Chapter 1: Building and Processing Dynamic Forms

In This Chapter

- **Using HTML forms with PHP**
- **Getting data from an HTML form**
- **Displaying data in a form**
- **Processing what users type into HTML forms**
- **Building the code for a real form**

A *dynamic Web site* is one in which the visitor interacts with the Web site. The Web site visitor provides information to the Web site in an HTML form, and the Web site performs actions based on the visitor's information. The Web site might display different Web pages based on the user's information or might store or use the information.

In this chapter, we don't tell you about the HTML required to display a form. We assume that you already know HTML. (If you don't know HTML or need a refresher, check out *HTML 4 For Dummies,* 5th Edition, by Ed Tittel and Mary Burmeister, from Wiley Publishing.) What we do tell you is how to use PHP to display HTML forms and to process the information that users type into the form.

Using Static HTML Forms

HTML forms are very important for interactive Web sites. In your previous experience, you might have displayed static HTML forms on Web pages. That is, forms whose content is predetermined and cannot change. In this section, you see how to display a static HTML form from a PHP script.

Displaying an HTML form

To display a form with PHP, you can do one of the following:

- **Use `echo` statements to echo the HTML for a form.** For example:

```
<?php
  echo "<form action='process_form.php'
              method='POST'>\n
     <input type='text' name='fullname' />\n
     <input type='submit' value='Submit Name' />\n
     </form>\n";
?>
```

- **Use plain HTML outside the PHP sections.** For a plain static form, there's no reason to include it in a PHP section. For example:

```
<?php
   statements in PHP section
?>
<form action="process_form.php" method="POST">
  <input type="text" name="fullname" />
  <input type="submit" value="Submit Name" />
</form>
<?php
  statements in PHP section
?>
```

Either of these methods produces the form displayed in Figure 1-1.

Figure 1-1: A form produced by HTML statements.

Getting information from the form

Joe Customer fills in the HTML form. He clicks the Submit Name button. You now have the information that you wanted — his name. So where is it? How do you get it?

You get the form information by running a script that receives the form information. When the user clicks the submit button, PHP automatically runs a script. The `action` parameter in the form tag tells PHP which script to run. For instance, in the preceding script, the parameter `action=process_form.php` tells PHP to run the script named `process_form.php` when the user clicks the submit button. The script name supplied in the `action` attribute can be any script you write to display, store, or otherwise use the form data it receives when the form is submitted. You can name it with any valid PHP script name.

When the user clicks the submit button, the script specified in the `action` attribute runs, and statements in this script can get the form information from PHP built-in arrays and use the information in PHP statements. The built-in arrays that contain form information are `$_POST`, `$_GET`, and `$_REQUEST`, which are *superglobal arrays*, special-purpose arrays that you can use anywhere in your script.

When the form uses the `POST` method, the information from the form fields is stored in the `$_POST` array. The `$_GET` array contains the variables passed as part of the URL, including fields passed from a form using the `GET` method. The `$_REQUEST` array contains all the array elements together that are contained in the `$_POST`, `$_GET`, and `$_COOKIES` arrays. Cookies are explained in Chapter 2 in this minibook.

When using PHP, it's almost always preferable to use the `POST` method for forms. When you use the `POST` method, the form data is passed as a package in a separate communication with the processing program, allowing an unlimited amount of data to be passed. When you use the `GET` method, in which the form data is passed in the URL, the amount of data that can be passed is limited. In addition, the `POST` method is more secure because the form data isn't displayed in the URL as it is with the `GET` method.

When the form is submitted, the script that runs can get the form information from the appropriate built-in array. In these built-in arrays, each array index is the name of the input field in the form. For instance, if the user typed **John Smith** in the input field shown in Figure 1-1 and clicked the submit button, the script `process_form.php` runs and can use an array variable in the following format:

```
$_POST['fullname']
```

Notice that the name typed into the form is available in the `$_POST` array because the `<form>` tag specified `method='POST'`. Also, note that the array key is the name given the field in the HTML form with the `name` attribute `name="fullname"`.

Registering long arrays

The superglobal arrays were introduced in PHP 4.1. Until that time, form information was passed in old arrays named `$HTTP_POST_VARS` and `$HTTP_GET_VARS`. It's very unlikely that you'll need to use them unless you're using some old scripts containing the long variables.

A `php.ini` setting, introduced in PHP 5, controls whether the old arrays are created. The following line in `php.ini` controls this setting:

```
register_long_arrays = On
```

In PHP 5, this setting is `On` by default. Unless you're running old scripts that need the old arrays, you should change the setting to `Off` so that PHP doesn't do this extra work.

In PHP 6, the `register_long_arrays` setting is removed from `php.ini`. The long arrays no longer exist. If you're using old scripts, you must change the long array names, such as `$HTTP_POST_VARS`, to the newer global array names, such as `$_POST`.

The superglobal arrays, including `$_POST` and `$_GET`, were introduced in PHP 4.1. Up until that time, form information was passed in old arrays named `$HTTP_POST_VARS` and `$HTTP_GET_VARS`. If you're using PHP 4.0 or earlier, you must use the long arrays. Both types of built-in arrays exist up through PHP 5. The long arrays no longer exist in PHP 6. If you're working with some old scripts that use the long array names, you need to change the array names from the long names, such as `$HTTP_POST_VARS`, to the superglobal array names, such as `$_POST`. In most cases, a search-and-replace in a text editor will make the change with one command per array.

A script that displays all the fields in a form is useful for testing a form. You can see what values are passed from the form to be sure that the form is formatted properly and that the form sends the field names and values that you expect.

Listing 1-1 shows a script that displays the information from all the fields sent by a `POST` type form when the user clicks the submit button. This script displays the field values from any form. We've named this script `process_form.php`. When the form shown earlier in Figure 1-1 is submitted, the script in Listing 1-1 runs.

Listing 1-1: A Script That Displays All the Fields from a Form

```
<?php
/*  Script name:  process_form.php
 *  Description:  Script displays all the information
 *                passed from a form.
 */
```

```
  echo "<html>
        <head><title>Form Fields</title></head>
        <body>";
  echo "<ol>";
  foreach($_POST as $field => $value)
  {
     echo "<li> $field = $value</li>";
  }
  echo "</ol>";
?>
</body></html>
```

If the user types the name **John Smith** into the form in Figure 1-1, the following output is displayed:

```
1 fullname = John Smith
```

The output displays only one line because the form in Figure 1-1 has only one field.

The script in Listing 1-1 is written to process the form information from any form that uses the `POST` method.

Organizing scripts that display forms

Best practices for PHP scripts suggest that code be organized into separate scripts, as follows:

- **Logic code:** The PHP code that performs the tasks for the scripts, which includes the `if` and `while` statements that control the flow of the script.
- **Display code:** The code that determines the look and feel of the Web pages, which includes the HTML and CSS code that defines the Web page.

It's easier to maintain and modify the script with separate files. When you want to change the look of the form, you need to edit only the file containing the code that defines the form. You won't accidentally change the logic of the script. By the same token, you can change the logic of the script without affecting the appearance of the form. In addition, if you need to display the form in different places in the script, you just include the file that defines the form wherever you need to display it. You don't add the entire code that displays the form inside your PHP script.

Script that contains the PHP logic

A simple logic script to display a form is shown in Listing 1-2.

Listing 1-2: A Script That Displays a Form

```
<?php
/*  Program name: display_form.php
 *  Description:  Script displays a form.
 */
include("form_phone.inc");
?>
```

The script consists of one statement that includes the file that displays the form. All the code that actually displays the form is in the script `form_phone.inc`, shown in Listing 1-3. This script is a simple case, with one include statement. A slightly more complicated script might include an `if` statement that displayed alternative forms, such as the following:

```
if(country == "Russia")
    include("form_Russian.inc");
elseif(country -- "USA")
    include("form_English.inc");
```

The logic script often includes statements that check the form information for errors after it's been submitted and statements that process the information. Validating and processing form information are discussed later in this chapter, in the section "Processing Information from the Form."

Script that contains the display code

The form script contains the HTML and CSS code that displays the Web page that includes the form. The look and feel of the Web page are defined in this file. Listing 1-3 shows the file that the logic script in Listing 1-2 includes in order to display a form that collects a Web visitor's phone number.

Listing 1-3: A Script That Defines a Form

```
<?php                                                          →1
/*  Program name: form_phone.inc
 *  Description:  Defines a form that collects a user's
 *                name and phone number.
 */
$labels = array( "first_name" => "First Name",                 →6
                 "middle_name" => "Middle Name",
                 "last_name" => "Last Name",
                 "phone" => "Phone");
$submit = "Submit Phone Number";                               →10
?>
 <html>                                                        →12
 <head><title>Customer Phone Number</title>
    <style type='text/css'>
    <!--
     #form {
```

```
      margin: 1.5em 0 0 0;
      padding: 0;
    }
    #field {padding-bottom: 1em;}
    label {
    font-weight: bold;
    float: left;
    width: 20%;
    margin-right: 1em;
    text-align: right;
    }
   -->
  </style>
 </head>
 <body>
<h3>Please enter your phone number below.</h3>
<form action='process_form.php' method='POST'>
<div id='form'>
<?php                                                          →35
  /* Loop that displays the form fields */
  foreach($labels as $field => $label)                         →37
  {
    echo "<div id='field'><label for='$field'>$label</label>
          <input id='$field' name='$field' type='text'
                 size='50%' maxlength='65' /></div>\n";
  }
  echo "</div>\n";
  echo "<input style='margin-left: 33%' type='submit'          →44
          value='$submit' />\n";
?>
</form></body></html>
```

The following explanation refers to the line numbers in the preceding code:

→1 The script begins with a PHP section (Lines 1–11).

→6 An array is created that contain the field names and labels used in the form. The keys are the field names. Setting up your fields in an array at the top of the script makes it easy to see what fields are displayed in the form and to add, remove, or modify fields.

→10 Creates a `$submit` variable $submit. The value assigned is the value displayed on the submit button.

→12 An HTML section (Lines 12–31) follows the PHP section. It includes the CSS needed to style the form (Lines 12–31) and three lines (Lines 32–34) that start the form, including the `<form>` tag. Notice that the `<form>` tag specifies `process_form.php` in the `action` attribute, meaning that the script `process_form.php` (Listing 1-1) is assigned to run and process the form when the user clicks the submit button.

→35 A second PHP section begins.

→37 A `foreach` statement begins. A form field is displayed for each element in the `$labels` array.

→44 Displays the submit button.

For security reasons, always include `maxlength` — which defines the number of characters that users are allowed to type into the field — in your HTML statement. Limiting the number of characters helps prevent the bad guys from typing malicious code into your form fields. If the information will be stored in a database, set `maxlength` to the same number as the width of the column in the database table.

When you run the script in Listing 1-2, the script in Listing 1-3 is included and displays the form shown in Figure 1-2.

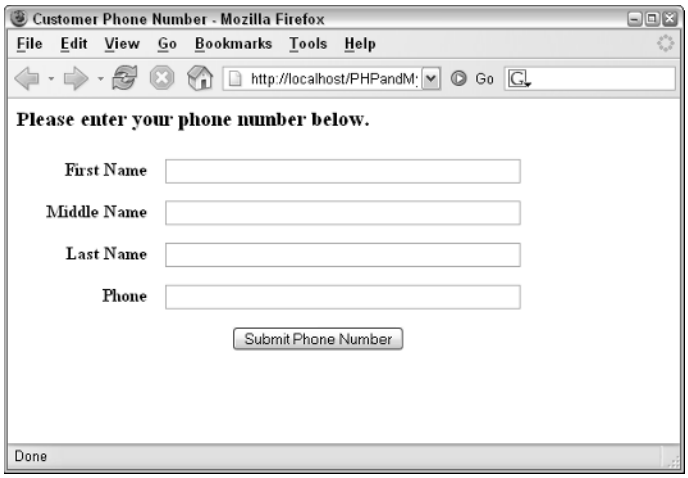

Figure 1-2: A form that collects a user's name and phone number.

When a user fills in the form shown in Figure 1-2 (created by the script in Listing 1-3) and submits it, the script `process_form.php` runs and produces output similar to the following:

```
1. first_name = Mary
2. middle_name = Quite
3. last_name = Contrary
4. phone = 555-5555
```

In `processform.php`, all elements of the `$_POST` built-in array are displayed.

Displaying Dynamic HTML Forms

PHP brings new capabilities to HTML forms. Because you can use variables in PHP forms, your forms can now be dynamic: They can be formatted at the time they are generated, rather than predetermined ahead of time as static forms are. The content of the form can change, based on information supplied by the user or information retrieved from the database. Here are the major capabilities that PHP brings to forms:

- Using variables to display information in input text fields
- Using variables to build dynamic lists for users to select from
- Using variables to build dynamic lists of radio buttons
- Using variables to build dynamic lists of check boxes

Displaying values in text fields

When you display a form on a Web page, you can put information into the text fields rather than just displaying a blank field. For example, if most of your customers live in the United States, you might automatically enter **US** in the country field when you ask customers for an address. If the customer does indeed live in the United States, you've saved the customer some typing. And if the customer doesn't live in the United States, he or she can just select the appropriate country. Also, the text automatically entered into the field doesn't have any typos — well, unless you included some yourself.

To display a text field that contains information, you use the following format for the input field HTML statements:

```
<input type="text" name="country" value="US">
```

This displays the value US in the field. By using PHP, you can make the form dynamic by using a variable to display this information, as shown in the two following statements:

```
<input type="text" name="country"
             value="<?php echo $country ?>" />

echo "<input type='text' name='country' value='$country' />";
```

The first example creates an input field in an HTML section, using a short PHP section for the value only. The second example creates an input field by using an `echo` statement inside a PHP section. If you're using a long form with only an occasional variable, using the first format is more efficient. If your form uses many variables, it's more efficient to use the second format.

If you have user information stored in a database, you might want to display the information from the database in the form fields. For instance, you might show the information to the user so that he or she can make any needed changes. Or you might display the shipping address for the customer's last online order so that he or she doesn't need to retype the address. Listing 1-4 shows the `form_phone_values_db.inc` file, containing the form code that displays a form with information from the database. This form is similar to the form shown in Figure 1-2, except that this form has information in it (retrieved from the database), and the fields in the form are blank.

To display this form, you run the `displayForm.php` script, shown in Listing 1-2, with the `include` statement in the script changed to:

```
include("form_phone_values_db.inc");
```

This includes the file that displays the form produced by the code in Listing 1-4.

Listing 1-4: Displaying an HTML Form with Information

```
<?php
/*  Program name: form_phone_values_db.inc
 *  Description:  Defines a form that gets a user's
 *                name and phone number from the database
 *                and displays them in a form.
 */
$labels = array ( "first_name" => "First Name",
                  "middle_name" => "Middle Name",
                  "last_name" => "Last Name",
                  "phone" => "Phone");
$submit = "Submit Phone Number";
$last_name = "Contrary";     // user name                    →12
include("dbstuff.inc");                                      →13
$cxn = mysqli_connect($host,$user,$passwd,$databname)
         or die ("couldn't connect to server");
$query = "SELECT * FROM phone
                   WHERE last_name='$last_name'";
$result = mysqli_query($cxn,$query)
          or die ("Couldn't execute query.");
$customer = mysqli_fetch_assoc($result);                     →20
?>
<html>
<head><title>Customer Phone Number</title>
  <style type='text/css'>
    <!--
     #form {
       margin: 1.5em 0 0 0;
       padding: 0;
     }
     #field {padding-bottom: 1em;}
```

```
     label {
       font-weight: bold;
       float: left;
       width: 20%;
       margin-right: 1em;
       text-align: right;
     }
    -->
  </style>
</head>
<body>
 <h3>Please enter your phone number below.</h3>
 <form action='process_form.php' method='POST'>
 <div id='form'>
<?php
 /* Loop that displays the form fields */
  foreach($labels as $field => $label)
  {
    echo "<div id='field'><label for='$field'>$label</label>
          <input id='$field' name='$field' type='text'
           size='50%' maxlength='65'
           value={$customer[$field]} /> </div>\n";        →52
  }
  echo "</div>\n";
  echo "<input style='margin-left: 33%' type='submit'
          value='$submit' />\n";
?>
</form></body></html>
```


The `form_phone_values_db.inc` file, shown in Listing 1-4, is similar to the `form_phone.inc` file shown in Listing 1-3. The difference is that the file connects to the database to get the name and address and displays them in the form.

The differences are shown in the following lines:

→12 Lines 12–20 create an array that stores the name and phone number retrieved from the database. Line 12 stores a user last name to use to retrieve the information from the database. Line 13 includes a file that contains the username, password, and database name to use when connecting to the database. Lines 14–20 retrieve the name and address and store them in a `$customer` array.

→52 This line adds a `value` attribute to the `<input>` tag to display the values in the form fields.

Figure 1-3 shows the Web page resulting from the script in Listing 1-4. The information in the form is the information stored in the database.

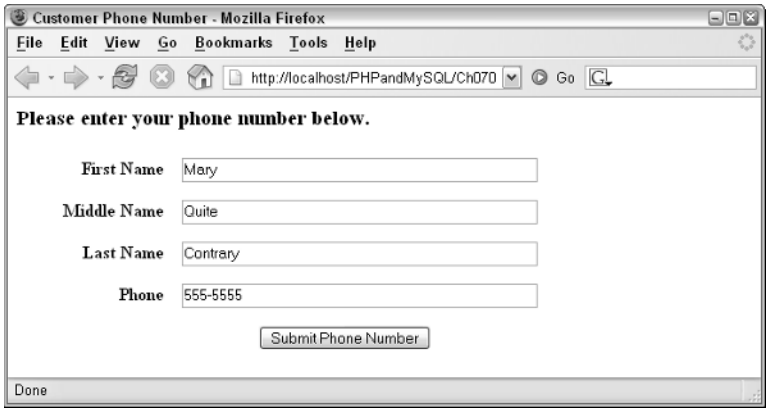

Figure 1-3: A form displaying the username and phone number.

When the user clicks the submit button in the form displayed in Figure 1-3, the `process_form.php` script runs and displays the following:

```
1. first_name = Mary
2. middle_name = Quite
3. last_name = Contrary
4. phone = 555-5555
```

Building selection lists

One type of field that you can use in an HTML form is a *selection list.* Instead of typing into a field, your users select from a list. For instance, in a product catalog, you might provide a list of categories from which users select what they want to view. Or the form for users' addresses might include a list of states that users can select. Or users might enter a date by selecting a month, day, and year from a set of lists.

Use selection lists whenever feasible. When the user selects an item from a list, you can be sure that the item is accurate, with no misspellings, odd characters, or other problems introduced by users' typing errors.

An HTML selection list for a list of categories in a catalog might look as follows:

```
<form action="process_form.php" method="POST">
<select name="category">
  <option value="clothes">clothes</option>
  <option value="furniture" selected>furniture</option>
  <option value="toys">toys</option>
</select>
<input type="submit" value="Select Category" />
</form>;
```

Figure 1-4 shows the selection list that these HTML statements produce. Notice that the Furniture option is selected when the field is first displayed. You determine this default selection by including `selected` in the option tag.

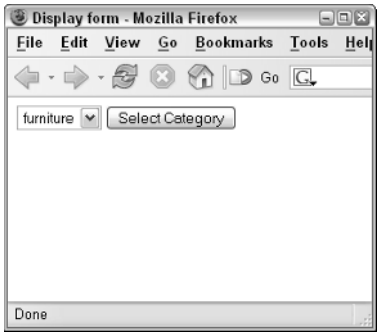

Figure 1-4: A selection field for a catalog.

When the user clicks the arrow on the drop-down list, the entire list drops down, as shown in Figure 1-5, and the user can select any item in the list. Notice that the Furniture option is selected until the user selects a different item.

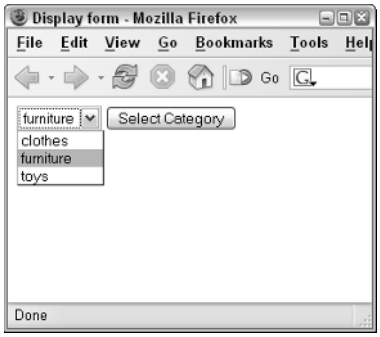

Figure 1-5: A selection field for a catalog with a drop-down list.

When using PHP, your options can be variables. This capability allows you to build dynamic selection lists. For instance, you must maintain the static list of product categories shown in the preceding example. If you add a new category, you must add an `option` tag manually. However, with PHP variables, you can build the list dynamically from the categories in the database. When you add a new category to the database, the new category is automatically added to your selection list without your having to change the PHP script. Listing 1-5 shows code from a file named `form_select.inc` that builds a selection list of categories from the database.

To display this form, you run the script `displayForm.php`, shown in Listing 1-2, with the include statement in the script changed to

```
include("form_select.inc");
```

This includes the file that displays the form produced by the code in Listing 1-5.

Listing 1-5: Building a Selection List

```
<?php
/*  Program name: form_select.inc
 *  Description:  file builds a selection list
 *                from the database.
 */
?>
<html>
<head><title>Categories</title></head>
<body>
<?php
 include("dbstuff.inc");
 $cxn = mysqli_connect($host,$user,$password,$database)
        or die ("couldn't connect to server");
 $query = "SELECT DISTINCT cat FROM Product ORDER BY cat";
 $result = mysqli_query($cxn,$query)
           or die ("Couldn't execute query.");
 /* create form containing selection list */
 echo "<form action='process_form.php' method='POST'
              style='margin-left: 2em'>
       <label for='cat'
              style='font-weight: bold'>Category:</label>
        <select id='cat' name='cat'
                 style='margin-top: 3em'>\n";
 while($row = mysqli_fetch_assoc($result))
 {
    extract($row);
    echo "<option value='$cat'>$cat</option>";
 }
 echo "</select>\n";
 echo "<input type='submit' style='margin-left: 3em'
              value='Select category' />
       </form>\n";
?>
</body></html>
```

Notice the following in the script in Listing 1-5:

- **Using `DISTINCT` in the query:** `DISTINCT` causes the query to get each category only once. Without `DISTINCT`, the query would return each category several times if it appeared several times in the database.

- **Using `ORDER BY` in the query:** The categories are sorted alphabetically.
- **`echo` statement before the loop:** The `<form>` and `<select>` tags are echoed before the `while` loop starts because they are echoed only once.
- **`echo` statement in the loop:** The `<option>` tags are echoed in the loop — one for each category in the database. No item is marked as selected, so the first item in the list is selected automatically.
- **`echo` statements after the loop:** The end `<form>` and `<select>` tags are echoed after the loop because they're echoed only once.

The selection list produced by this script is longer than the selection list shown earlier in Figure 1-5 because there are more categories in the database. The Clothes option is selected in this script because it is the first item in the list — not because it's specifically selected as it is in the HTML tags that produce Figure 1-5. The drop-down list produced by this script is in alphabetical order, as shown in Figure 1-6.

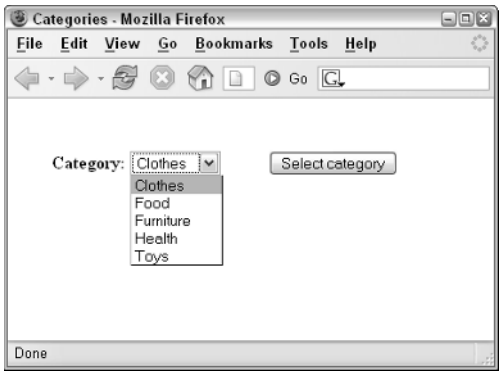

Figure 1-6: A dynamic selection list for a catalog.

You can use PHP variables also to set up which option is selected when the selection box is displayed. For instance, suppose that you want the user to select a date from month, day, and year selection lists. You believe that most people will select today's date, so you want today's date to be selected by default when the box is displayed. Listing 1-6 shows the file `form_date.inc`, which displays a form for selecting a date and selects today's date automatically.

To display this form, you run the script `displayForm.php`, shown in Listing 1-2, with the include statement in the script changed to:

```
include("form_date.inc");
```

This includes the file that displays the form produced by the code in Listing 1-6.

Listing 1-6: Building a Date Selection List

```
<?php
/*  Program name: form_date.inc
 *  Description:  Code displays a selection list that
 *                customers can use to select a date.
 */
 echo "<html>
       <head><title>Select a date</title></head>
       <body>";

 $monthName = array(1 => "January", "February", "March",
                         "April", "May", "June", "July",
                         "August", "September", "October",
                         "November", "December");
 $today = time();                    //stores today's date
 $f_today = date("M-d-Y",$today);    //formats today's date

 echo "<div style = 'text-align: center'>\n";
 echo "<h3>Today is $f_today</h3><hr />\n";
 echo "<form action='process_form.php' method='POST'>\n";

 /* build selection list for the month */
 $todayMO = date("n",$today);  //get the month from $today
 echo "<select name='dateMonth'>\n";
 for ($n=1;$n<=12;$n++)
 {
   echo " <option value=$n";
   if ($todayMO == $n)
   {
     echo " selected";
   }
   echo " > $monthName[$n]\n</option>";
}
 echo "</select>\n";

 /* build selection list for the day */
 $todayDay= date("d",$today);    //get the day from $today
 echo "<select name='dateDay'>\n";
 for ($n=1;$n<=31;$n++)
 {
   echo " <option value=$n";
   if ($todayDay == $n )
   {
     echo " selected";
   }
   echo " > $n</option>\n";
}
```

```
 echo "</select>\n";

 /* build selection list for the year */
 $startYr = date("Y", $today);  //get the year from $today
 echo "<select name='dateYear'>\n";
 for ($n=$startYr;$n<=$startYr+3;$n++)
 {
   echo " <option value=$n";
   if ($startYr == $n )
   {
     echo " selected";
   }
   echo " > $n</option>\n";
}
 echo "</select>\n";
 echo "</form></div>\n";
?>
</body></html>
```

The Web page produced by the script in Listing 1-6 is shown in Figure 1-7. The date appears above the form so that you can see that the selection list shows the correct date. The selection list for the month shows all 12 months when it drops down. The selection list for the day shows 31 days when it drops down. The selection list for year shows 4 years.

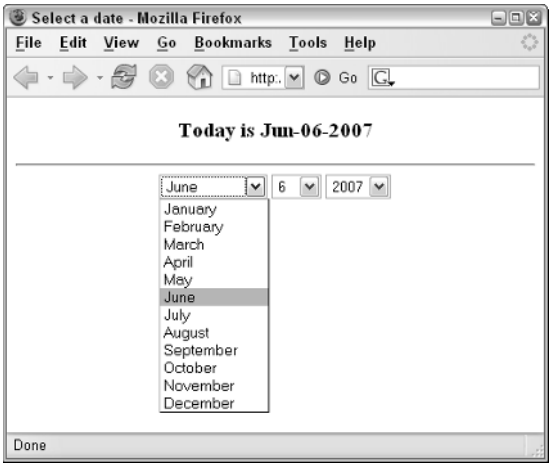

Figure 1-7: A selection field for the date with today's date selected.

The script in Listing 1-6 produces the Web page in Figure 1-7 by following these steps:

1. **It creates an array containing the names of the months.**

The keys for the array are the numbers. The first month, January, starts with the key `1` so that the keys of the array match the numbers of the months.

2. **It creates variables containing the current date.**

   `$today` contains the date in a system format and is used in the form. `$f-today` is a formatted date that is used to display the date in the Web page.

3. **It displays the current date at the top of the Web page.**

4. **It builds the selection field for the month:**

   a. Creates a variable containing today's month.

   b. Echoes the `<select>` tag, which should be echoed only once.

   c. Starts a `for` loop that repeats 12 times.

   d. Inside the loop, echoes the `<option>` tag by using the first value from the `$monthName` array.

   e. If the number of the month being processed is equal to the number of the current month, it adds the word `"selected"` to the `<option>` tag.

   f. Repeats the loop 11 more times.

   g. Echoes the closing `<select>` tag for the selection field, which should be echoed only once.

5. **It builds the selection field for the day.**

   Uses the procedure described in Step 4 for the month. However, only numbers are used for this selection list. The loop repeats 31 times.

6. **It builds the selection field for the year:**

   a. Creates the variable `$startYr`, containing today's year.

   b. Echoes the `<select>` tag, which should be echoed only once.

   c. Starts a `for` loop. The starting value for the loop is `$startYr`. The ending value for the loop is `$startYr+3`.

   d. Inside the loop, echoes the `<option>` tag, using the starting value of the `for` loop, which is today's year.

   e. If the number of the year being processed is equal to the number of the current year, it adds the word `"selected"` to the `<option>` tag.

   f. Repeats the loop until the ending value equals `$startYr+3`.

   g. Echoes the closing `<select>` tag for the selection field, which should be echoed only once.

7. **It echoes the ending tag for the form.**

Building lists of radio buttons

You might want to use radio buttons instead of selection lists. For instance, you can display a list of radio buttons for your catalog categories and have users select the button for the category that they're interested in.

The format for radio buttons in a form is

```
<input type="radio" name="name" value="value" />
```

You can build a dynamic list of radio buttons representing all the categories in your database in the same manner that you build a dynamic selection list in the preceding section. Listing 1-7 shows the file `form_radio.inc`, which displays a list of radio buttons based on categories in the database.

To display this form, you run the script `displayForm.php`, shown in Listing 1-2, with the `include` statement in the script changed to:

```
include("form_radio.inc");
```

This includes the file that displays the form produced by the code in Listing 1-7.

Listing 1-7: Building a List of Radio Buttons

```
<?php
/*  Program name: form_radio.inc
 *  Description:  Program displays a list of radio
 *                buttons from database info.
 */
 echo "<html>
       <head><title>Radio Buttons</title></head>
       <body>";
 include("dbstuff.inc");
 $cxn = mysqli_connect($host,$user,$password,$database)
        or die ("Couldn't connect to server");
 $query = "SELECT DISTINCT cat FROM Product
                  ORDER BY cat";
 $result = mysqli_query($cxn,$query)
           or die ("Couldn't execute query.");

 echo "<div style='margin-left: .5in; margin-top: .5in'>
       <p style='font-weight: bold'>
        Which type of product are you interested in?</p>
       <p>Please choose one category from the
          following list:</p>\n";

 /* create form containing radio buttons */
 echo "<form action='process_form.php' method='POST'>\n";
```

(continued)

Listing 1-7 *(continued)*

```
 while($row = mysqli_fetch_assoc($result))
 {
    extract($row);
    echo "<input type='radio' name='category'
                value='$cat' />$cat\n";
    echo "<br />\n";
}
 echo "<p><input type='submit' value='Select category' />
      </form></div>\n";
?>
</body></html>
```

The Web page produced by this file is shown in Figure 1-8.

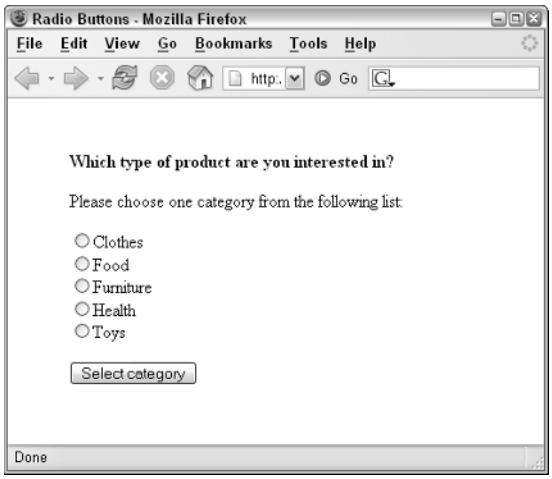

Figure 1-8: List of radio buttons produced by the code in form_radio.inc.

Building lists of check boxes

You might want to use check boxes in your form. Check boxes are different from selection lists and radio buttons because they allow users to select more than one option. For instance, if you display a list of product categories with check boxes, a user can select two or three or more categories. The file `form_checkbox.inc` in Listing 1-8 creates a list of check boxes.

To display this form, you run the script `displayForm.php`, shown in Listing 1-2, with the `include` statement in the script changed to

```
include("form_checkbox.inc");
```

This includes the file that displays the form produced by the code in Listing 1-8.

Listing 1-8: Building a List of Check Boxes

```
<?php
/*  Program name: form_checkbox.inc
 *  Description:  Program displays a list of
 *                checkboxes from database info.
 */
  echo "<html>
        <head><title>Checkboxes</title></head>
        <body style='margin: .5in'>";
  include("dbstuff.inc");
  $cxn = mysqli_connect($host,$user,$passwd,$databname)
         or die ("couldn't connect to server");
  $query = "SELECT DISTINCT cat FROM Product
                   ORDER BY cat";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  echo "<h3>Which products are you interested in?
         <span style='font-size: 80%; font-weight: normal'>
         (Check as many as you want)</span></h3>\n";

  /* create form containing checkboxes */
  echo "<fieldset>
        <legend style='font-weight: bold'>Products</legend>
        <form action='process_form.php' method='POST'>
        <ul style='list-style: none'>\n";
  while($row = mysqli_fetch_assoc($result))
  {
     extract($row);
     echo "<li><input type='checkbox' name='interest[$cat]'
                 id='$cat' value='$cat' />
                 <label for='$cat'
                    style='font-weight: bold'>$cat</label>
           </li>\n";
  }
  echo "</ul></fieldset>";
  echo "<p><input type='submit'
                  value='Select Categories' /></p>
        </form></body></html>\n";
?>
```

Notice that the input field uses an `$interest` array as the name for the field because more than one check box can be selected. This script creates an element in the array with a key/value pair for each check box that's selected. For instance, if the user selects both *furniture* and *toys,* the following array is created:

```
$interest[Furniture]=Furniture
$interest[Toys]=Toys
```

The script that processes the form has the selections available in the `POST` array, as follows:

```
$_POST['interest']['Furniture']
$_POST['interest']['Toys']
```

Figure 1-9 shows the Web page produced by `form_checkbox.inc`.

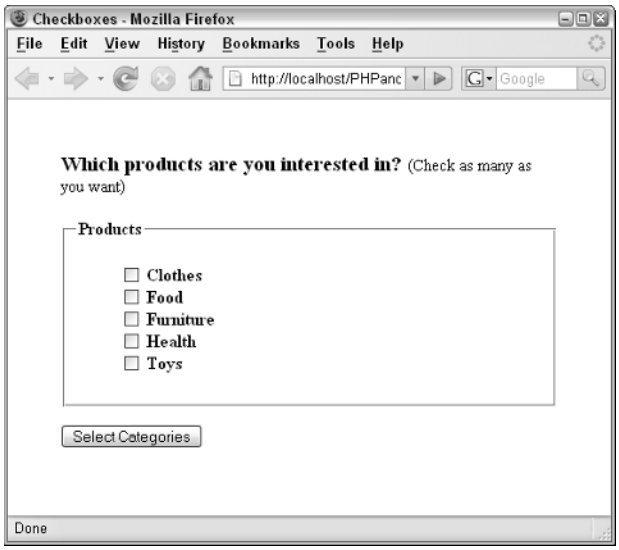

Figure 1-9: A list of check boxes produced by form_checkbox.inc.

Processing Information from the Form

Say that Joe Customer fills in an HTML form, selecting from lists and typing information into text fields. When he clicks the submit button, the script listed in the action attribute of the `<form>` tag, such as `action=process_form.php`, runs and receives the information from the form. Handling form information is one of PHP's best features. You don't need to worry about the form data — just get it from one of the built-in arrays and use it.

The form data is available in the processing script in arrays, such as `$_POST` or `$_GET`. The key for the array element is the name of the input field in the form. For instance, if you echo the following field in your form

```
echo "<input type='text' name='firstName' />";
```

the processing script can use the variable `$_POST[firstName]`, which contains the text that the user typed into the field. The information that the user

selects from drop-down lists or radio buttons is similarly available for use. For instance, if your form includes the following list of radio buttons,

```
echo "<input type='radio' name='pet' value='dog' />dog\n";
echo "<input type='radio' name='pet' value='cat' />cat\n";
```

you can access the variable `$_POST[pet]`, which contains either `dog` or `cat`, depending on what the user selected.

You handle check boxes in a slightly different way because the user can select more than one check box. As shown in Listing 1-8, the data from a list of check boxes can be stored in an array so that all the check boxes are available. For instance, if your form includes the following list of check boxes,


```
echo "<input type='checkbox' name='interest[dog]'
             value='dog' />dog\n";
echo "<input type='checkbox' name='interest[cat]'
             value='cat' />cat\n";
```

you can access the data by using the multidimensional variable `$_POST[interest]`, which contains the following:

```
$_POST[interest][dog] = dog
$_POST[interest][cat] = cat
```

You now have all the information that you wanted in the `$_POST` or `$_GET` array. Well, maybe. Joe Customer might have typed information that contains a typo. Or he might have typed nonsense. Or he might even have typed malicious information that can cause problems for you or other people using your Web site. Before you use Joe's information or store it in your database, you want to check it to make sure that it's the information you asked for. Checking the data is called *validating* the data.

Validating the data includes the following:

- **Checking for empty fields:** You can require users to enter information in a field. If the field is blank, the user is told that the information is required, and the form is displayed again so that the user can type the missing information.
- **Checking the format of the information:** You can check the information to see that it's in the correct format. For instance, *ab3&*xx* clearly is not a valid ZIP code. Checking the format is also important to identify security problems. Security issues are discussed in detail in Book IV.

Checking for empty fields

When you create a form, you can decide which fields are required and which are optional. Your decision is implemented in the PHP script. You check the

fields that require information. If a required field is blank, you send a message to the user, indicating that the field is required, and you then redisplay the form.

The general procedure to check for empty fields is

```
if(empty($_POST['last_name']))
{
    echo "You did not enter your last name.
          Last name is required.<br />\n";
    redisplay the form;
    exit();
}
echo "Welcome to the Members Only club.
      You may select from the menu below.\n";
      display the menu;
```

Notice the `exit` statement, at the end of the `if` statement, that stops the script. Without the `exit` statement, the script would continue to the statements after the `if` statement. In other words, without the `exit` statement, the script would display the form and then continue to echo the welcome statement and the menu.

In many cases, you want to check all the fields in the form. You can do this by looping through the array `$_POST`. The following statements check the array for any empty fields:

```
foreach($_POST as $value)
{
   if($value == "")
   {
     echo "You have not filled in all the fields.\n";
     redisplay the form;
     exit();
   }
}
echo "Welcome";
```

When you redisplay the Web form, make sure that it contains the information that the user already typed. If users have to retype correct information, they're likely to get frustrated and leave your Web site.

In some cases, you might require the user to fill in most but not all fields. For instance, you might request a fax number in the form or provide a field for a middle name, but you don't really mean to restrict registration on your Web site to users with middle names and faxes. In this case, you can make an exception for fields that aren't required, as follows:

```
foreach($_POST as $field => $value)
{
  if($field == "")
  {
     if($field != "fax" and $field != "middle_name")
     {
       echo "You have not filled in $field\n";
       display the form;
       exit();
     }
  }
}
echo "Welcome";
```


Notice that the inner `if` conditional statement is true only if the field is not the fax field and is not the middle name field. For those two fields, the script doesn't display an error message and stop.

As an example, Listings 1-9 and 1-10 display and process the phone form shown earlier in this chapter. Listing 1-9 shows `form_phone_values.inc` that contains the display code for the form. The logic code is shown in Listing 1-10.

The logic code both displays the form and processes the information submitted in the form. Performing both tasks in a single script is more efficient than creating two separate scripts. To process the submitted information in the same script, the action attribute of the `<form>` tag specifies the current file. You can do this by providing the name of the script, or you can specify the current file by using a variable provided by PHP, as follows:

```
<form action="$_SERVER['PHP_SELF']" method='POST'>
```

The element in the `$_SERVER` superglobal array with the key `PHP_SELF` contains the path/filename to the script that is currently running.

The display code in Listing 1-9 displays the same form as the code in Listing 1-2. The display code has been modified slightly. The form includes a hidden field that is used by the logic code in Listing 1-10.

Listing 1-9: Displays a Form with a Hidden Field

```
<?php
/*  Program name: form_phone_values.inc
 *  Description:  Defines a form that collects a user's
 *                name and phone number.
 */
$labels = array( "first_name" => "First Name",
```

(continued)

Listing 1-9 *(continued)*

```
                  "middle_name" => "Middle Name",
                  "last_name" => "Last Name",
                  "phone" => "Phone");
$submit = "Submit Phone Number";
?>
 <html>
 <head><title>Customer Phone Number</title>
    <style type='text/css'>
    <!--
     #form {
       margin: 1.5em 0 0 0;
       padding: 0;
     }
     #field {padding-bottom: 1em;}
     label {
     font-weight: bold;
     float: left;
     width: 20%;
     margin-right: 1em;
     text-align: right;
     }
    -->
   </style>
 </head>
 <body>
<h3>Please enter your phone number below.</h3>
<?php
  echo "<form action='$_SERVER[PHP_SELF]' method='POST'>
  <div id='form'>"
if(isset($message))                                          →36
  {
     echo $message;
  }
  /* Loop that displays the form fields */
  foreach($labels as $field => $label)
  {
    echo "<div id='field'><label for='$field'>$label</label>
          <input id='$field' name='$field' type='text'
             size='50%' maxlength='65'
             value='".@$$field."' /></div>\n";               →46
  }
  echo "<input type='hidden' name='sent' value='yes' />\n";
  echo "<input style='margin-left: 33%' type='submit'
          value='$submit' />\n";
?>
</form></body></html>
```

This code is similar to the code shown in Listing 1-2, with the following differences:

- **→33** The `action` attribute of the `<form>` tag shows `$_SERVER['PHP_SELF']`, which specifies the current script.
- **→36–39** If a variable named message exists, it is displayed. This variable can contains an error message that is created in the logic script (Listing 1-10) if any errors are found.
- **→46** A `value` attribute is added to the `<input>` tag for the fields. The value is a variable variable, `$$field`, which will output a variable with the name of the field, such as `$first_name` or `$middle_name`. Notice the `@` before the variable name to prevent a notice from displaying when the variable does not exist.
- **→48** A hidden field is added to the form. When the user clicks the submit button, the hidden field is sent with the form.

When the `checkBlank.php` script in Listing 1-10 first runs, it displays the form by including the `form_phone_values.inc` file. When the form is submitted, the script checks all the required form fields for blank fields. All the fields are required except `middle_name`. The following is an overview of the script's structure:

```
if (form has been submitted)
   Test whether any required fields are blank.
   if(blanks are found)
      display error message
      redisplay form
   if(no blanks are found)
      display "All required fields contain information"
else (form is displayed for the first time, not submitted)
   display blank form
```

Listing 1-10: Checking for Blank Fields

```
<?php
/*  Program name: checkBlank.php
 *  Description:  Program checks all the form fields for
 *                blank fields.
 */
 if(isset($_POST['sent']) && $_POST['sent'] == "yes")      →6
 {
/* check each field except middle name for blank fields */
  foreach($_POST as $field => $value)                      →9
  {
  if($value == "")                                         →11
    {
```

(continued)

Listing 1-10 *(continued)*

```
      if($field != "middle_name")                                    →13
      {
         $blank_array[] = $field;                                    →15
      } // endif field is not middle name
    } // endif field is blank
    else                                                             →18
    {
      $good_data[$field] = strip_tags(trim($value));
    }
  }  // end of foreach loop for $_POST
  /* if any fields were blank, create error message and
     redisplay form */
  if(@sizeof($blank_array) > 0)                                      →25
  {
    $message = "<p style='color: red; margin-bottom: 0;
         font-weight: bold'>
         You didn't fill in one or more required fields.
         You must enter:
         <ul style='color: red; margin-top: 0;
             list-style: none' >";
    /* display list of missing information */
    foreach($blank_array as $value)
    {
      $message .= "<li>$value</li>";
    }
    $message .= "</ul>";
    /* redisplay form */
    extract($good_data);                                             →40
    include("form_phone_values.inc");                                →41
    exit();                                                          →42
  } // endif blanks
  echo "All required fields contain information";                   →44
 } // endif submitted
 else                                                                →46
 {
  include("form_phone_values.inc");
 }
?>
```

The following numbers in the explanation of the script shown in Listing 1-10 refer to the line numbers in the listing:

→6 Begins an `if` statement that checks for the hidden field. If the hidden field exists, the form was submitted by a user. The `if` statement executes. If the hidden field does not exist, the script is running for the first time, not started by a form submitted by a user, and the script jumps to the `else` statement on Line 46.

→9 Starts a `foreach` statement that loops through the `$_POST` array. All the information from the form is in the array.

→11 Starts an `if` statement that executes if the field is blank.

→13 Starts an `if` statement that executes if the field is not `middle_name`. `middle_name` is not a required field, so it is allowed to be blank. If the blank field is not `middle_name`, the field name is stored in the array called `$blank_array` (Line 15).

→18 Starts an `else` statement that executes if the field is not blank. The data is cleaned and stored in `$good_data` so it can be safely displayed in the form.

→25 Determines whether any blank fields were found by checking whether `$blank_array` contains any elements. If one or more blank fields were found, the script constructs an error message and stores it in `$message`. The form is displayed (Lines 40 and 41). The error message is displayed at the top of the form, and the information from `$good_data` is displayed in the fields. An `exit` statement (Line 42) stops the script after the form displays. The user must click the submit button to continue.

→44 If no blank fields were found, the `if` statement on Line 25 does not execute. The `echo` statement on Line 44 displays a message that all fields are okay.

→46 Begins an `else` statement that executes the first time the script is run, before the user submits the form. The blank form is displayed.

Don't forget the `exit` statement. Without the `exit` statement, the script would continue and would display `All required fields contain information` after displaying the form.

Figure 1-10 shows the Web page that results if the user didn't enter a first or a middle name. Notice that the list of missing information doesn't include Middle Name because Middle Name isn't required. Also, notice that the information the user originally typed into the form is still displayed in the form fields.

Checking the format of the information

Whenever users must type information in a form, you can expect a certain number of typos. You can detect some of these errors when the form is submitted, point out the error(s) to the user, and then request that he or she retype the information. For instance, if the user types **8899776** in the ZIP code field, you know this isn't correct. This information is too long to be a ZIP code and too short to be a ZIP+4 code.

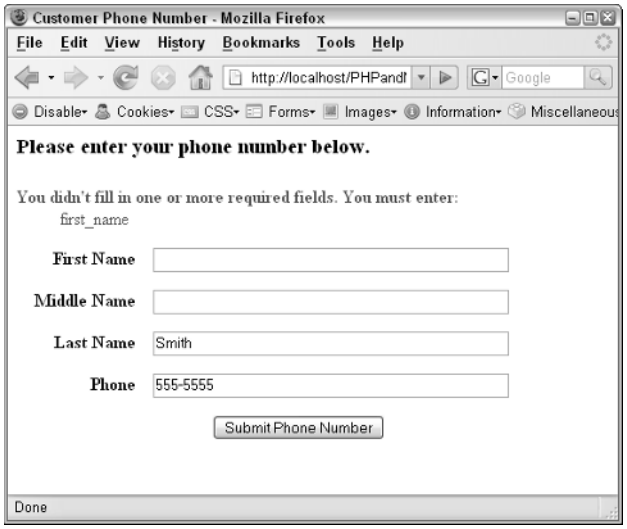

Figure 1-10: The result of processing a form with missing information.

You also need to protect yourself from malicious users — users who might want to damage your Web site or your database or steal information from you or your users. You don't want users to enter HTML tags into a form field — something that might have unexpected results when sent to a browser. A particularly dangerous tag would be a script tag that allows a user to enter a script into a form field. Checking data, as described in this section, protects against both accidental problems and malicious users.

If you check each field for its expected format, you can catch typos and prevent most malicious content. Checking information is a balancing act. You want to catch as much incorrect data as possible, but you don't want to block any legitimate information. For instance, when you check a phone number, you might limit it to numbers. The problem with this check is that it would screen out legitimate phone numbers in the form 555-5555 or (888) 555-5555, so you also need to allow hyphens (-), parentheses ( ), and spaces. You might limit the field to a length of 14 characters, including parentheses, spaces, and hyphens, but this screens out overseas numbers or numbers that include an extension.

The bottom line: You need to think carefully about what information you want to accept or screen out for any field.

You can check field information by using *regular expressions,* which are patterns. You compare the information in the field against the pattern to see whether it matches. If it doesn't match, the information in the field is incorrect, and the user must type it over. (See Book II, Chapter 2 for more on regular expressions.)

In general, use these statements to check fields:

```
if( !preg_match("pattern",$variablename) )
{
    echo error message;
    redisplay form;
    exit();
}
echo "Welcome";
```

The `preg_match` function matches the pattern to the input value in `$variablename`, where pattern is a regular expression. (See Book II, Chapter 2 for information on matching strings to regular expressions.) Notice that the condition in the `if` statement is negative. That is, the `!` (exclamation mark) means `"not"`. So, the `if` statement actually says this: If the value in the variable does *not* match the pattern, execute the `if` block.

For example, suppose that you want to check an input field that contains the user's last name. You can expect names to contain letters, not numbers, and possibly apostrophe and hyphen characters (as in *O'Hara* and *Smith-Jones*) and also spaces (as in *Van Dyke*). Also, it's difficult to imagine a name longer than 50 characters. Thus, you can use the following statements to check a name:

```
if( !preg_match("/[A-Za-z' -]{1,50}/",$last_name)
{
    echo error message;
    redisplay form;
    exit();
}
echo "Welcome";
```

If you want to list a hyphen (-) as part of a set of allowable characters that are surrounded by square brackets (`[ ]`), you must list the hyphen at the beginning or at the end of the list. Otherwise, if you put it between two characters, the script will interpret it as the range between the two characters, such as *A–Z*.

You also need to check multiple-choice fields. Although multiple choice prevents honest users from entering mistakes, it doesn't prevent clever users with malicious intentions from entering unexpected data into the fields. You can check multiple-choice fields for acceptable output with the following type of `regex`:

```
if( !preg_match("/(male|female)/",$gender) )
```

If the field contains anything except the value `male` or the value `female`, the `if` block executes.

The script `checkFormat.php`, shown in Listing 1-11, checks field information for valid formats. The script displays and processes the form in the file `form_phone_values.inc`, shown in Listing 1-9. This script checks only for valid format, not for empty required fields. In most cases, you would want to check for both blank fields, as shown in the script `checkBlanks.php` in Listing 1-10 and for valid formats.

Listing 1-11: Checking for Invalid Formats in Form Fields

```
<?php
/*  Program name: checkFormat.php
 *  Description:  Program checks all the form fields for
 *                valid formats.
 */
 if(isset($_POST['sent']) && $_POST['sent'] == "yes")      →6
{
 /* validate data from the form */
  foreach($_POST as $field => $value)                      →9
  {
    if(!empty($value))                                     →11
    {
      $name_patt = "/^[A-Za-z' -]{1,50}$/";                →13
      $phone_patt = "/^[0-9)(xX -]{7,20}$/";
      if(preg_match("/name/i",$field))                     →15
      {
         if(!preg_match($name_patt,$value))                →17
         {
           $error_array[] = "$value is not a valid name";
           $bad_data[$field] = strip_tags(trim($value));
         }
         else                                              →22
         {
           $good_data[$field] = strip_tags(trim($value));
         }
      }  // endif name format check
      if(preg_match("/phone/i",$field))                    →27
      {
         if(!preg_match($phone_patt,$value))               →29
         {
           $error_array[] = "$value is not a
                              valid phone number";
           $bad_data[$field] = strip_tags(trim($value));
         }
         else
         {
           $good_data[$field] = strip_tags(trim($value));
         }
      }  // endif phone format check
    } // endif not blank
  }  // end of foreach loop for $_POST
  /* if any fields were invalid, create error message and
```

```
    redisplay form */
  if(@sizeof($error_array) > 0) // errors are found          →44
  {
   /* create error message */
    $message = "<ul style='color: red; list-style: none' >";
    foreach($error_array as $value)
    {
      $message .= "<li>$value</li>";
    }
    $message .= "</ul>";
   /* redisplay form */
    extract($good_data);                                     →54
    extract($bad_data);
    include("form_phone_values.inc");
    exit();                                                  →57
  } // end if blanks
  echo "All required fields contain valid information";      →59
} // end if submitted
else                                                         →61
{
  include("form_phone_values.inc");
}
?>
```


The numbers in the following explanation of Listing 1-11 refer to the line numbers in the listing:

→6 Begins an `if` statement that checks for the hidden field. If the hidden field exists, the form was submitted by a user. The `if` statement executes. If the hidden field doesn't exist, the script is running for the first time, not started by a form submitted by a user, and the script jumps to the `else` statement on Line 61.

→9 Starts a `foreach` statement that loops through the `$_POST` array. All the information from the form is in the array.

→11 Starts an `if` statement that executes if the field is not blank. This is necessary so that fields that are allowed to be blank aren't tested for an invalid format.

→13 Creates variables that contain the pattern that matches the correct format. `$name_patt` contains the pattern for name variables to match; `$phone_patt` contains the pattern for phone numbers to match.

→15 Starts an `if` statement that executes for name fields. Notice that the pattern to match is followed by an `i`, which means to ignore case.

→17 Starts an `if` statement that tests whether the data in the name field matches the pattern. If the data doesn't match the pattern, an error message is added to an array named `$error_array`, and the data is cleaned and added to an array called `$bad_data`. If

the data does match the pattern, the `else` statement (Line 22) is executed, which adds the data to an array named `$good_data`.

→27 Starts an `if` statement that executes for the field that contains the phone number.

→29 Starts an `if` statement that tests whether the data in the phone field matches the pattern. If the data does not match the pattern, an error message is added to an array named `$error_array` and the data is cleaned and added to an array called `$bad_data`. If the data does match the pattern, the else statement is executed, which adds the data to an array named `$good_data`.

→44 Determines whether any invalid formats were found by checking whether `$error_array` contains any elements. If one or more invalid formats were found, the script constructs an error message and stores it in `$message`. The form is displayed (Lines 54–56). The error message is displayed at the top of the form and the information from `$good_data` and `$bad_data` is displayed in the fields. An `exit` statement (Line 57) stops the script after the form displays. The user must click the submit button to continue.

→59 If no invalid formats were found, the `if` statement on Line 44 doesn't execute. The `echo` statement on Line 59 displays a message that all fields contain valid formats.

→61 Begins an `else` statement that executes the first time the script is run, before the user submits the form. The blank form is displayed.

The Web page in Figure 1-11 results when the user accidentally types nonsense for his or her phone number.

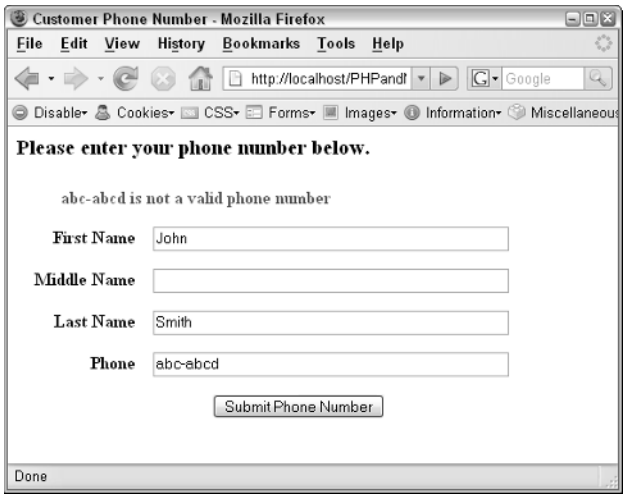

Figure 1-11: The result of processing a form with incorrect information.

Giving users a choice with multiple submit buttons

You can use more than one submit button in a form. For instance, in a customer order form, you might use a button that reads *Submit Order* and another button that reads *Cancel Order.* The logic code in your script can check the value of the submit button and process the form differently, depending on which button the user clicks.

Listing 1-12 shows the display code for a form with two buttons. The script containing the logic code that displays and processes the form is shown in Listing 1-13.

Listing 1-12: Displaying a Form with Two Submit Buttons

```
<?php
/*  Program name: form_two.inc
 *  Description:  Displays a form with two buttons.
 */
?>
<html>
<head><title>Two Buttons</title></head>
<body>
<?php
echo "<form action='$_SERVER[PHP_SELF]' method='POST'>
        <p><label for='last_name'
              style='font-weight: bold'>Name: </label></p>
        <p><input id='last_name' name='last_name' type='text'
             size='50%' maxlength='65' /></p>
        <p><input type='submit' name='display_button'
                value='Show Address' />
        <input type='submit' name='display_button'
                value='Show Phone Number' /></p>
        <input type='hidden' name='sent' value='yes' />
      </form>";
?>
</body></html>
```

Notice that the submit button fields have a name: `display_button`. The fields each have a different value. Whichever button the user clicks sets the value for `$display_button`. The form produced by this file is shown in Figure 1-12.

The script `processTwoButtons.php` in Listing 1-13 processes the form in Listing 1-12. The script displays the form shown in Figure 1-12. When the user clicks a button, the script performs different actions, depending on which button the user clicks.

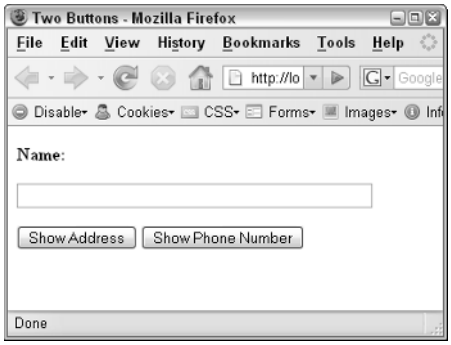

Figure 1-12: The form with two submit buttons.

Listing 1-13: Processing Two Submit Buttons

```
<?php
/*  Program name: processTwoButtons.php
 *  Description:  Displays different information depending
 *                on which submit button was clicked.
 */
 if(isset($_POST['sent']) && $_POST['sent'] == "yes")
{
   include("dbstuff.inc");
   $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("Couldn't connect to server");
   if($_POST['display_button'] == "Show Address")
   {
     $query = "SELECT street,city,state,zip FROM Customer
                 WHERE last_name='$_POST[last_name]'";
     $result = mysqli_query($cxn,$query)
               or die ("Couldn't execute query.");
     $row = mysqli_fetch_assoc($result);
     extract($row);
     echo "$street<br />$city, $state  $zip";
  }
  else
  {
     $query = "SELECT phone FROM Customer
                   WHERE last_name='$_POST[last_name]'";
     $result = mysqli_query($cxn,$query)
               or die ("Couldn't execute query.");
     $row = mysqli_fetch_assoc($result);
     echo "Phone: {$row['phone']}";
  }
 }
 else
 {
   include("form_two.inc");
 }
?>
```

The script executes different statements, depending on which button the user clicks. If the user clicks the button for the address, the script outputs the address for the name submitted in the form; if the user clicks the Show Phone Number button, the script outputs the phone number.

Creating a Form That Allows Customers to Upload a File

Sometimes you want to receive an entire file of information from a user, such as user résumés for your job-search Web site or pictures for your photo album Web site. Or, suppose you're building the catalog from information supplied by the Sales department. In addition to descriptive text about the product, you want Sales to provide a picture of the product. You can supply a form that Sales can use to upload an image file.

Using a form to upload the file

You can display a form that allows a user to upload a file by using an HTML form designed for that purpose. The general format of the form is as follows:

```
<form enctype="multipart/form-data"
          action="processfile.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
  <input type="file" name="user_file" />
  <input type="submit" value="Upload File" />
</form>
```

Notice the following points regarding the form:

- **The `enctype` attribute is used in the `<form>` tag.** You must set this attribute to `multipart/form-data` when uploading a file to ensure that the file arrives correctly.
- **A hidden field is included that sends a value (in bytes) for `MAX_FILE_SIZE`.** If the user tries to upload a file that is larger than this value, it won't upload. You can set this value as high as 2MB. If you need to upload a file larger than 2MB, you must change the default setting for `upload_max_filesize` in `php.ini` to a larger number before sending a value larger than 2MB for `MAX_FILE_SIZE` in the hidden field.
- **The input field that uploads the file is of type `file`.** Notice that the field has a name — `user_file` — as do other types of fields in a form. The filename that the user enters into the form is sent to the processing script and is available in the built-in array called `FILES`. We explain the structure and information in `FILES` in the following section.

When the user submits the form, the file is uploaded to a temporary location. The script needs to copy the file to another location because the temporary file is deleted as soon as the script is finished.

Processing the uploaded file

Information about the uploaded file is stored in the PHP built-in array called `$_FILES`. An array of information is available for each file that was uploaded, resulting in `$_FILES` being a multidimensional array. As with any other form, you can obtain the information from the array by using the name of the field. The following is the array available from `$_FILES` for each uploaded file:

```
$_FILES['fieldname']['name']
$_FILES['fieldname']['type']
$_FILES['fieldname']['tmp_name']
$_FILES['fieldname']['size']
```

For example, suppose that you use the following field to upload a file, as shown in the preceding section:

```
<input type="file" name="user_file" />
```

If the user uploads a file named `test.txt` in the form, the resulting array that can be used by the processing script looks something like this:

```
$_FILES[user_file][name] = test.txt
$_FILES[user_file][type] = text/plain
$_FILES[user_file][tmp_name] = D:\WINNT\php92C.tmp
$_FILES[user_file][size] = 435
```

In this array, `name` is the name of the file that was uploaded, `type` is the type of file, `tmp_name` is the path/filename of the temporary file, and `435` is the size of the file. Notice that `name` contains only the filename, but `tmp_name` includes the path to the file as well as the filename.

If the file is too large to upload, the `tmp_name` in the array is set to `none`, and the size is set to `0`. The processing script must move the uploaded file from the temporary location to a permanent location. The general format of the statement that moves the file is as follows:

```
move_uploaded_file(path/tempfilename,path/permfilename);
```

The *`path/tempfilename`* is available in the built-in array element `$_FILES['fieldname']['tmp_file']`. The *`path/permfilename`* is the path to the file where you want to store the file. The following statement

moves the file uploaded in the input field, given the name `user_file`, shown earlier in this section:

```
move_uploaded_file($_FILES['user_file']['tmp_name'],
   'c:\data\new_file.txt');
```

The destination directory (in this case, `c:\data`) must exist before the file can be moved to it. This statement doesn't create the destination directory.

Security can be an issue when uploading files. Allowing strangers to load files onto your computer is risky; malicious files are possible. You want to check the files for as many factors as possible after they're uploaded, using conditional statements to check file characteristics, such as expected file type and size. In some cases, for even more security, it might be a good idea to change the name of the file to something else so that users don't know where their files are or what they're called.

Putting it all together

A script that allows a user to upload a file is provided in this section. The script displays a form for the user to upload a file, saves the uploaded file, and then displays a message after the file has been successfully uploaded.

The form that the user uses to upload the file is stored in the file `form_upload.inc`, shown in Listing 1-14.

Listing 1-14: A File That Displays the File Upload Form

```
<!-- Program Name: form_upload.inc
     Description:  Displays a form to upload a file -->
<html>
<head><title>File Upload</title></head>
<body>
<ol><li>Enter the file name of the product picture you
        want to upload or use the browse button
        to navigate to the picture file.</li>
    <li>When the path to the picture file shows in the
        text field, click the Upload Picture button.</li>
</ol>
<div style='text-align: center'><hr />
<form enctype="multipart/form-data"
        action="<?php echo $_SERVER['PHP_SELF'] ?>"
        method="POST">
  <p><input type="hidden" name="MAX_FILE_SIZE"
        value="500000" />
```

(continued)

Listing 1-14 *(continued)*

```
  <input type="file" name="pix" size="60" /></p>
  <p><input type="submit" name="Upload"
        value="Upload Picture" /></p>
</form></div></body></html>
```

The file includes the `enctype` attribute in the `<form>` tag and a hidden field that sets `MAX_FILE_SIZE` to `500,000`. A Web page displaying the form is shown in Figure 1-13.

Figure 1-13: The file-uploading form produced by the code in Listing 1-14.

The form produced by the code in Listing 1-14 allows users to select a file to upload. The form has a text field for inputting a filename and a Browse button that enables the user to navigate to the file and select it.

The PHP script that displays the form and processes the uploaded file is shown in Listing 1-15. This script expects the uploaded file to be an image file and tests to make sure that it is an image file, but any type of file can be uploaded.

Listing 1-15: Uploading a File with a POST Form

```
<?php
 /* Script name: fileUpload.php
  * Description: Uploads a file via HTTP with a POST form.
  */
  if(!isset($_POST['Upload']))                                    →5
  {
    include("form_upload.inc");
  }
  else                                                            →9
  {
    if($_FILES['pix']['tmp_name'] == "none")                      →11
    {
```

```
      echo "<p style='font-weight: bold'>
        File did not successfully upload. Check the
            file size. File must be less than 500K.</p>";
      include("form_upload.inc");
      exit();
    }
    if(!preg_match("/image/",$_FILES['pix']['type']))     →19
    {
      echo "<p style='font-weight: bold'>
        File is not a picture. Please try another
            file.</p>";
      include("form_upload.inc");
      exit();
    }
    else                                                  →27
    {
      $destination='c:\data'."\\".$_FILES['pix']['name'];
      $temp_file = $_FILES['pix']['tmp_name'];
      move_uploaded_file($temp_file,$destination);
      echo "<p style='font-weight: bold'>
        The file has successfully uploaded:
            {$_FILES['pix']['name']}
            ({$_FILES['pix']['size']})</p>";
    }
  }
?>
```


The following discussion of the script refers to the line numbers in the listing:

- **→5** This line is an `if` statement that tests whether the form has been submitted. If not, the form is displayed by including the file containing the form code. The `include` file is shown in Listing 1-14.
- **→9** This line starts an `else` block that executes if the form has been submitted. This block contains the rest of the script and processes the submitted form and uploaded file.
- **→11** This line begins an `if` statement that tests whether the file was successfully uploaded. If not, an error message is displayed, and the form is redisplayed.
- **→19** This line is an `if` statement that tests whether the file is a picture. If not, an error message is displayed, and the form is redisplayed.
- **→27** This line starts an `else` block that executes if the file has been successfully uploaded. The file is moved to its permanent destination, and a message is displayed that the file has been uploaded.

When the file is successfully uploaded and stored in its permanent location, the message `The file has successfully uploaded:` is displayed, followed by the filename and the file size.