SYBEX Sample Chapter

Creating Interactive Websites with PHP and Web Services

Eric Rosebrock

Chapter 4: Building a Website Template with PHP

ISBN: 0-7821-4279-6

Sybex Inc.
1151 Marina Village Parkway
Alameda, CA 94501
U.S.A.
Phone: 510-523-8233
www.sybex.com

CHAPTER 4

Building a Website Template with PHP

Building a website template with PHP has to be one of the most fascinating elements of synchronizing the look and feel of your website. By creating one file that includes the Hypertext Markup Language (HTML) layouts of your site, you can dramatically reduce the time and effort needed to manage changes in your site's layout.

I have been using this method for the past few years to develop all of my websites. It is much easier to work with one file and then include the layout sections I need by using custom functions; that way, I can develop the PHP code for each portion of my website without looking at the HTML. Furthermore, you can change the look and feel of your entire website by merely changing the one template file.

In this chapter, I will show you the methods I have used to successfully design a website template with PHP. I will also share one of the greatest secrets to getting ranked high in the search engines and, at the same time, introduce you to PHP classes and Object-Oriented Programming (OOP).

Designing Your Layout

In this chapter, you'll design a simple table-based layout in HTML for our examples. Alternatively, you may use many different types of software that allow you to easily design your website. Software such as Adobe Photoshop and Adobe Image Ready allow you to create a graphical layout and then "slice" the image file into separate images for an HTML layout. Using the Image Ready software, for example, you can export an HTML file with all of your slices in place. This allows you to get true What-You-See-Is-What-You-Get (WYSIWYG) results when designing in applications such as Photoshop.

Let's begin the initial layout based on the concepts from Chapter 2, "Planning Your Project." To keep this as simple as possible, let's design a basic table-based layout. After you work through this chapter, you should have no problem designing a more advanced graphical-based layout using tables and cells.

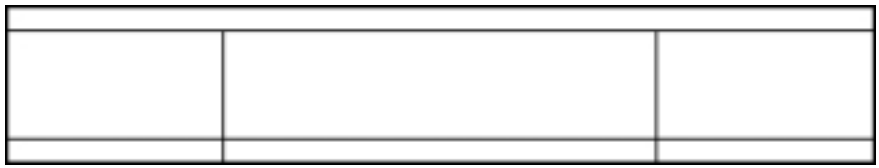
For my initial design, I used Adobe Photoshop to create a simple logo for the site (see Figure 4.1). Although this may not be important now, I will use this example logo as a reference point for the HTML layout. You should create a similar logo that matches your site.

FIGURE 4.1: Adobe Photoshop logo design

Creating the HTML

Let's move on to the HTML design. Using your favorite HTML editor, create a table with a 100-percent width, three table rows, and three columns in the second and third rows. Set the cell padding to 0, the cell spacing to 0, and the border to 0. Figure 4.2 shows the table (I colored the borders for the table so that you can see exactly what I am talking about).

FIGURE 4.2:
Table example



The code in Listing 4.1 is used for the table layout page. Do not worry about any of the head section information; you will get to that later:

Listing 4.1 HTML Layout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<table width="100%" border="2" cellpadding="0"
 cellspacing="0" bordercolor="#000000">
  <tr>
    <td colspan="3"> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
</table>
</body>
</html>
```

Now that you have the layout, if you want, you can insert your logo into the page in the first cell. I inserted my logo so that I have something to reference during this chapter; you can insert the logo you created. The following is an important tip: When building a layout like this, you always want to use hyperlinks and image links that relate to the site root or the document root of your website. For example, if you were to simply type `images/logo.gif` as the image source path, you are not telling the web browser to go to the root of your website and begin the path from that point. However, if you were to type `/images/logo.gif`, the leading forward slash would direct your web browser to the beginning of your document root and start looking for the image from that point. This is important when building a template for your website. Always try to reference the images and links with the leading forward slash!

You have your layout started now. Let's save this file and name it `layout.php` in your website's document root. Inside this new PHP document, you will start out by creating custom functions around the header and footer portions of the layout script. To better understand what you are about to do, look at Figures 4.3 through 4.5. These figures depict how you separate the HTML table into parts with the custom functions you are about to use. The shaded areas are the areas in use for the function. Specifically, Figure 4.3 shows the use of a custom header function, Figure 4.4 shows the use of a custom footer function, and Figure 4.5 shows the use of a content area.

FIGURE 4.3:
HTML usage for custom header function

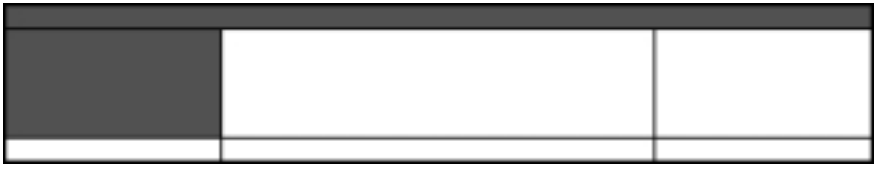


FIGURE 4.4:
HTML usage for custom footer function

FIGURE 4.5:
HTML usage for content area

Creating the PHP Code

Now that you have an understanding of how this HTML layout is going to be divided so you can sandwich your content between the header and footer, let's take a look at the PHP code you are going to use to create the functions required to make this happen. See Listing 4.2.

Listing 4.2 ***layout.php* File**

```
<?php
function myheader($ptitle){
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
</head>

<body>
<table width="100%" border="2" cellpadding="0"
cellspacing="0" bordercolor="#000000">
  <tr>
    <td colspan="3"> </td>
  </tr>
  <tr>
    <td> </td>
    <td>
    <!-- End Header and Begin Content -->
<?php
} // close myheader()
function footer(){
?>
<!-- End Content and Begin Footer -->
    </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
</table>
</body>
</html>
}  //close footer()
?>
```

Now that you have seen the code, let's break it down for further understanding. First, you are going to start the PHP engine by initiating a `<?php` open tag. Next, you are going to create your own function by naming it `myheader` and giving the argument that you are going to pass into the function. You are going to use the variable name `$ptitle`, which stands for *page*

title. You will understand why you use this argument when you get to the "Creating the META Content Class" section later in this chapter. Here are the first few lines of code:

```
<?php
function myheader($ptitle){
?>
```

As you can see, after you name the function and define the arguments, you use a close tag for PHP. Next, you insert the HTML that you want to be above the main content area of the website. At this point, you can also insert the image logo into the first cell of the first table row. Here is the HTML code for the `myheader` function:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
</head>

<body>
<table width="100%" border="2" cellpadding="0"
  cellspacing="0" bordercolor="#000000">
  <tr>
    <td colspan="3"><img src="/images/logo.jpg" ALT="My PHP Site"></td>
  </tr>
  <tr>
    <td> </td>
    <td>
    <!-- End Header and Begin Content -->
```

Next, you issue another open tag that allows you to end the custom `myheader` function and start the footer function that is going to contain the footer HTML:

```
<?php
}
function footer(){
?>
```

You may have noticed that I put HTML comments in my code such as `<!-- End Header and Begin Content -->`. This allows me to debug my scripts more easily when using the source view of my web browser. If you create hidden comments such as the examples in my code, you will be able to determine where your header and footer begin much more easily. Here is the rest of the HTML code for the footer function:

```
<!-- End Content and Begin Footer -->
    </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
```

```
    <td> </td>
    <td> </td>
  </tr>
</table>
</body>
</html>
```

Now that you have the HTML code inserted, let's close out the footer function by issuing a PHP close tag and complete this file for now:

```
<?php
} // close myheader()
?>
```

Using the *layout.php* File

With your layout file coded into PHP, you are ready to put it to use for the first time. Create a text file, call it `index.php`, and save it in the same directory, preferably in your document root, along with the `layout.php` file. Here are the contents of the `index.php` file:

```
<?php
// include the layout file
include $_SERVER['DOCUMENT_ROOT'].'/layout.php';

// Use the myheader function from layout.php
myheader("Welcome to My Website!");

// Enter some content such as this message:
echo "Welcome to My Website!";

// Use the footer function from layout.php
footer();
?>
```

On the first line after the open tag, I used a comment in PHP. You can start a comment with the two forward slashes for a single-line comment such as `// include the layout file`. On the next line of code, you include the `layout.php` file using the PHP `include` function. Now that the file is included, you can use the functions available in the included file. In the next example, you will see how to call the `myheader` function that is located inside the `layout.php` from within the `index.php` file:

```
// Use the myheader function from layout.php
myheader("Welcome to My Website!");
```

Notice how you call the function and pass an argument into the function. In this example, I have included the page title in place of the `$ptitle` variable I used when creating the function in the `layout.php` file. You will use this argument later in this chapter in the "Creating the META Content Class" section.

Your next objective is to fill in the middle of the web page with your content. In the following code example, you use a simple `echo` statement to put the words *Welcome to My Website* between the custom header and footer functions:

```
// Enter some content such as this message:
echo "Welcome to My Website!";
```

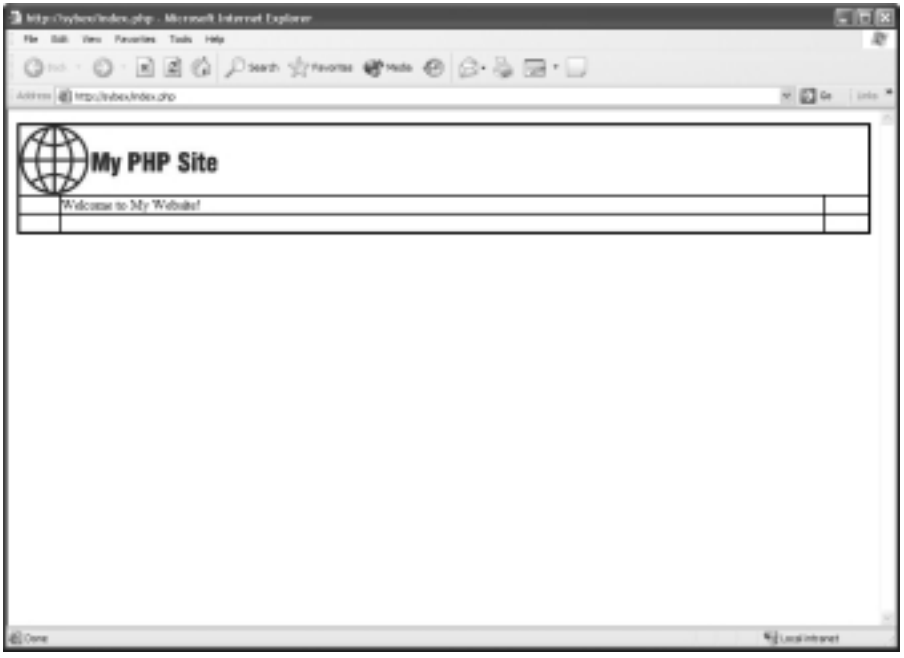
This example is as simple as it can be. Keep in mind that you can include HTML files or dynamic content to fill this section, but for now, I am keeping this simple for this demonstration.

Your final objective in this script is to display the footer portion of your `layout.php` file. So, you will call the footer function from the `layout.php` file into the current file. You achieve this much like the example when you called your `myheader` function. You will also issue the PHP close tag to end the script:

```
// Use the footer function from layout.php
footer();
?>
```

When you open this script in your web browser, using a URL such as `http://yourphpsite.com/index.php`, you should see something similar to Figure 4.6.

FIGURE 4.6: Viewing the `index.php` script



In Figure 4.6, you will see the borders around in the table. In the final `layout.php` file, you will remove the borders because they are not going to be used in the real layout. For Figure 4.6, I left the borders turned on to show you a clear picture of the results for this example.

You have created a basic HTML layout and chopped it up for PHP to use as custom header and footer functions with which to surround your content. With this layout, you can always include files in the left and right columns that allow you to add more links or whatever you desire.

Introducing Classes

As you are developing websites, you will find that you often need to use a piece of code that you have used before. Fortunately, the developers of PHP have taken into consideration that developing the same code repeatedly is a waste of time; as such, they have created a method of using pieces of code with variables to alter the way the code is used. This is called a *class*.

A class is a collection of functions and variables associated within a group. When you begin developing with PHP, you may discover Object Oriented Programming (OOP). OOP allows you to create reusable code and use it whenever and wherever you desire. A good example of OOP is to build an HTML META Content class that you can use throughout any of your websites. You can store this file on your hard drive and use it in as many projects as you like, and it will still serve the same purpose with minimal changes.

In the following sections, you will create the META Content class and include it into your `layout.php` file. Furthermore, by using the arguments you passed through the `myheader` function, you will dynamically generate unique page titles, keywords, and descriptions for your META Content for every page on your website. This is one of my most secretly kept techniques for building web pages throughout my PHP career, and it has proven to work well with search engines. Let's get started with the OOP by designing the first class.

Creating the Basic Class Structure

Based on my experience, I try to create classes inside their own file, and then I name the file with the prefix `cls`*XXXX* where *XXXX* is the name of the class. This is not required, but it will definitely help you figure out what file performs certain actions by just viewing the filename. The following code is a standard class structure:

```
<?php
class ClassName{
   var $var1;
   var $var2;

   function myfunction1(){
```

```
        // Import the vars into this function
        $result = "$this->var1 and $this->var2";

        // Return the result
        return $result;
    }

    function myfunction2(){
        // Reference myfunction1 within this function
        $myfunction1 = $this->myfunction1();

        // Return result
        return $myfunction1;
    }

}
?>
```

In this code, you see the basic class structure. First, you use your open tag for PHP, initiate a new class, and then give the class a name. The `var` in this code simply references a variable that has been defined after the class has been initialized. Next, you will see the first function that is used within this class. Within the first function, it assigns the variables defined into this function using the `$this->` operator.

The previous example also depicts how you would call functions or variables from within the class. The `$this->` operator is best thought of as meaning "within *this* class." I will give you a more practical example when you design the META Content class.

The following example shows you how to include the class file from the previous example in your current PHP script, initialize it, and put this class and its functions to use:

```
<?php
include $_SERVER['DOCUMENT_ROOT']."/classes/clsMyClass.php";

$myclass = &new ClassName;

$myclass->var1 = "Apples";
$myclass->var2 = "Oranges";

echo $myclass->myfunction1()."<br />";
echo $myclass->myfunction2()."<br />";
?>
```

First, you start PHP with the open tag, and then you include the class filename. Notice how you use `$_SERVER['DOCUMENT_ROOT']` and then the full path of the file when you use an `include` function. This ensures that the reference path to the file will be accurate every time. This is my preference, and it never fails—unless I type something wrong or the file does not exist.

After you have the file included, you call the class by using `$myclass = &new ClassName;`. The `$myclass` is simply a reference name to the class, and the `&new` is an operator that calls the class object by the class name. From this point forward, any time you want to use a function or an object from within that class, you will call it by using `$myclass->` followed by the function name.

The output from the previous example would be as follows:

```
Apples and Oranges
Apples and Oranges
```

The first *Apples and Oranges* text was generated by the `myfunction1`, and the second *Apples and Oranges* was generated from `myfunction1` from within `myfunction2`.

Now that you have been introduced to a basic class structure, let's create a more practical class for everyday use, the META Content class.

Creating the META Content Class

It is now time to create your first OOP class. This class will generate all of the META Content from an argument passed to the function. You will get this argument from the `$ptitle` in the `myheader` function from your `layout.php`. From the `$ptitle` string, you will break apart each word into a keyword, fill in the META data, and then generate the META Content description, copyright information, and page titles.

Create a new text file, name this file `clsMetaContent.php`, and save it in a separate directory of your choice in your web server's document root. I prefer to put mine in a directory called `classes` for a more organized web structure. The example in Listing 4.3 is the complete META Content class. I will break this class down into small portions and explain it one step at a time:

Listing 4.3 META Content Class

```
<?php
class Meta{

   function metadata($ptitle){

      // Formulate the description for each page.
      if(empty($ptitle)){
         $description = $this->description;
      } else {
         $description = "$ptitle - $this->description";
      }

      // Make the keywords of the title lower case
      $keywords = strtolower($ptitle);
```

```
        // Replace double spaces with single spaces
        $keywords = str_replace("  ", " ", $keywords);

        // Make string comma seperated
        $meta_words = str_replace(" ", ", ", $keywords);

        // If no Page Title, Use Alternative
        if(!$ptitle){
           $meta .= "<TITLE>$this->sitename - ".
           $meta .= "$this->slogan</TITLE>\n";
        } else {
           $meta .= "<TITLE>$this->sitename: ".
           $meta .= "$ptitle</TITLE>\n";
        }

        // Append META content to the $meta string for output
        $meta .= "<META NAME=\"KEYWORDS\"
         CONTENT=\"$meta_words, $this->keywords2\">\n";
        $meta .= "<META NAME=\"DESCRIPTION\"
         CONTENT=\"$this->description\">\n";
        $meta .= "<META NAME=\"ROBOTS\"
        CONTENT=\"INDEX,FOLLOW\">\n";
        $meta .= "<META NAME=\"GENERATOR\"
         CONTENT=\"$this->company_name\">\n";
        $meta .= "<META NAME=\"AUTHOR\"
         CONTENT=\"$this->company_name\">\n";
        $meta .= "<META NAME=\"REVISIT-AFTER\"
         CONTENT=\"2 DAYS\">\n";
        $meta .= "<META NAME=\"RESOURCE-TYPE\"
        CONTENT=\"document\">\n";
        $meta .= "<META NAME=\"COPYRIGHT\"
        CONTENT=\"Copyright (c) 2003
         $this->company_name\">\n";
        $meta .= "<META NAME=\"DISTRIBUTION\"
         CONTENT=\"Global\">\n";
        $meta .= "<META NAME=\"GENERATOR\"
         CONTENT=\"$this->generator\">\n";
        $meta .= "<META NAME=\"RATING\"
        CONTENT=\"GENERAL\">\n";
        $meta .= "<META HTTP-EQUIV=\"REPLY-TO\"
        CONTENT=\"webmaster@yourdomain.com\">\n";
        $meta .= "<META HTTP-EQUIV=\"Content-Type\"
        CONTENT=\"text/html;
         charset=iso-8859-1\">\n";

        return $meta;
    }
}
?>
```

This class is a great learning tool for understanding many different PHP functions, especially text formatting and manipulation functions. Let's begin:

```
<?php
class Meta{
```

With any PHP script, you must issue the open tag `<?php`. Next, you initialize a class and give it a name by using the code `class Meta`. This class is named `Meta`. Also, you will enclose the entire contents of this class with the brackets `{}`.

Your first and only function in this META class is the function that generates the META content and returns it to the reference that executes this function in your PHP script. You create a custom function by giving it a name and defining the arguments allowed for that function; in this case, it is the `$ptitle` that you will get from your `layout.php myheader` function:

```
function metadata($ptitle){
```

Now you begin having fun with PHP! The first portion of the META function will use an `IF ELSE` control structure to determine if the `$ptitle` has any value assigned to it. If it does not, or the string is empty, you will use an alternative META description. Notice how you reference the `var $description` in this example from within this function. I will show you how to assign the value to the `var $description` later in this chapter. For now, take notice of how I used the `$this->description` operator:

```
// Formulate the description for each page.
if(empty($ptitle)){
   $description = $this->description;
} else {
   $description = "$ptitle - $this->description";
}
```

For text formatting, I will convert all of the words in the `$ptitle` string into lower case and assign this new converted string to the variable named `$keywords`:

```
// Make the keywords of the title lower case
$keywords = strtolower($ptitle);
```

Before you break apart the `$keywords` string, you need to remove any double spaces and replace them with a single space to ensure that your `str_replace` function works properly:

```
// Remove extra spaces
$keywords = str_replace("  ", " ", $keywords);
```

The next portion of code takes the `$keywords` string and convert it to a comma-separated word list for each word in the string. You achieve this by using the `str_replace` function once again:

```
// Make string comma seperated
$meta_words = str_replace(" ", ", ", $keywords);
```

Using the same method to determine if the `$ptitle` string is empty, you use the empty function along with the `IF ELSE` control structure. If the `$ptitle` string is empty, you will make a page title from the `$sitename var` and the `$slogan var`. This prevents no page title from being displayed at all:

```
// If no Page Title, Use Alternative
if(!$ptitle){
   $meta .= "<TITLE>$this->sitename - ".
   $meta .= "$this->slogan</TITLE>\n";
} else {
   $meta .= "<TITLE>$this->sitename: ".
   $meta .= "$ptitle</TITLE>\n";
}
```

The next portion of code uses the string appending technique to append to the output of `$meta`. You also use the `$this->` operator and all of the strings you have been creating and manipulating throughout the `meta` function so far:

```
// Append META content to the $meta string for output
$meta .= "<META NAME=\"KEYWORDS\"
  CONTENT=\"$meta_words, $this->keywords2\">\n";
   $meta .= "<META NAME=\"DESCRIPTION\"
  CONTENT=\"$this->description\">\n";
$meta .= "<META NAME=\"ROBOTS\"
  CONTENT=\"INDEX,FOLLOW\">\n";
$meta .= "<META NAME=\"GENERATOR\"
  CONTENT=\"$this->company_name\">\n";
$meta .= "<META NAME=\"AUTHOR\"
  CONTENT=\"$this->company_name\">\n";
$meta .= "<META NAME=\"REVISIT-AFTER\"
  CONTENT=\"2 DAYS\">\n";
$meta .= "<META NAME=\"RESOURCE-TYPE\"
  CONTENT=\"document\">\n";
$meta .= "<META NAME=\"COPYRIGHT\"
  CONTENT=\"Copyright (c) 2003
  $this->company_name\">\n";
$meta .= "<META NAME=\"DISTRIBUTION\"
  CONTENT=\"Global\">\n";
$meta .= "<META NAME=\"GENERATOR\"
  CONTENT=\"$this->generator\">\n";
$meta .= "<META NAME=\"RATING\"
  CONTENT=\"GENERAL\">\n";
$meta .= "<META HTTP-EQUIV=\"REPLY-TO\"
  CONTENT=\"webmaster@yourdomain.com\">\n";
$meta .= "<META HTTP-EQUIV=\"Content-Type\"
  CONTENT=\"text/html;
  charset=iso-8859-1\">\n";
```

Now you have one long string named `$meta` that contains the entire HTML, you need to pass back to your PHP script and fill in your `<HEAD></HEAD>` section for your page titles, META data, and so on. Now all you have left for this function is to return the string when the function inside this class is called within a PHP script. You achieve this last operation by using the return control structure:

```
return $meta;
```

Finally, you need to close this function and complete the class as well as issue the PHP close tag:

```
    }
}
?>
```

Congratulations! You have now completed your first useful PHP object-oriented Class. Let's put this OOP lesson to use!

Using the Meta Content Class

Now that you have your class created, it's time to test it out! Currently, the `layout.php` file does not have any META content, and the `<HEAD></HEAD>` sections are empty. Let's open the `layout.php` files and fill in those blanks with your spiffy new class!

The newly modified `layout.php` file will look like Listing 4.4 when you are done adding the META content class.

Listing 4.4 Modified *layout.php* File

```
<?php
function myheader($ptitle){

include $_SERVER['DOCUMENT_ROOT']."/classes/clsMetaContent.php";
$meta = &new Meta;

$meta->company_name = "My Company";
$meta->description = "This is my first PHP enabled website.";
$meta->keywords2 = "PHP, MySQL, Web Development";
$meta->sitename = "My PHP Site";
$meta->slogan = "Be patient, I'm learning!";
$meta->generator = "PHP";
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<?php echo $meta->metadata($ptitle); ?>
</head>

<body>
```

```
<table width="100%" border="2" cellpadding="0"
 cellspacing="0" bordercolor="#000000">
  <tr>
    <td colspan="3"><img src="/images/logo.jpg" ALT="My PHP Site"></td>
  </tr>
  <tr>
    <td> </td>
    <td>
    <!-- End Header and Begin Content -->
<?php
}
function footer(){
?>
<!-- End Content and Begin Footer -->
    </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
</table>
</body>
</html>
<?php
}
?>
```

The only part of the `layout.php` file that you have modified is the portion that contains the `myheader` function. You will begin right below the section where you named and initialized the `myheader` function. First, you need to include the `clsMetacontent.php` file:

```
include $_SERVER['DOCUMENT_ROOT']."/classes/clsMetaContent.php";
```

Once again, notice how you use the `$_SERVER['DOCUMENT_ROOT']` to ensure you have the complete path to the actual file included. Next, you will initialize the class and assign a reference variable to it:

```
$meta = &new Meta;
```

Do you remember all of that `var $varname` stuff I was talking about when you created the META Content class? Well, here is how you assign values to those vars:

```
$meta->company_name = "My Company";
$meta->description = "This is my first PHP enabled website.";
$meta->keywords2 = "PHP, MySQL, Web Development";
$meta->sitename = "My PHP Site";
$meta->slogan = "Be patient, I'm learning!";
$meta->generator = "PHP";
```

Skip down to below the <head> tag in your HTML portion and change it as follows:

```
<head>
<?php echo $meta->metadata($ptitle); ?>
</head>
```

NOTE If you desire to use the shortcut syntax for echo, you may use <?=$meta->metadata ($ptitle) ?> instead of the full <?php echo $meta->metadata($ptitle); ?>. If you want to use the shortcut syntax, you must ensure that the short_open_tag value is set to On in your php.ini file.

You can leave the rest of your layout.php script as you originally coded it in this chapter. Open your script in your web browser from your web server and take a look at the source of the HTML. You can do this via Internet Explorer by selecting View ➢ Source. You should see the HTML in Listing 4.5.

Listing 4.5 HTML Output to Web Browser from Test Scripts

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<TITLE>My PHP Site: Welcome to My Website!</TITLE>
<META NAME="KEYWORDS" CONTENT="welcome, to, my, website!,
 PHP, MySQL, Web Development">
<META NAME="DESCRIPTION" CONTENT="This is my first PHP enabled website.">
<META NAME="ROBOTS" CONTENT="INDEX,FOLLOW">
<META NAME="GENERATOR" CONTENT="My Company">
<META NAME="AUTHOR" CONTENT="My Company">
<META NAME="REVISIT-AFTER" CONTENT="2 DAYS">
<META NAME="RESOURCE-TYPE" CONTENT="document">
<META NAME="COPYRIGHT" CONTENT="Copyright (c) 2003 My Company">
<META NAME="DISTRIBUTION" CONTENT="Global">
<META NAME="GENERATOR" CONTENT="PHP">
<META NAME="RATING" CONTENT="GENERAL">
<META HTTP-EQUIV="REPLY-TO" CONTENT="webmaster@yourdomain.com">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
</head>

<body>
<table width="100%" border="2" cellpadding="0"
  cellspacing="0" bordercolor="#000000">
  <tr>
    <td colspan="3"><img src="/images/logo.jpg" ALT="My PHP Site"></td>
  </tr>
  <tr>
    <td> </td>
    <td>
    <!-- End Header and Begin Content -->
Welcome to My Website!
```

```
<!-- End Content and Begin Footer -->
    </td>
    <td> </td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
</table>
</body>
</html>
```

If you see the same output, or something similar, depending on the settings you chose for your code, then congratulations! You have just created your first portable OOP class that you can use in all of your websites. The search engines will love your site, and you will have page titles on all of your pages.

What's Next?

In this chapter, you have designed a simple layout and used that layout to create a theme for your entire website. You now have a method of simply modifying one file and changing every single page on your site easily. Throughout the rest of this book, you are going to be building on this basic template to create a complete website, so please do not delete your files created in this chapter yet!

You are moving right along, and it is now time to start working with databases, user input forms, and e-mail systems in the next chapter.