The track_vars configuration directive enables the recording of several important session variable arrays, including $HTTP_GET_VARS[], $HTTP_POST_VARS[], $HTTP_POST_FILES, $HTTP_COOKIE_VARS[], $HTTP_ENV_VARS[], and $HTTP_SERVER_VARS[]. These arrays are discussed in further detail in Chapter 13, "Cookies and Session Tracking."

It is important to note that there are many more configuration directives than the ones listed here, although those listed are likely to be the ones that most users will find useful. Many of these directives will be addressed in their respective later chapters.

Basic PHP Constructs

Now I'll introduce several preliminary concepts related to PHP before delving into the core topics of the language that make up the rest of this book.

Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as '*escaping to PHP*.' There are four ways to do this:

- Default tags
- Short tags
- Script tags
- ASP-style tags

Default Tags

The default tags are perhaps those most commonly used by PHP programmers, due to clarity and convenience of use:

```
<?php print "Welcome to the world of PHP!"; ?>
```

These tags may also be the most practical ones because the initial escape characters are followed by `php`, which explicitly makes reference to the type of code that follows. This can be useful because you may be simultaneously using

several technologies in the same page, such as JavaScript, server-side includes, and PHP. Any ensuing PHP code will then follow the initial escape sequence, preceded by the closing escape sequence, "?>".

Short Tags

The short tag style is the shortest available for escaping to PHP code:

```
<? print "Welcome to the world of PHP!"; ?>
```

Short tags must be enabled in order for them to work. There are two ways to do this:

- Include the –enable-short-tags option when compiling PHP.
- Enable the short_open_tag configuration directive found within the php.ini file.

Script Tags

Several text editors will mistakenly interpret PHP code as HTML (that is, viewable) code, interfering with the Web page development process. To eliminate this problem, use the following escape tags:

```
<script language="php">
print "Welcome to the world of PHP!";
</script>
```

ASP-Style Tags

A fourth and final way to embed PHP code is through the use of ASP (Active Server Page)-style tags. This way is much like the short tag way just described, except that a percentage sign (%) is used instead of a question mark.

```
<% print "Welcome to the world of PHP!"; %>
```

A variation of the ASP-style tag that can result in a lesser degree of code clutter is available. This variation eliminates the need to include a 'print' statement in the enclosed PHP code. The equals sign (=) immediately following the opening ASP tag signals the PHP parser to output the value of the variable:

```
<%= $variable %>
```

Making use of this convenient tag style, we could execute the following:

```
<%
// set variable $recipe to something…
$recipe = "Lasagna";
%>
Luigi's favorite recipe is <%=$recipe;%>
```

There are actually two separate PHP scripts in this listing. The first assigns the value "Lasagna" to the variable $recipe. Later on, when it is necessary to display the value of the variable $recipe, you can use the ASP-style variation for this sole purpose. Incidentally, you could also use short tags (`<?...?>`) in much the same way.

Embedding HTML in PHP Code

Perhaps the most powerful characteristic of PHP is its ability to both output and be written directly alongside other languages, HTML and JavaScript, for example. Listing 1-2 illustrates this concept.

Listing 1-2: Display of HTML using PHP code

```
<html>
<head>
<title>Basic PHP/HTML integration</title>
</head>
<body>
<?
```

Figure 1-2. A simple PHP function, date(), formats the date for display in the browser title bar.

```
// Notice how HTML tags are included in the print statement.
print "<h3>PHP/HTML integration is cool.</h3>";
?>
</body>
</html>
```

Listing 1-2 illustrates how PHP can incorporate HTML code directly in print statements. Notice how level-three header (`<h3>…</h3>`) tags can be placed right inside the PHP code. These tags will appear in the final document as if they were regular HTML output.

Listing 1-3 illustrates how PHP can dynamically insert information into a Web page. The current date will be inserted into the title, as shown in Figure 1-2.

Listing 1-3: Dynamic date insertion

```
<title>PHP Recipes | <? print (date("F d, Y")); ?></title>
```

The simple PHP function date() can format the current date in several different ways. This formatted date value can then be output into the title.

PHP is also capable of modifying the format of the HTML itself through the designation and subsequent insertion of tag characteristics in the file. Listing 1-4 shows how this is possible, assigning a font characteristic (h3) to a variable (`$big_font`) and later inserting it as needed in the display text.

Listing 1-4: Dynamic HTML tags

```
<html>
<head>
<title>PHP Recipes | <? print (date("F d, Y")); ?></title>
```

```
</head>
<?
$big_font = "h3";
?>
<body>
<? print "<$big_font>PHP Recipes</$big_font>"; ?>
</body>
</html>
```

Listing 1-4 is a variation of Listing 1-3, this time first assigning level-three header (`<h3>…</h3>`) tags to a variable and then later using this variable in a print statement. These tags will appear in the final document as if they were regular HTML output.

Multiple-PHP Script Embedding

To allow for flexibility when building dynamic Web applications, you can embed several separate PHP scripts throughout a page. Listing 1-5 illustrates this.

Listing 1-5: Embedding multiple PHP scripts in a single document

```
<html>
<head>
<title>
<?
    print "Another PHP-enabled page";
    $variable = "Hello World!";
?>
</title></head>
<body>
<? print $variable; ?>
</body>
</html>
```

Listing 1-5 begins as a typical (albeit simple) HTML page would. The flexibility offered by this feature is that variables can be assigned in one code section and still used later on in another code section on the same page.

Commenting PHP Code

You should sufficiently comment the code even for relatively short and uncomplicated scripts. There are two commenting formats in PHP:

- *Single-line comments* are generally used for short explanations or notes relevant to the local code.

- *Multiline comments* are generally used to provide pseudocode algorithms and more detailed explanations when necessary.

Both methods ultimately result in the same outcome and have no bearing on the overall performance of the script. Which to use is left up to you.

Single-Line Comments

Two commenting styles are geared toward single-line comments. Both work exactly the same way, but they employ different escape characters. One style uses a double backslash (`//`) at the beginning of a comment, and the other style uses a pound symbol (`#`) at the beginning of a comment. Here are examples of each style:

```
<?
// set the color of the roses.
$rose_color = "red";

# set the color of the violets.
$violet_color = "blue";
print "Roses are $rose_color, violets are $violet_color";
?>
```

Of course, it is possible to use single-line comments to build multiline comments using either style, as seen in the following listing: