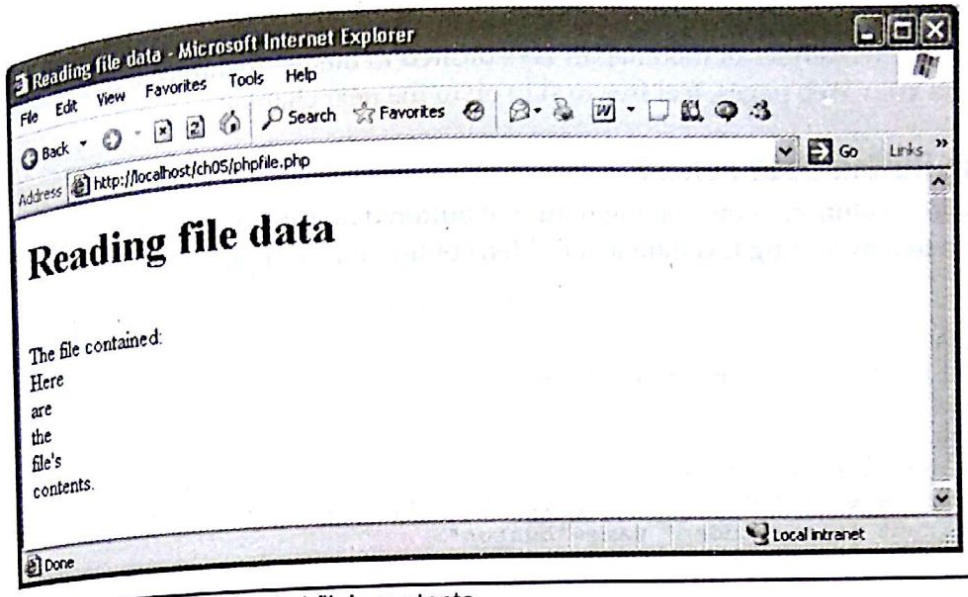

FIGURE 5-20 Displaying an uploaded file's contents

```
  <body>
    <h1>Reading file data</h1>
    <br>
    The file contained:
    <br>
    <?php
      $handle = fopen($_FILES['userfile']['tmp_name'], "r");
      while (!feof($handle)){
        $text = fgets($handle);
        echo $text, "<br>";
      }
      fclose($handle);
    ?>
  </body>
</html>
```

You can see all this at work in Figure 5-20, where the uploaded file's contents are displayed.

Although this example worked with a text file, you can also upload binary files, of course. All you have to do is to use the techniques for handling binary files coming up in the general discussion of file-handling in this book.

So now you've been able to upload files that the user passes to you. Cool.

Handling Buttons

There's still one popular HTML control that we haven't covered—buttons. Although you often see buttons in Web pages, they are more difficult to work with using server-side scripts for one reason—they pop back. That is, there's no data that's stored that will be sent to the server when a Submit button is clicked.

So how do you handle buttons in Web pages with PHP? There are a couple of solutions here, and the remainder of this chapter is dedicated to them—if you have little interest in buttons in your Web pages, feel free to skip on to the next chapter.

Making Button Data Persist

One obvious solution to the fleeting nature of button data is to make that data persist. You might do that by storing text data in a hidden control, for example:

```
<html>
  <head>
    <title>Handling buttons</title>
  </head>
  <body>
    <h1>Handling buttons</H1>
    <form name="form1" action="phpbuttons.php" method="post">
      <input type="hidden" name="button">
        .
        .
        .
    </form>
  </body>
</html>
```

Then you can add, say, three standard HTML buttons to the page:

```
<html>
  <head>
    <title>Handling buttons</title>
  </head>
  <body>
    <h1>Handling buttons</H1>
    <form name="form1" action="phpbuttons.php" method="post">
      <input type="hidden" name="button">
      <input type="button" value="Button 1" onclick="setbutton1()">
      <input type="button" value="Button 2" onclick="setbutton2()">
      <input type="button" value="Button 3" onclick="setbutton3()">
    </form>
  </body>
</html>
```

So how do you store data in the hidden control when the user clicks a button? Since these are standard HTML buttons, all the action has to take place in the browser—so that means using a browser-side scripting language like JavaScript; there's no other choice. Here's how you can store the name of the button that was clicked in the hidden control:

```
<html>
  <head>
    <title>Handling buttons</title>
    <script language="JavaScript">
      function setbutton1()
```



```
      {
        document.form1.button.value = "button 1"
        .
        .
        .
      }

      function setbutton2()
      {
        document.form1.button.value = "button 2"
        .
        .
        .
      }

      function setbutton3()
      {
        document.form1.button.value = "button 3"
        .
        .
        .
      }
    </script>
  </head>
  <body>
    <h1>Handling buttons</H1>
    <form name="form1" action="phpbuttons.php" method="post">
      <input type="hidden" name="button">
      <input type="button" value="Button 1" onclick="setbutton1()">
      <input type="button" value="Button 2" onclick="setbutton2()">
      <input type="button" value="Button 3" onclick="setbutton3()">
    </form>
  </body>
</html>
```

After the name of the button has been stored in the hidden field, you can complete the process by submitting the form from JavaScript:

```
<html>
  <head>
    <title>Handling buttons</title>
    <script language="JavaScript">
      function setbutton1()
      {
        document.form1.button.value = "button 1"
        form1.submit()
      }

      function setbutton2()
      {
        document.form1.button.value = "button 2"
        form1.submit()
      }
```

```
        function setbutton3()
        {
          document.form1.button.value = "button 3"
          form1.submit()
        }
      </script>
    </head>
    <body>
      <h1>Handling buttons</H1>
      <form name="form1" action="phpbuttons.php" method="post">
        <input type="hidden" name="button">
        <input type="button" value="Button 1" onclick="setbutton1()">
        <input type="button" value="Button 2" onclick="setbutton2()">
        <input type="button" value="Button 3" onclick="setbutton3()">
      </form>
    </body>
</html>
```

You can see this page, phpbuttons.php, in Figure 5-21.

Reading the data now that you've stored it in a hidden field is easy using PHP. Here's what the script that reads the button you've clicked, phpbuttons.php, looks like:

```
<html>
  <head>
    <title>
      Reading buttons
    </title>
  </head>
  <body>
    <h1>Reading buttons</h1>
```

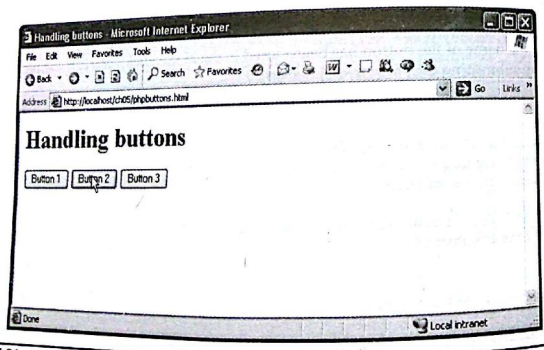

FIGURE 5-21 Buttons in a Web page

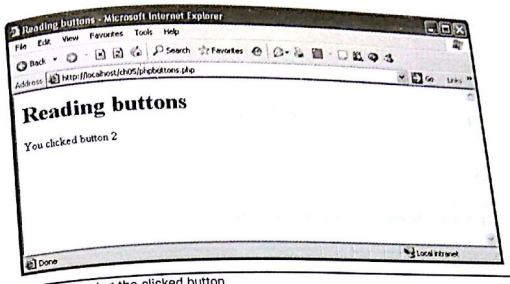

FIGURE 5-22 Displaying the clicked button

```
    You clicked
    <?php
      if (isset($_REQUEST["button"])) {
        echo $_REQUEST["button"], "<br>";
      }
    ?>
  </body>
</html>
```

You can see the result in Figure 5-22, where the PHP script has correctly identified the clicked button.

However, this is a book on PHP, not on JavaScript. Isn't there a better way to handle buttons in Web pages?

Using Submit Buttons as HTML Buttons

Since the action in PHP takes place on the server, you can also use submit buttons in place of standard HTML buttons. Submit buttons look the same as HTML buttons, so the user won't be any wiser.

Here's a way you might use Submit buttons to mimic HTML buttons—create three forms for three buttons:

```
<html>
  <head>
    <title>Reading submit buttons</title>
  </head>

  <body>
    <h1>Reading submit buttons</h1>
    <form name="form1" action="phpsubmit.php" method="post">
```

```
      <input type="submit" value="Button 1">
        .
        .
        .
    </form>
    <form name="form2" action="phpsubmit.php" method="post">
      <input type="submit" value="Button 2">
        .
        .
        .
    </form>
    <form name="form3" action="phpsubmit.php" method="post">
      <input type="submit" value="Button 3">
        .
        .
        .
    </form>
  </body>
</html>
```

Then give each form its own hidden control with the button's name like this in phpsubmit.html:

```
<html>
  <head>
    <title>Reading submit buttons</title>
  </head>

  <body>
    <h1>Reading submit buttons</h1>
    <form name="form1" action="phpsubmit.php" method="post">
      <input type="hidden" name="button" value="button 1">
      <input type="submit" value="Button 1">
    </form>

    <form name="form2" action="phpsubmit.php" method="post">
      <input type="hidden" name="button" value="button 2">
      <input type="submit" value="Button 2">
    </form>

    <form name="form3" action="phpsubmit.php" method="post">
      <input type="hidden" name="button" value="button 3">
      <input type="submit" value="Button 3">
    </form>
  </body>
</html>
```

You can see the result in Figure 5-23, where the three apparent "buttons" are really Submit buttons.

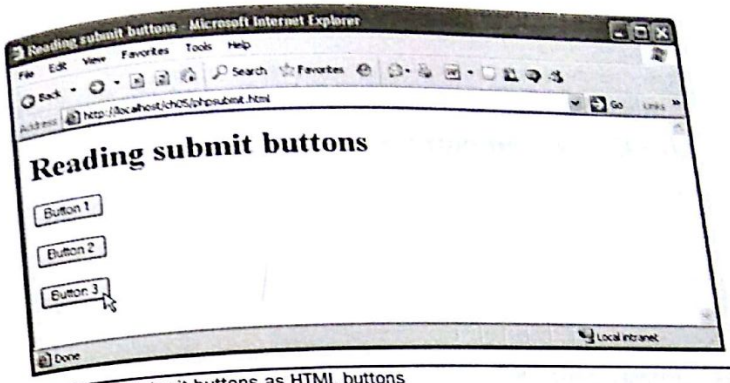

FIGURE 5-23 Using Submit buttons as HTML buttons

Now all you need to do is to read the value stored in the hidden control in your PHP script to determine which button was clicked, like this in phpsubmit.php:

```
<html>
  <head>
    <title>
      Reading submit buttons
    </title>
  </head>
  <body>
    <h1>
      Reading submit buttons
    </h1>
    You clicked
    <?php
      if (isset($_REQUEST["button"])) {
        echo $_REQUEST["button"], "<br>";
      }
    ?>
  </body>
</html>
```

You can see the result in Figure 5-24, where the PHP script has correctly identified the clicked button. Not bad.

In fact, there's an easier way to use Submit buttons as standard HTML buttons—you can actually read the values (that is, the captions) of Submit buttons in PHP. So you don't need any hidden controls to hold the name of each button, like this in phpsubmit2.html:

```
<html>
  <head>
    <title>
      Using submit buttons with values
    </title>
  </head>
```

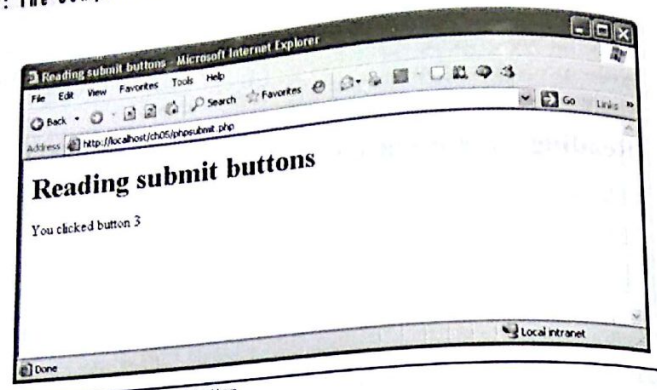

FIGURE 5-24 Identifying a clicked button

```
  <body>
    <h1>
      Using submit buttons with values
    </h1>
    <form name="form1" action="phpsubmit2.php" method="post">
      <input type="submit" name="button" value="button 1">
    </form>

    <form name="form2" action="phpbuttons2.php" method="post">
      <input type="submit" name="button" value="button 2">
    </form>

    <form name="form3" action="phpbuttons2.php" method="post">
      <input type="submit" name="button" value="button 3">
    </form>
  </body>
</html>
```

You can see this page, phpsubmit2.html, in Figure 5-25.

Now you can simply read the value of each Submit button in phpsubmit2.php:

```
<html>
  <head>
    <title>
      Reading submit buttons
    </title>
  </head>
  <body>
    <h1>
      Reading submit buttons
    </h1>
```


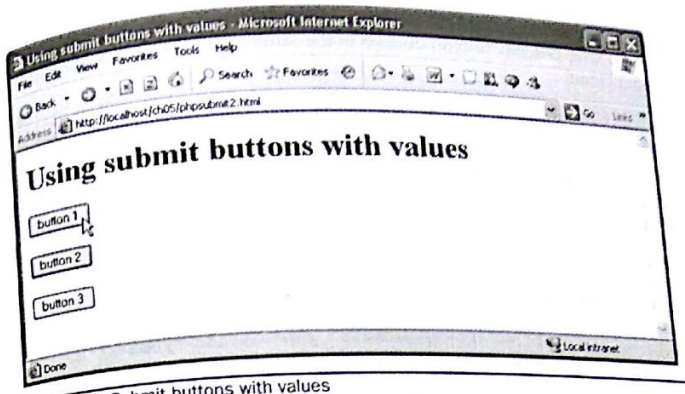

FIGURE 5-25 Using Submit buttons with values

```
    You clicked
    <?php
      if (isset($_REQUEST["button"])) {
        echo $_REQUEST["button"], "<br>";
      }
    ?>
  </body>
</html>
```

And you can see the results from phpsubmit2.php in Figure 5-26, where the correct button was identified.

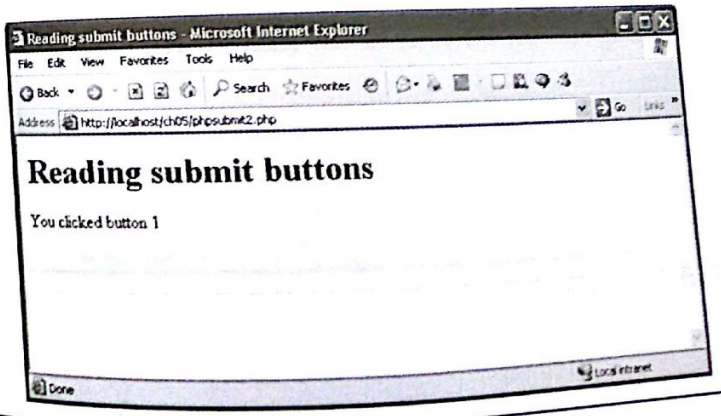

FIGURE 5-26 Reading Submit buttons with values

In fact, you can make this even simpler—you don't need three separate forms here; you can use three different Submit buttons in the same form. That looks like this in phpsubmit3.html:

```
<html>
  <head>
    <title>
      Using submit buttons in the same form
    </title>
  </head>

  <body>
    <h1>
      Using submit buttons in the same form
    </h1>
    <form name="form1" action="phpsubmit3.php" method="post">
      <input type="submit" name="button" value="button 1">

      <input type="submit" name="button" value="button 2">

      <input type="submit" name="button" value="button 3">
    </form>
  </body>
</html>
```

You can see this page in Figure 5-27.

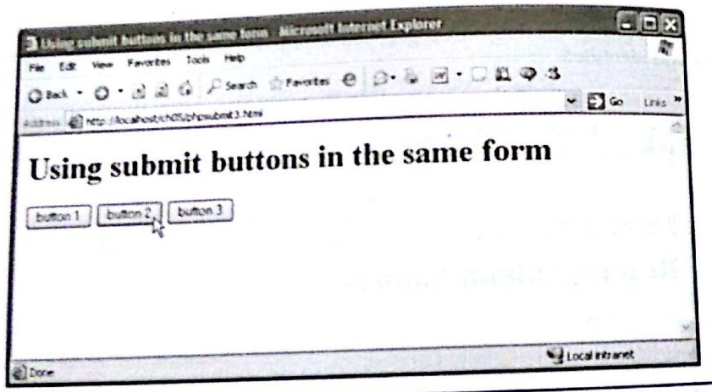

Figure 5-27 Multiple Submit buttons in the same form

Then you can simply read the value of the clicked Submit button in phpsubmit3.php:

```
<html>
  <head>
    <title>
      Reading submit buttons
    </title>
  </head>
  <body>
    <h1>
      Reading submit buttons
    </h1>
    You clicked
    <?php
      if (isset($_REQUEST["button"])) {
        echo $_REQUEST["button"], "<br>";
      }
    ?>
  </body>
</html>
```

And you can see the result in Figure 5-28, where phpsubmit3.php did its thing correctly—determined which button was clicked.

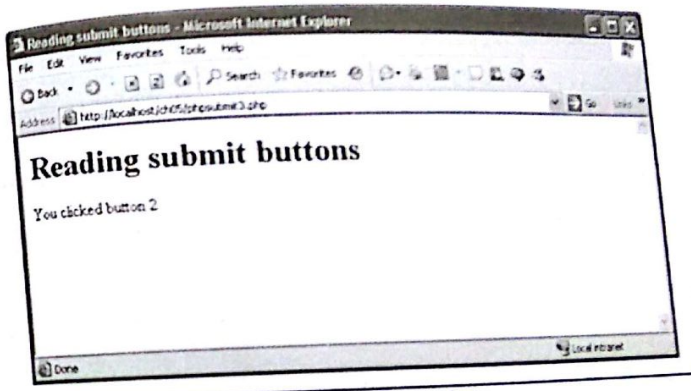

Figure 5-28 Reading multiple Submit buttons in the same form