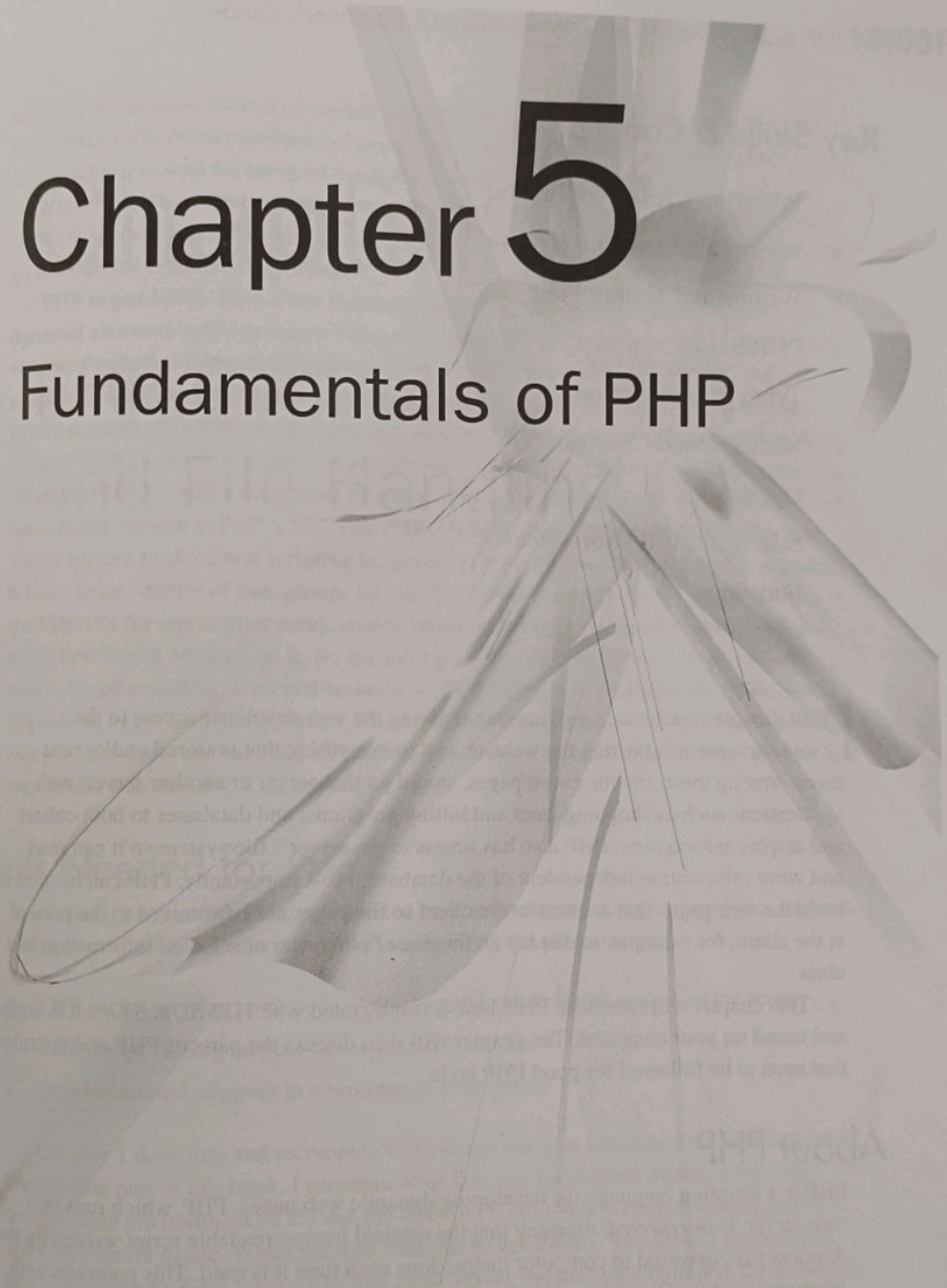
Chapter 5

Fundamentals of PHP



Key Skills & Concepts

- Preparing a PHP Workstation
- Integrating PHP with HTML
- Writing and Testing PHP
- PHP Basics
- Types of Information
- Variables and Constants
- Operators
- Statements and Expressions
- Functions

PHP provides server-side programming, giving the web developer access to the web server that is hosting the website, and to everything that is stored and/or runs there. Among these are other web pages, stored on that server or another server; web applications such as shopping carts and billing programs; and databases to both collect and display information. PHP also has access to the server's file system so it can read and write information independent of the database. Most importantly, PHP can be used to build the web pages that are sent to the client so the pages are customized to the person at the client, for example, to display an invoice of purchases or selected information in a class.

This chapter will introduce PHP, how it is integrated with HTML, and how it is written and tested on your computer. The chapter will then discuss the parts of PHP and the rules that need to be followed for good PHP code.

About PHP

PHP is a scripting language for developing dynamic web pages. PHP, which runs in a web server, is *interpreted*, meaning that the original human-readable script written by the developer is converted to computer instructions each time it is used. This contrasts with languages such as C and Java (not JavaScript), which are *compiled* instead of interpreted,

meaning the original code is converted to computer instructions as a one-time process and stored in the converted form, so when it is used, it can immediately be executed and does not have to wait for interpretation. While interpreted languages like PHP may be slightly slower, they can be changed immediately before they are used, giving them a lot of flexibility and making development much easier—something you'll appreciate as you go through the remainder of the book.

PHP began life in 1994 when Rasmus Lerdorf developed it to help him add some dynamic elements to his *personal home page,* hence PHP. Since then, it has spread like wildfire because it is powerful and easy to use, it integrates well with HTML, and it supports a number of databases, most importantly, MySQL. Rasmus also put it in the public domain, which means that not only is it free to use, but anybody with the skill can enhance it. As a result, a number of people have worked on it. Today, it is in its fifth major revision (PHP 5), with a number of minor revisions along the way (as this is written, the latest stable version is PHP 5.5.9, with PHP 5.6 in developmental testing). PHP is now a fully mature professional scripting language, as you can see from its website, php.net. It has a large number of user groups all over the world (valuable for getting help—see the PHP site for one in your area), sizable steering and documentation committees, and many developers working on it, for the most part, on a volunteer basis. It also now has a more official sounding, if recursive, name—"PHP: Hypertext Preprocessor." While it is impossible to get an accurate count, many millions of web pages have been developed using it, and many professional web developers believe that PHP and MySQL are the best way to add a database to a website of any size.

Tools Needed for PHP

To effectively work with PHP, you need to have a development environment that supports both the writing and the testing of PHP scripts. You'll need tools for:

- **PHP script writing**, with code assistance and validation as you write the script
- **PHP script testing** on your development computer
- **Developmental support** in a browser, to help debug

Chapter 1 describes and recommends packages that you can download for these tools. For the first part of this book, I recommended that you use Aptana Studio integrated development environment (IDE) and the WAMP (Windows servers for Apache, MySQL, and PHP) server group. While you can continue to use these if you wish, I recommend that you switch to Zend Studio IDE and Zend Server because they both provide powerful

support for PHP (they were developed by a couple of the early PHP developers). Zend Studio, which uses the same foundation as Aptana, provides better PHP debugging help.

Preparing a PHP Workstation

If you do not already have Zend Studio and Zend Server, return to Chapter 1 and follow the instructions to download and install them. They are used in the discussion in the remaining chapters of this book and will prove very beneficial to your PHP work.

Setting Up Zend Server

When you have completed downloading Zend Server, use these steps to set it up and prepare it for use.

1. After completing the installation of Zend Server, you are left with a Thank You page in your browser, as you see in Figure 5-1.

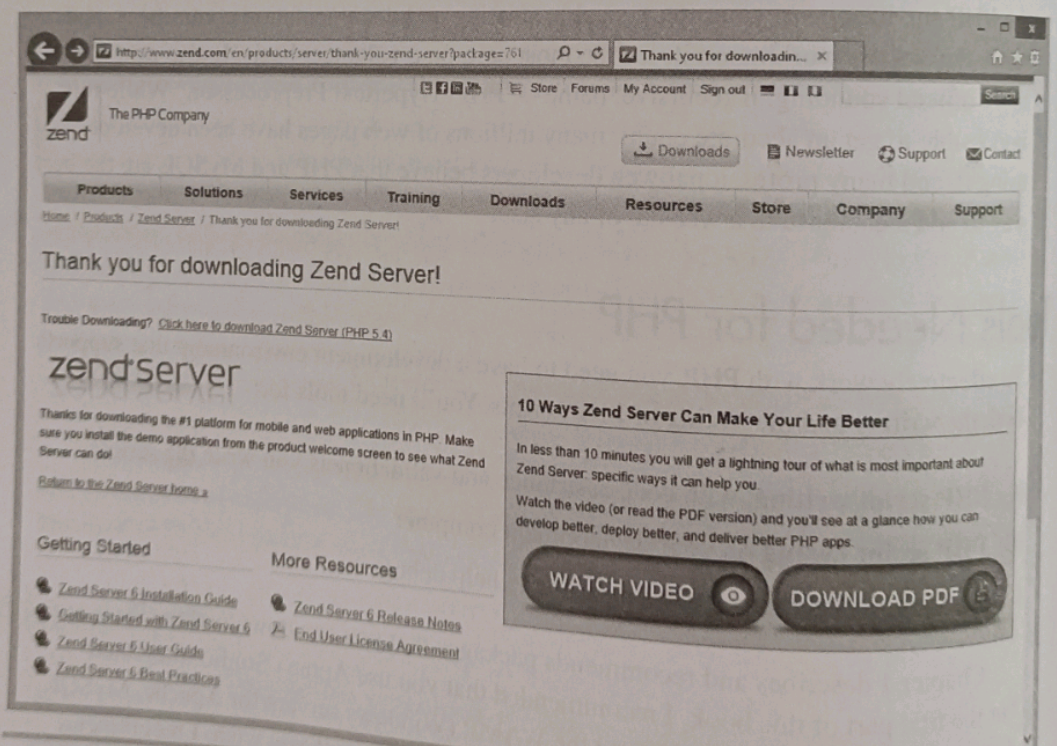
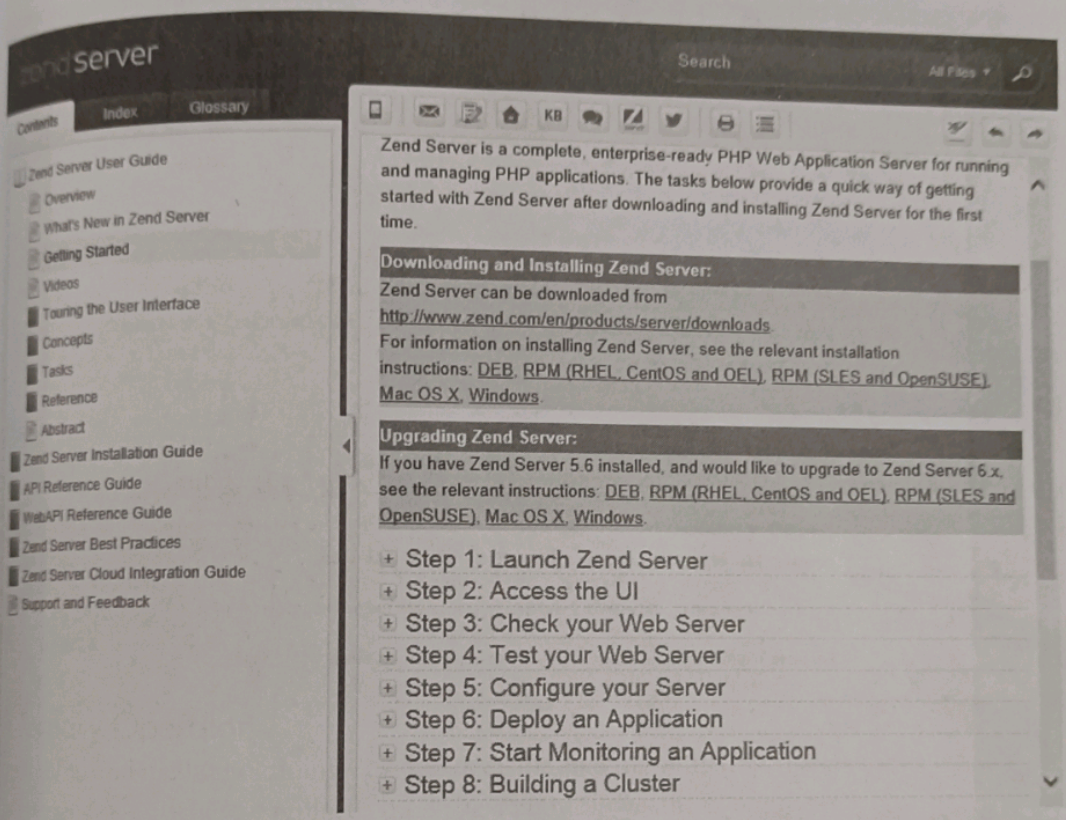


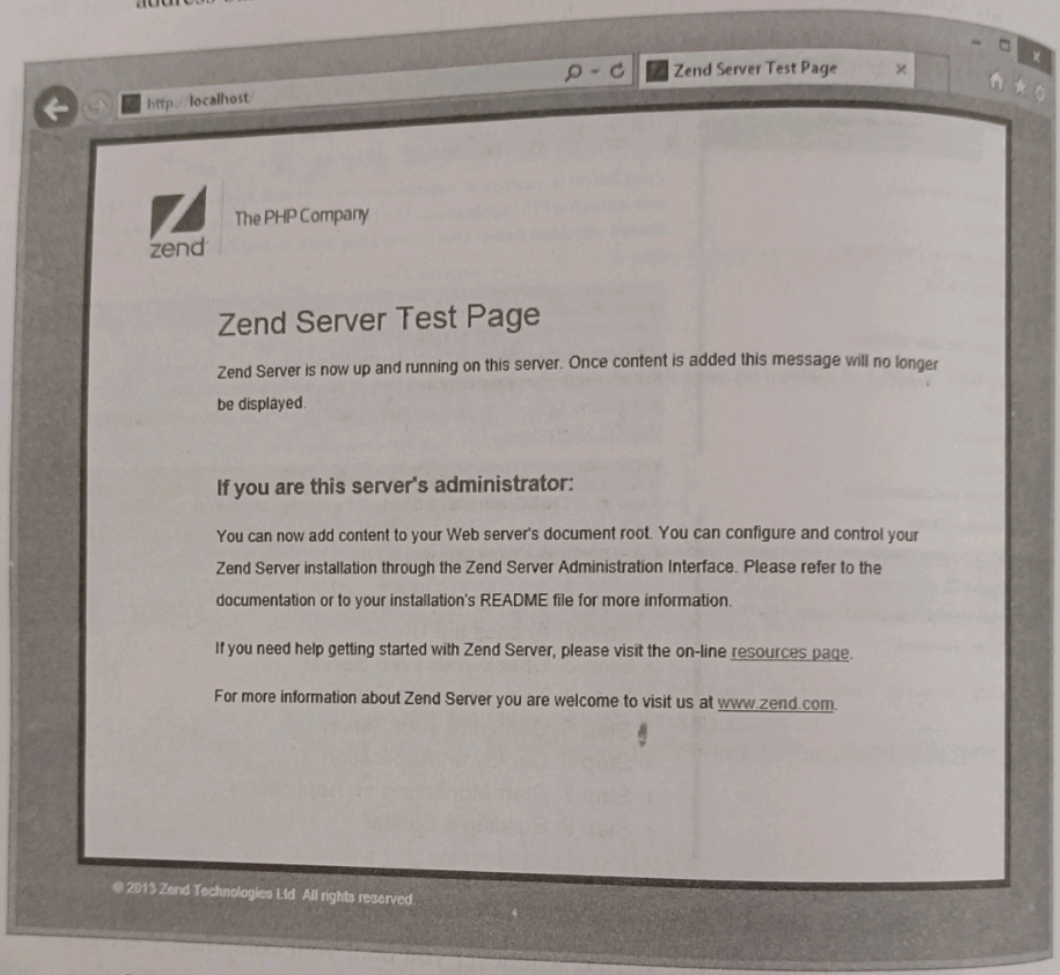
Figure 5-1 Zend Studio provides a number of pages to review to prepare for its use.

2. Open the Getting Started link | Zend Server User Guide | Getting Started. You did step 1 as a part of the installation, so begin with step 2 and work down each of the steps. Here are some tips:



a. The Apache server component of Zend Server that provides the localhost in your browser is automatically started whenever you boot your computer. You can tell the Apache server is running by hovering over this symbol in the notification area of the taskbar or in the hidden icons.

b. Test the localhost in a browser by opening a browser and typing **localhost** in the address bar. The Zend Server Test Page should appear.



c. Open the Zend Server user interface (UI) in your browser by typing **localhost:10081/zendserver**. You are asked to log in. For the username, type **admin**, and then enter the password you entered in step 10 near the end of the installation. This should open the page shown in Figure 5-2.

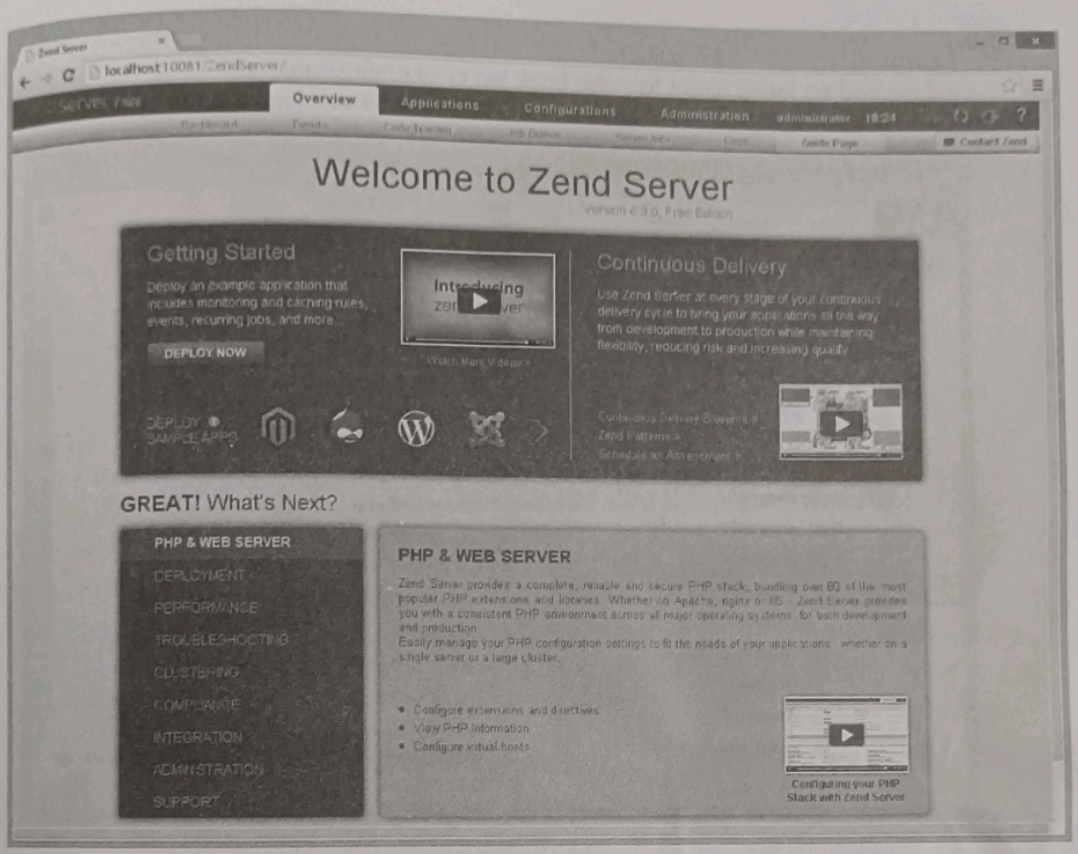
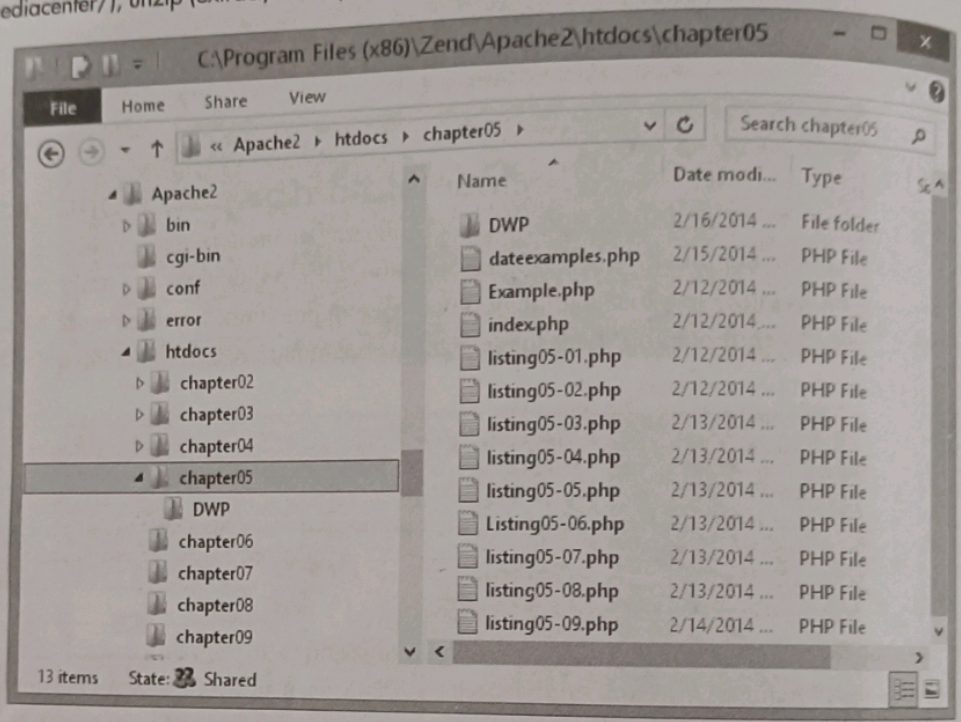


Figure 5-2 The initial view of the Zend Server UI should be the Overview Guide Page.

d. For files to appear in your browser at localhost, place them in the folder at c:\program files (x86)\zend\apache2\htdocs, assuming that Zend Server is installed in the default folder c:\program files (x86).

TIP

If you download the listing scripts from this book (go to mhprofessional.com/mediacenter/), unzip (extract) the .zip file into the htdocs folder so it looks like this:



Setting Up Zend Studio

When you finished installing Zend Studio, it should have left a shortcut on your desktop. Use that shortcut to start Zend Studio, and with the following steps, set up Zend Studio for use.

1. When Zend Studio first opens, the Workspace Launcher is displayed with the default workspace. Based upon what we just did in the steps for Zend Server, enter or browse to the c:\program files (x86)\zend\apache2\htdocs folder, click Use This As The Default, and click OK. Zend Studio should open, as you see in Figure 5-3.

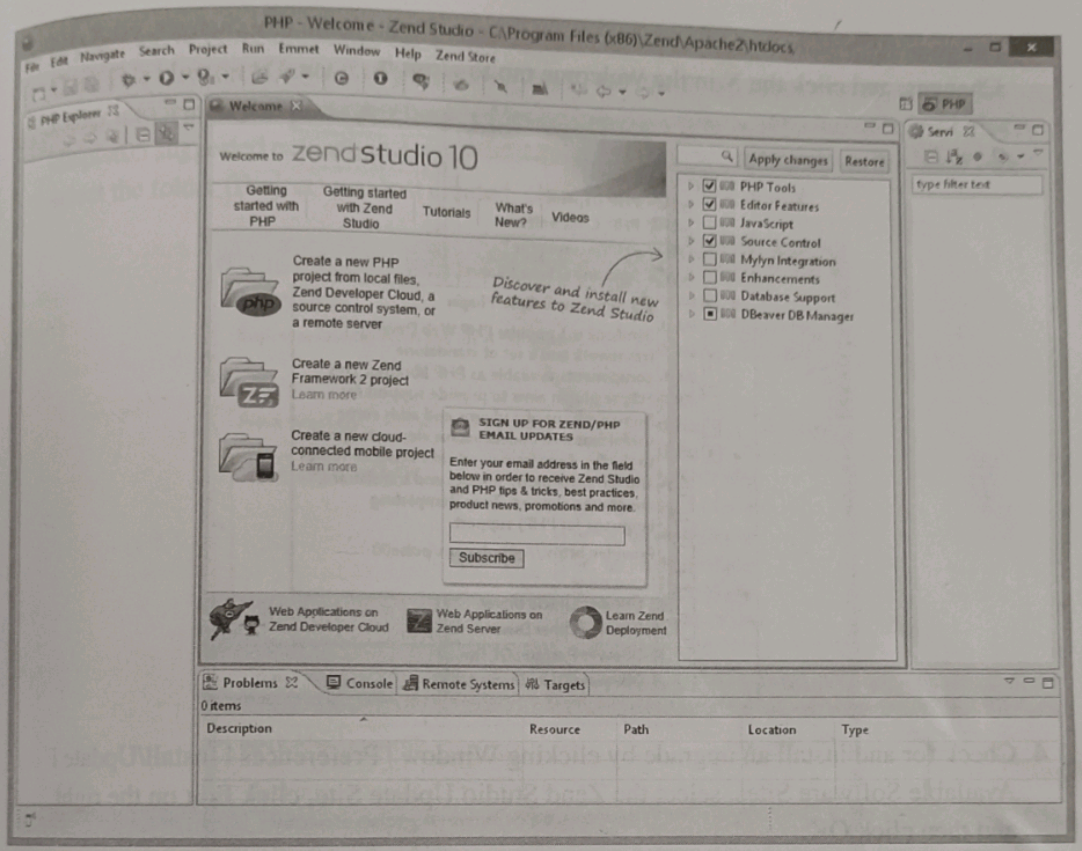
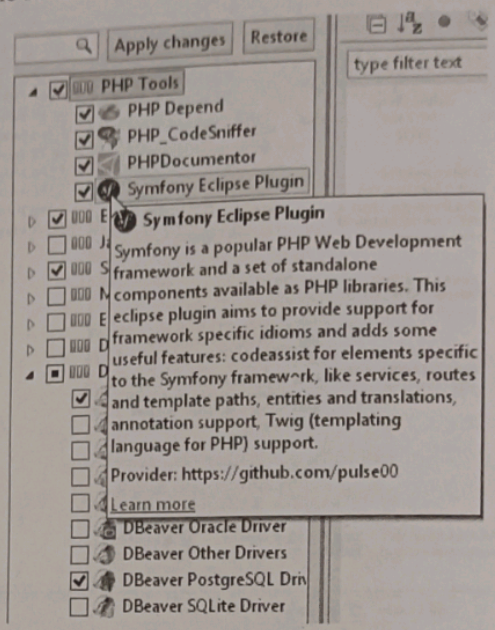


Figure 5-3 Zend Studio provides an environment and support tailored to PHP.

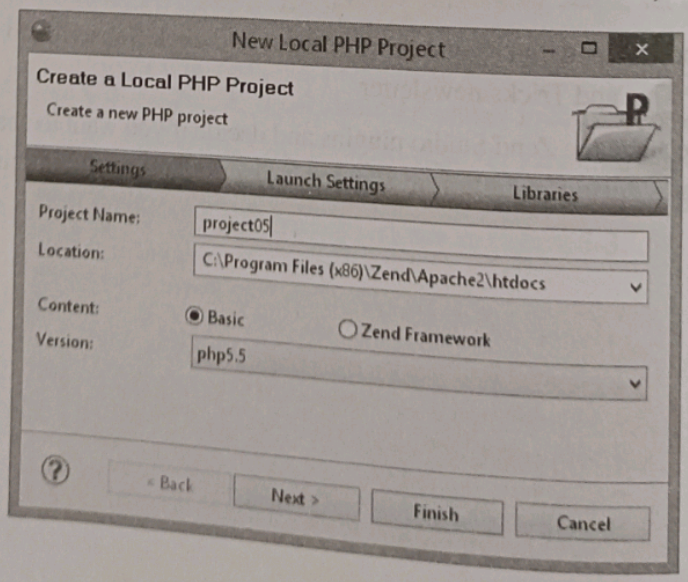
2. I recommend that you enter your email address and click Subscribe to receive the Zend Studio PHP Tips and Tricks newsletter.
3. Review once again the Zend Studio plugins and decide if you want to change the selections that you made during installation. We don't discuss any of the plugins in this book,

but you might want to explore some on your own. If you make any changes, click Apply Changes, and click the X in the Welcome tab to close it.

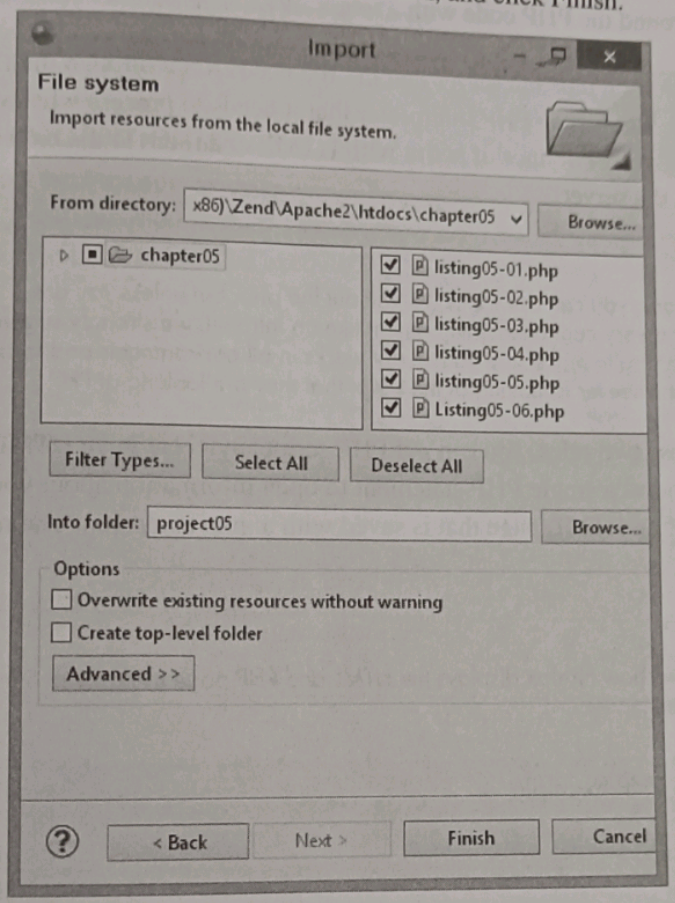


4. Check for and install an upgrade by clicking Window | Preferences | Install/Update | Available Software Sites, select the Zend Studio Update Site, click Edit on the right, and then click OK.

5. Create a PHP project in which to store the files you create in this chapter by clicking File | New | Local PHP Project. Enter a project name, such as **Project05**; if the location isn't already c:\program files (x86)\zend\apache2\htdocs, make it so, and click Finish.



6. If you want to bring in the listings from this book, you can import them into your project. Click File | Import | General | File System | Next. For the From Directory, browse to the folder into which you place this book's files (c:\program files (x86)\zend\apache2\htdocs\ chapter05 suggested earlier), select the individual files you want, browse to the Into Folder, select the folder (Project) you just created, click OK, and click Finish.



PHP Introduction

PHP is a way within a web page of telling the server that you want something done. It may be to open another web page, write in a file, or extract information from a database. Whatever it is, in an HTML page, the developer has decided that he or she needs something from the server. This is the job of PHP. Note that it started from an HTML page, so the first question is how to put PHP on an HTML page. The second question is what can be done with it once it is there.

Integrating PHP with HTML

Putting PHP in an HTML file is very simple:

- You change the file extension to .php.
- You surround the PHP code with `<?php ... ?>`.

The .php extension tells the server that the file needs to go through the PHP interpreter. The `<?php ... ?>` tells the PHP interpreter that it needs to process whatever is in the middle and to replace the PHP lines of script with HTML code sent to the browser so it can interact with the server.

NOTE

In some situations, you can use `<? ... ?>` without the `php`, but unless you are developing for a very controlled environment like an intranet, we strongly suggest that you include the `php` to make sure your script works in all environments and to simply alert or remind whoever is looking at the script that they are looking at PHP.

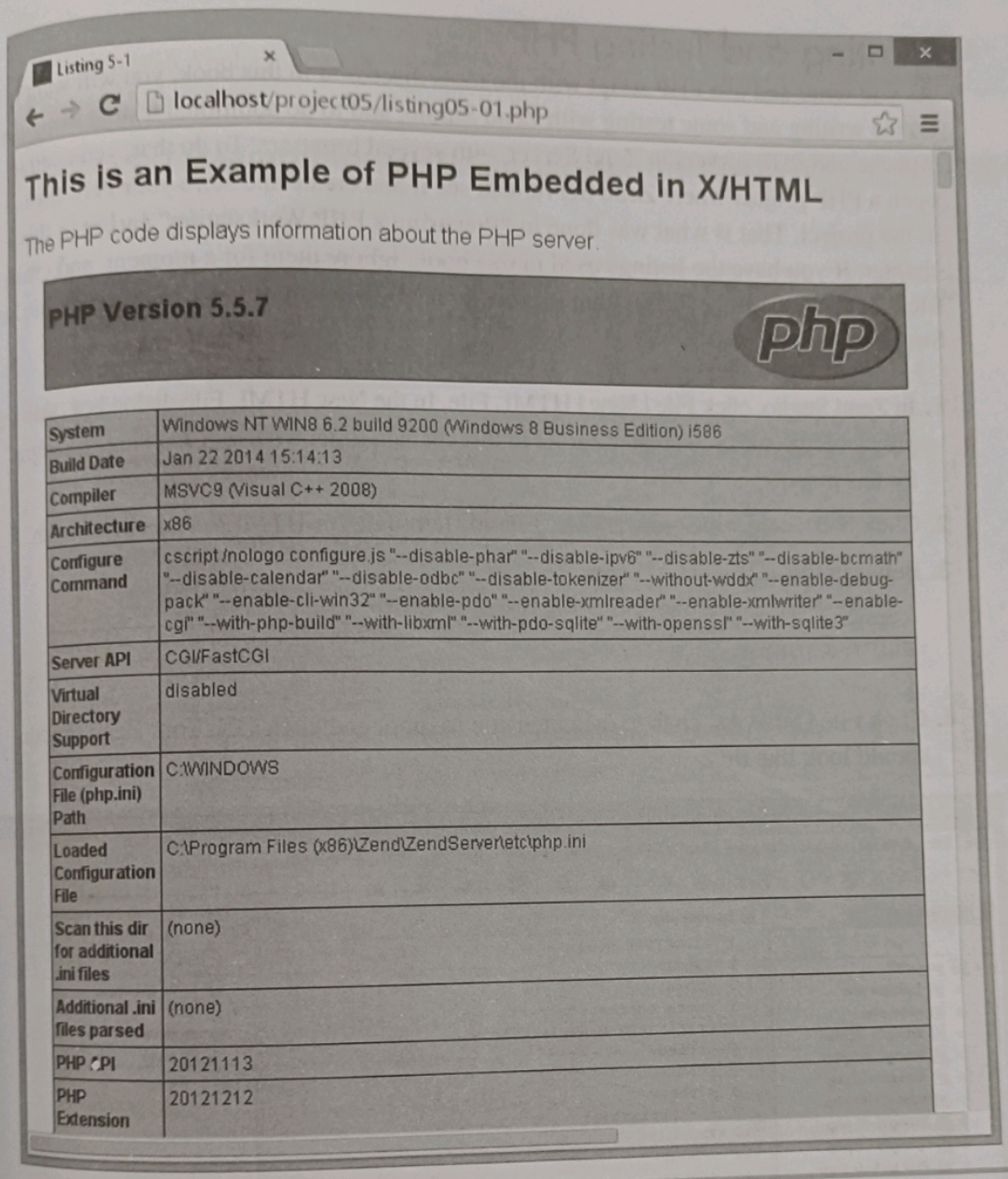
Given those two rules, you can put PHP script anywhere in an HTML file. For example, Listing 5-1 shows a single PHP statement to open information about your PHP server in the body of an HTML page that is saved with a .php extension. The result is shown in Figure 5-4.

NOTE

Figure 5-4 shows how Firefox displays the HTML and PHP page using Zend Server with a test PHP server.

Listing 5-1 PHP Example

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Listing 5-1</title>
   </head>
   <body>
      <h1>This is an Example of PHP Embedded in HTML</h1>
      <p>The PHP code displays information about the PHP server.</p>
      <?php
         phpinfo();
      ?>
   </body>
</html>
```



System	Windows NT WIN8 6.2 build 9200 (Windows 8 Business Edition) i586
Build Date	Jan 22 2014 15:14:13
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--disable-phar" "--disable-ipv6" "--disable-zts" "--disable-bcmath" "--disable-calendar" "--disable-odbc" "--disable-tokenizer" "--without-wddx" "--enable-debug-pack" "--enable-cli-win32" "--enable-pdo" "--enable-xmlreader" "--enable-xmlwriter" "--enable-cgi" "--with-php-build" "--with-libxml" "--with-pdo-sqlite" "--with-openssl" "--with-sqlite3"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files (x86)\Zend\ZendServer\etc\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20121113
PHP Extension	20121212

Figure 5-4 PHP operates from any part of an HTML page.

NOTE

The remaining listings in this chapter will not include the `<!DOCTYPE>` and `<meta>` statements to make the listings more compact.

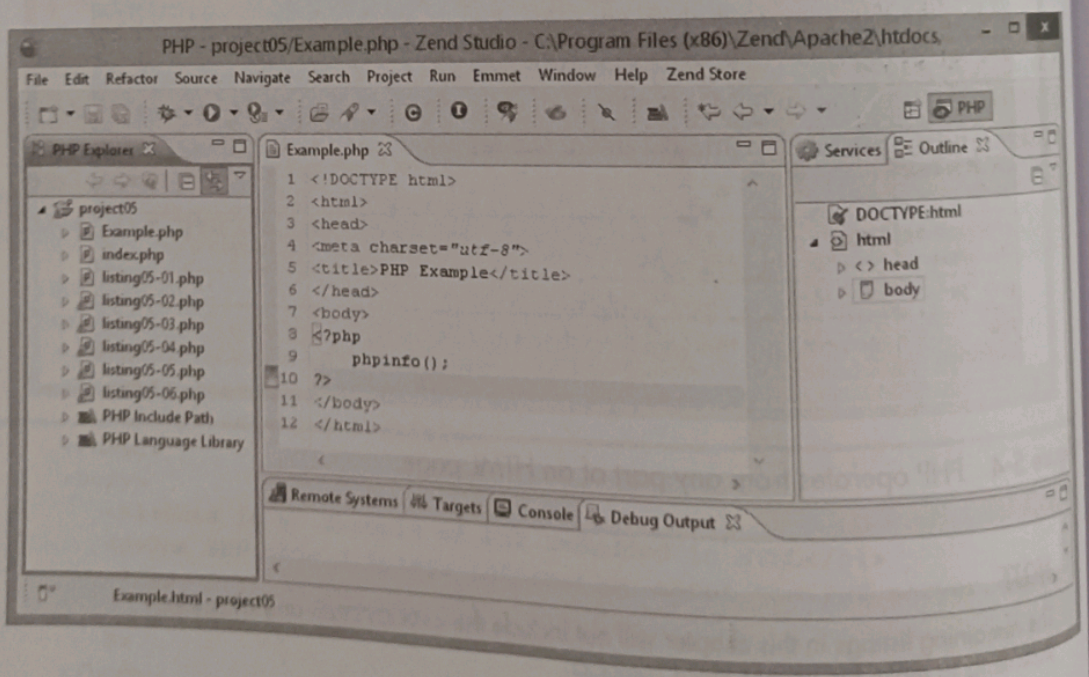
Writing and Testing PHP

As you write and test PHP script with the tools discussed in this book, you will want to do the writing and some testing with Zend Studio, and to do additional testing using the Apache and PHP servers in Zend Server with several browsers. To do that, you need to open a PHP project where Zend Server can see it and create and store your PHP files in the project. That is what was done in "Preparing a PHP Workstation" earlier in this chapter. If you have the listings used in this book, ignore them for a moment, and create a file that has PHP embedded within it. Then look at the result of this code, first within Zend Studio and then in browsers.

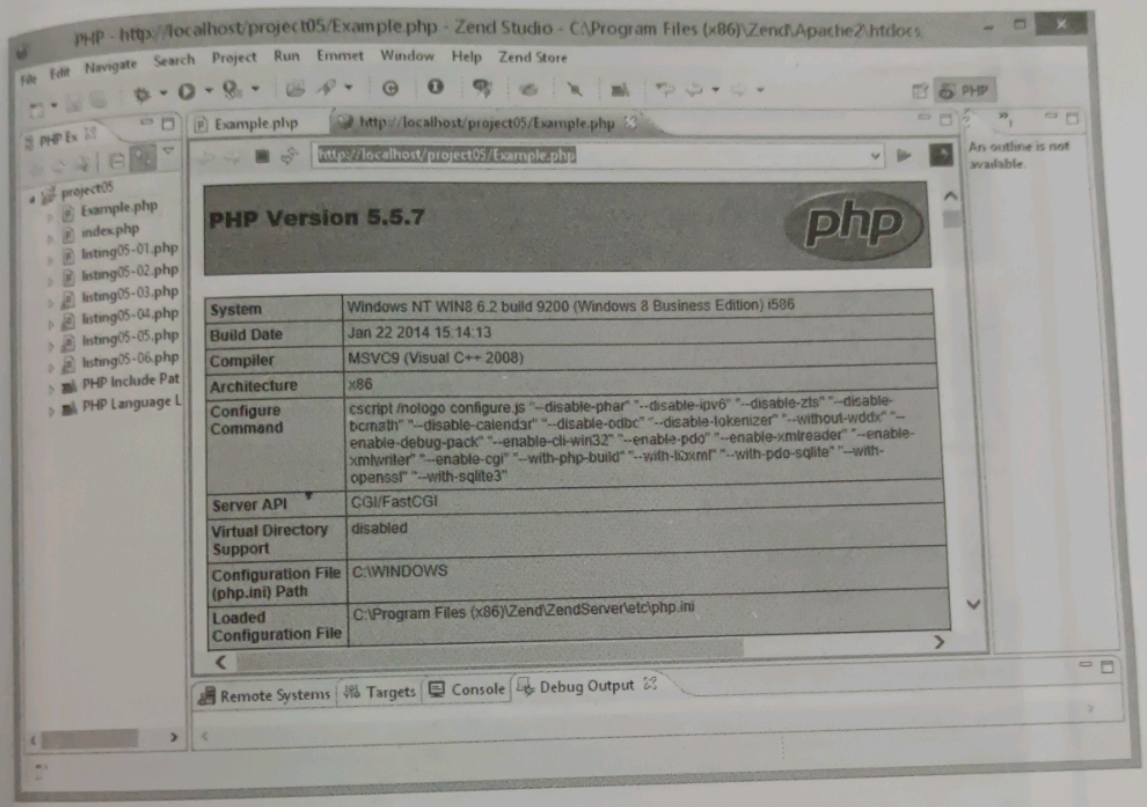
1. In Zend Studio, click File | New | HTML File. In the New HTML File dialog box, select the project, project05, you created earlier in this chapter. Enter a name; I'm using "Example" for this purpose.
2. Click Next, select the New HTML File (5) template for HTML5, and click Finish.
3. Between the `<body>` tags, enter:

```
<?php
phpinfo();
?>
```

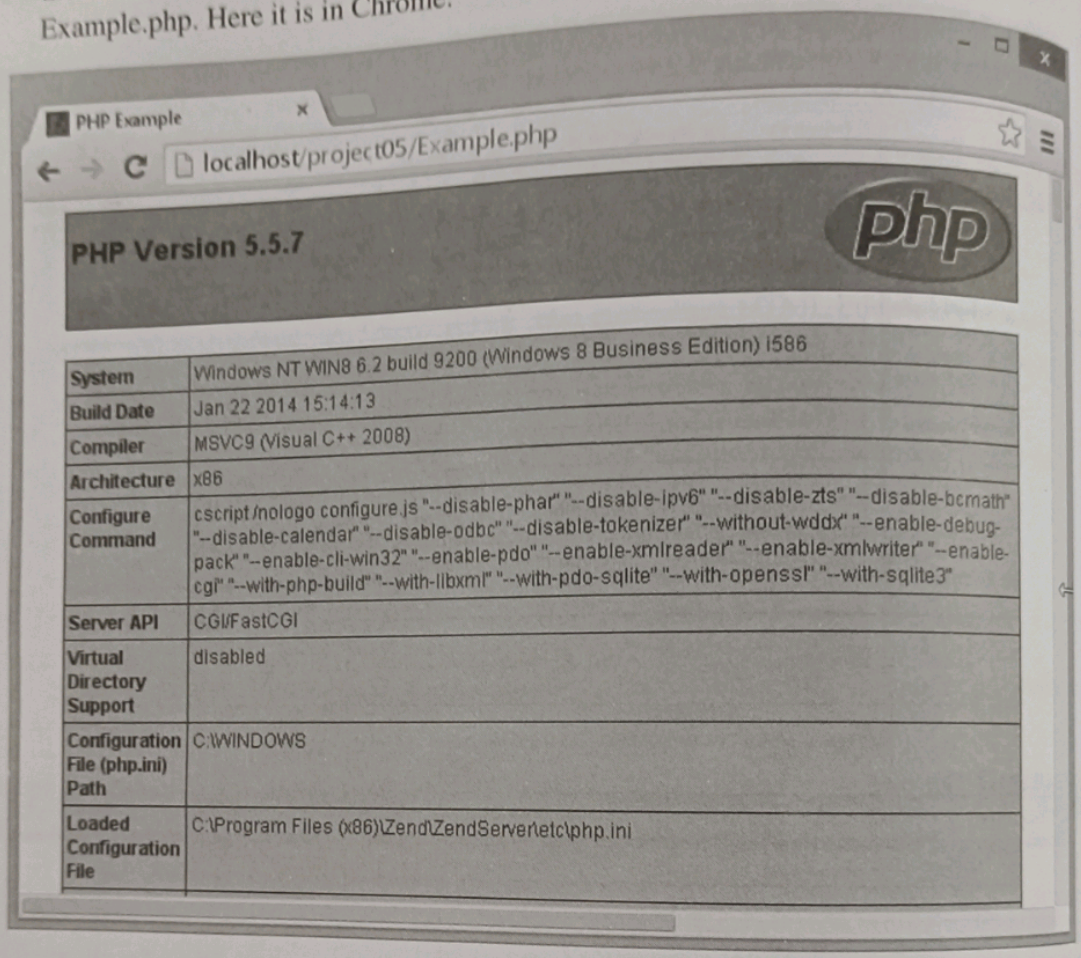
4. Click File | Save As, change the extension to .php, and click OK. Your Zend Studio should look like this:



5. To display the results of the code within Zend Studio, click the Run icon | PHP Web Application | OK, confirm the web server URL, and click OK. Here's the result:



6. You can get the same result outside of Zend Studio by opening a browser and entering the same URL (you can copy it from the display in Zend Studio): localhost/project05/Example.php. Here it is in Chrome:



System	Windows NT WIN8 6.2 build 9200 (Windows 8 Business Edition) i586
Build Date	Jan 22 2014 15:14:13
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--disable-phar" "--disable-ipv6" "--disable-zts" "--disable-bcmath" "--disable-calendar" "--disable-odbc" "--disable-tokenizer" "--without-wddx" "--enable-debug-pack" "--enable-cli-win32" "--enable-pdo" "--enable-xmlreader" "--enable-xmlwriter" "--enable-cgi" "--with-php-build" "--with-libxml" "--with-pdo-sqlite" "--with-openssl" "--with-sqlite3"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files (x86)\Zend\ZendServer\etc\php.ini

If you do not see the correct results in steps 5 and 6, then most likely, the file location is not where the Apache web server thinks it ought to be. As mentioned earlier, the default location is c:\program files (x86)\zend\apache2\htdocs. Use Windows Explorer to see if the file is there.

If it works in Zend Studio and not in your browser, then make sure the Apache server is running as you tested earlier in "Setting Up Zend Server." If so, look at what is in the browser's address bar. It should be localhost/project05/Example.php.

It is very important, before you go further in this chapter, that you make sure that Zend Studio and Zend Server, along with your browsers, are all working properly. You need to be able to test your work as you go along.

TIP

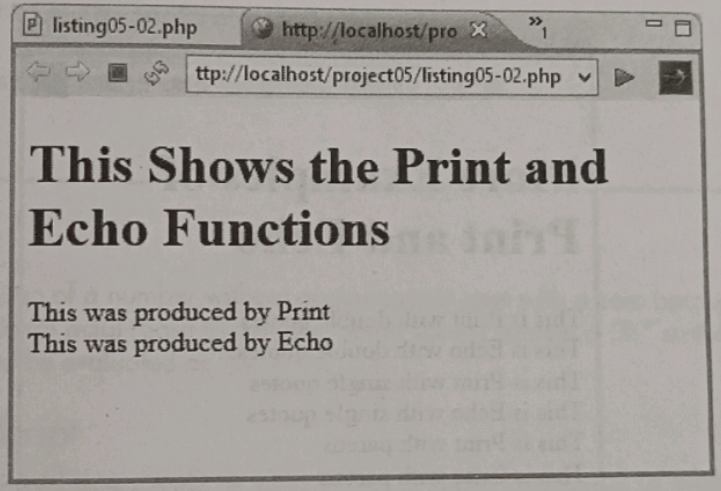
Zend Studio, Apache Friends, and PHP all have forums and user groups that you can contact with questions on how to resolve problems.

PHP Basics

PHP has several basic facilities that both give it power and make it easier to use. These include the ability to display information at any point in your script, adding comments in several ways, and a set of useful coding conventions.

Display Information

In debugging script, it is often helpful to display a comment or a variable while the script is running. PHP gives you two ways to do this with the `echo()` and `print()` functions, which are very similar, and for many purposes, they are the same. Listing 5-2 shows both functions, which produce this image with the Run command in Zend Studio:



Listing 5-2 print/echo Example

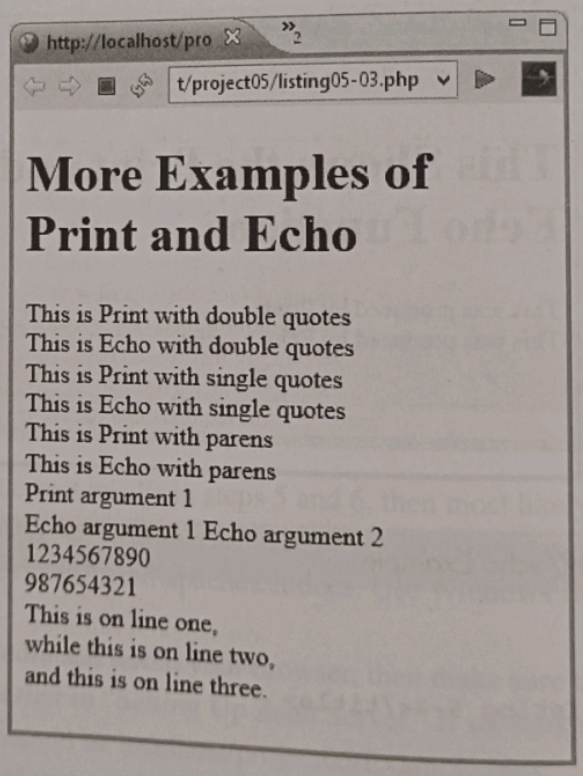
```
<html>
  <head>
    <title>Listing 5-2</title>
  </head>
  <body>
    <h1>This Shows the Print and Echo Functions</h1>
    <?php
      print "This was produced by Print <br />";
      echo "This was produced by Echo";
    ?>
  </body>
</html>
```

Both `print` and `echo`, while called "functions" and included in function lists, are unlike most functions in that their arguments are not required to be in parentheses, although you can use parentheses if you want. The quote marks are required unless the argument is a number, although you can use either single or double quote marks.

Since the information within the quotes is sent to the browser for processing, you can include HTML tags and have them treated as if they are in a line of HTML. The `<br />` in Listing 5-2 shows this. This also means that you can place text on multiple lines within a single set of quotes.

There are only two significant differences between `print` and `echo`. `print` returns a value (1) when it is processed, so you can test to see if that happened (explained later in the chapter). `echo` does not do that, but `echo` can take multiple arguments, separated by commas, while `print` can take only one.

Examples of the various ways that `print` and `echo` can be used are shown in Listing 5-3, with the results shown next:



Listing 5-3 More print and echo Examples

```
<body>    <h1>More Examples of Print and Echo</h1>
  <?php
     print "This is Print with double quotes <br />";
     echo "This is Echo with double quotes <br />";
     print "This is Print with single quotes <br />";
     echo "This is Echo with single quotes <br />";
     print ("This is Print with parens <br />");
     echo ("This is Echo with parens <br />");
     print "Print argument 1<br />"; #Second causes error
     echo "Echo argument 1  ", "Echo argument 2<br />";
     print 1234567890;  //Without quotes
     echo "<br />" , 987654321 , "<br />";
     print "This is on line one,<br />
        while this is on line two, <br />
        and this is on line three.";
     /* echo and print are essentially
        the same with one argument. */
  ?>
</body>
```

NOTE

In Listing 5-3, the echo of a number without quotes cannot start with a zero because PHP interprets that as an octal (base 8) number, which cannot contain a "9," so the whole number would be evaluated as "0."

Commenting Script

PHP allows you to add comments to your script in three ways, one of which was shown in Listing 5-3:

- On a single line starting with #

```
#This is a comment
```

- On a single line starting with //

```
//This is also a comment
```

- On multiple lines enclosed with /* */

```
/* This is a comment that can
be on several lines. */
```

TIP

It is strongly recommended that you comment your script profusely to make it easier to work with in the future.

Coding Conventions

PHP is fairly easygoing as far as conventions are concerned. In the first three listings, though, you have probably noticed several conventions that should be added to your PHP rules:

- Each PHP statement should end with a semicolon (`;`). The only exception is the last statement before the closing `?>`, which can have a semicolon, but it is not required.
- Text, any combination of letters and numbers, also called a *string*, in an argument needs to be enclosed in quotation marks, either single (`' '`) or double (`" "`). (Some differences are explained in a Note later in this section.) Quotation marks must be in like pairs.
- Legitimate numbers, which can have a decimal point, do not have to be in quotation marks.
- Multiple arguments are separated by commas (`,`).
- Most functions require that their arguments be enclosed in parentheses.

TIP

When you are debugging script and you get the message "Syntax Error," check to see if a semicolon is missing at the end of the previous line or if you have an unclosed/unmatched quote mark. Often, one of those is the problem.

You may have wondered since quotation marks are used to identify a string, how you display a quotation mark that is part of the string to be printed or echoed. For this purpose, PHP uses the backslash (\) in what is called an *escape sequence*, like this:

```
print "My name is \"Marty\"";
```

Most of the characters that PHP assigns a special use for can be used as a literal character by preceding it with a backslash. In addition, PHP has defined several escape sequences. Here are some of the more common escape sequences:

- `\"` produces a double quotation mark.
- `\'` produces a single quotation mark.
- `\\` produces a backslash.
- `\$` produces a dollar sign.
- `\r` produces a carriage return.

- `\n` produces a linefeed.
- `\t` produces a tab.

NOTE

While either single or double quotation marks can be used to enclose a string, in some circumstances, one or the other is preferable. With single quotation marks enclosing the string, you can use literal double quotation marks without the backslash in the string, as shown next. With single quotation marks, though, escape sequences other than `\'` or `\\` will display the backslash and not perform their function. With double quotation marks, all escape sequences work. For example: `print 'My name is "Marty"';`

Parts of PHP

PHP, while a relatively simple language compared with C or Java, is still very complex, as you would expect of any comprehensive language such as PHP. In this section, we'll take PHP apart and explore PHP information types, statements, variables, operators, and functions. In the following section, we explore control structures, classes, and objects.

NOTE

As with any language, a large number of elements are in each part of the language, far more than can be covered here. To look at the complete list, go to php.net/manual/en/langref.php. Also, w3schools.com/php/ provides a good reference as well as a good tutorial.

Types of Information

PHP is not as sensitive as other languages to the type of information you are working with, and you normally do not have to specify a data type, such as integer or string. PHP will determine it for itself based on the context in which you are using it. It is still a good idea for you to keep in mind the type you are working with. The possible data types are shown in Table 5-1. There is further discussion of these in the next chapter, including how the type is set, unset, and used with variables and arrays.

All of the examples in Table 5-1, except for `NULL`, are also arrays. Other notes on data types include the following:

- The Boolean `FALSE` is equivalent to the integer 0, the floating point number 0.0, an empty string or a string of `"0"`, an array of zero elements, or `NULL`. Everything else is `TRUE`.
- Integers are, by default, decimal (base 10) numbers. To make a number octal (base 8), precede it with `0` (zero), for example, `02` or `06`. To make a number hexadecimal (base 16), precede it with `0x`, for example, `0x4` or `0x8`.
- Very large integers (larger than 2,147,483,647) are considered floating point numbers.

Data Type	Name	Description	Examples
Arrays	`array`	A set of two or more pieces of data that can be any of these data types in a comma-separated list	[98101, 'WA', 'Seattle', '123 E 3rd']
Booleans	`bool`	Either TRUE or FALSE; not case sensitive, but commonly uppercase	TRUE, FALSE
Floating point numbers	`float`	A fractional number with a decimal; may be negative, and may use scientific notation	7.34, -21.89, 2.31e3
Integers	`int`	A whole number without a decimal; may be negative	43, 928, -4
Null	`null`	The absence of any value	NULL
Strings	`string`	A series of characters (one of 256 letters, numbers, and special characters) enclosed in either single or double quotation marks	"Mike", 'Seattle', "1495 W. 18th St"
Objects	`object`	Data and information to process it, often a piece of script	
Resources	`resource`	A reference to an external element; commonly used with MySQL	

Table 5-1 PHP Data Types

- If you divide two integers, you get a floating point number, unless the numbers are evenly divisible.
- Floating point numbers are not accurate to the last digit because of the infinite progression of fractions like one-third. Therefore, you should not compare two floating point numbers for equality.
- A string containing a number (either integer or floating point) immediately following the left quote can be used as a number. For example, `"18.2"` and `"4 cars"` can both be used as numbers, while `"his 4 cars"` cannot.
- A specific value in an array is identified with a *key*, which can be an integer, a string that evaluates as an integer, the truncated integer portion of a floating point number, the Boolean `TRUE`, which evaluates to the integer `1`, or `FALSE`, which evaluates to the integer `0`.

NOTE

In this book, as in the PHP Manual, two additional pseudo-types are used for discussion purposes only: "mixed" is used for a combination of any of the first six types, and the "numbers" type is used for a combination of integers and floating point.

Variables and Constants

As you write PHP script, you need to name items that you are working with so you can repeatedly refer to them. There are two common types of items you can name:

- **Variables**, which are items that can contain different values at different times during script execution, start with a dollar sign (`$`) followed by a name that you give them.
- **Constants**, which will contain the same value throughout the execution of your script, are by convention, all uppercase names that you give them. For example, `NULL` is a constant.

The name that you give to either variables or constants (or any other label in PHP) is case sensitive; can begin with either the letters *a–z* or *A–Z*, or an underscore (_); can be of any length; and can contain letters, numbers, underscores, and the characters in Western European alphabets. While you may find that some special characters will be allowed, the best practice is to not use them.

PHP keywords with specific meanings that you should not use for naming variables, constants, or other labels in PHP are shown in Table 5-2.

A large number of predefined constants in PHP are used for a variety of purposes, like true constants (`M_PI` = 3.1415926535898 and `M_E` = 2.718281828459), constants used with particular functions (`SORT_NUMERIC` and `SORT_STRING` used with sort functions), constants used to define something (`CAL_GREGORIAN` and `CAL_JULIAN` used with calendars), and constants used for formatting (`DATE_ATOM` and `DATE_RFC822` used with the date functions).

abstract	and	array	as	bool	break
callable	case	catch	class	clone	const
continue	declare	default	die	do	echo
else	elseif	empty	enddeclare	endfor	endforeach
endif	endswitch	endwhile	eval	exit	extends
false	final	finally	float	for	foreach
function	global	goto	if	implements	int
include	include_once	instanceof	insteadof	interface	isset
list	namespace	new	null	object	or
print	private	protected	public	require	require_once
return	static	string	switch	throw	trait
true	try	unset	use	var	while
xor	yield				

Table 5-2 PHP Reserved Keywords

You can find a list of predefined constants within each major section of predefined functions in the PHP Manual – Function Reference at php.net/manual/en/funcref.php.

A good practice is to make variable and constant names more self-descriptive and in the process, stay away from any possibility of conflict with a predefined name. For example, if you are collecting a buyer's name and address, you might be tempted to use `$name`, `$street`, and `$city`. While nothing is wrong with those names, it is a better practice to get in the habit of using compound names that are both more descriptive and stay away from common names, for example, `$buyer_name`, `$buyer_street`, and `$buyer_city`. It may take a couple of seconds more to type these names, but they are not going to be confused with similar names. Also, most programmers get really good at cutting and pasting to reduce typing.

TIP

If you are having trouble finding a bug in a program, look at the names you have assigned variables and constants, and try changing any that could possibly have a conflict with other names, such as changing `$name` to `$buyer_name`.

Operators

Having created a variable or a constant, you are going to want to assign it a value. PHP has defined a number of operators of various types to do this, as shown in Table 5-3.

NOTE

Another set of comparison operators, called "ternary operators," will be discussed in the next chapter.

If you combine several operators in a single expression, the order of precedence is as follows, beginning with the highest or first executed: `++`, `--`, `!`, `~`, `@`, `*`, `/`, `%`, `+`, `-`, `.`, `<<`, `>>`, `<`, `<=`, `>`, `>=`, `<>`, `==`, `!=`, `===`, `!==`, `&`, `^`, `|`, `&&`, `||`, `=`, `+=`, `-=`, `.=`, `and`, `xor`, `or`. You can use parentheses to get around the order of precedence. Other notes about the PHP operators include the following:

- The modulus (`%`) does not give the percent the first number is of the second; rather, it gives the remainder, that part that is left after whole division.
- The equal sign (`=`) does not mean "equal"; it means "assign" or "replace." For comparisons such as "is a equal to b," use the double equal sign (`==`).
- If $a = 2 and $b = 2.0, they are not identical because one is an integer and the other is a floating point number, but they are numerically equal.

Type of Operator	Name	Example	Explanation
Arithmetic			Performs arithmetic
+	Add	`$a + $b`	Sum
-	Subtract	`$a - $b`	Difference
*	Multiply	`$a * $b`	Product
/	Divide	`$a / $b`	Quotient
%	Modulus	`$a % $b`	Remainder
Assignment			Replaces a value with another
=	Assign	`$a = 7;`	Sets `$a` to 7

+=	Increment	`$a = 7;`
		`$a += 2;`
		`returns 9`
			Increments `$a` by 2

.=	Concatenate	`$a = "Joe ";`
		`$a .= "Blow";`
		`returns "Joe Blow"`
			Adds a string to an existing string

[]=	Append	`$array []=` `$something`	Appends `$something` to the end of the array
Bitwise			Turns specific bits in an integer on or off
&	And	`$a & $b`	Sets bits in both `$a` and `$b`
|	Or	`$a | $b`	Sets bits in either `$a` or `$b`
^	Xor	`$a ^ $b`	Sets bits in `$a` or `$b`, but not both
~	Not	`~$a`	Does not sets bits in `$a`
<<	Shift left	`$a << $b`	Shifts bits `$a` by `$b` steps to the left (each step is multiplying by 2)
>>	Shift right	`$a >> $b`	Shifts bits in `$a` by `$b` steps to the right (each step is dividing by 2)
Comparison			Compares two values
==	Equal	`$a == $b`	Returns TRUE if `$a` equals `$b`
===	Identical	`$a === $b`	Returns TRUE if `$a` is identical to `$b`
!= or <>	Not equal	`$a != $b or` `$a <> $b`	Returns TRUE if `$a` is not equal to $b
!==	Not Identical	`$a !== $b`	Returns TRUE if `$a` is not identical to $b
<	Less than	`$a < $b`	Returns TRUE if `$a` is less than `$b`
>	Greater than	`$a > $b`	Returns TRUE if `$a` greater than `$b`

Table 5-3 PHP Operators *(continued)*

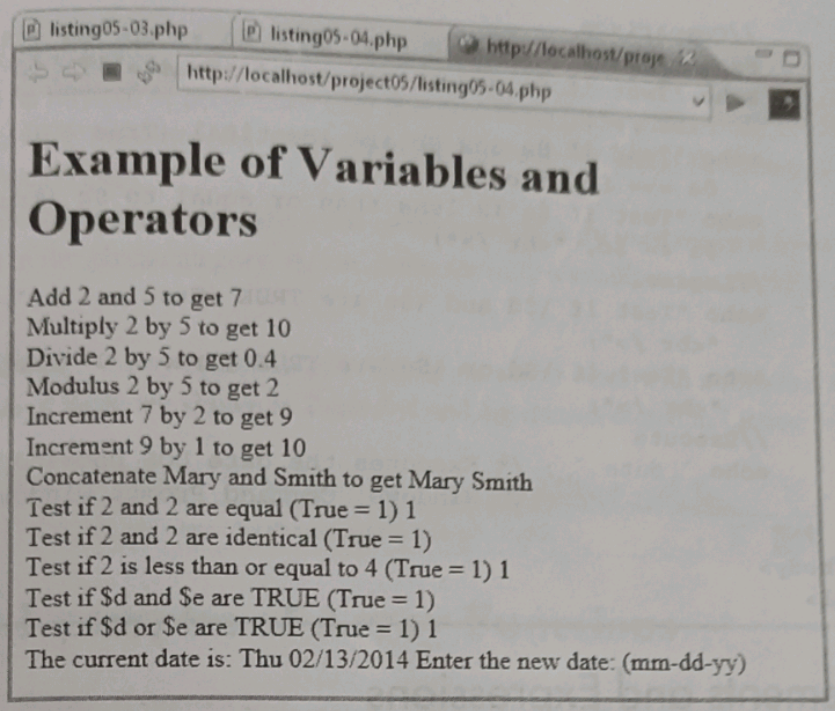
Type of Operator	Name	Example	Explanation
<=	Less than or equal to	`$a <= $b`	Returns TRUE if `$a` is less than or equal to `$b`
>=	Greater than or equal to	`$a >= $b`	Returns TRUE if `$a` is greater than or equal to `$b`
Increment			Changes the value by one
++	Increment	`++$a` `$a++`	Adds one to `$a` and returns `$a` Returns `$a`, then adds one to it
--	Decrement	`--$a` `$a--`	Subtracts one from $a and returns `$a` Returns `$a`, then subtracts one from it
Logical			Logical consequence
and &&	And	`$a and $b`	Returns TRUE if both `$a` and `$b` are TRUE
or ||	Or	`$a or $b`	Returns TRUE if either `$a` or `$b` is TRUE
xor	Xor	`$a xor $b`	Returns TRUE if either `$a` or `$b` is TRUE, but not both
!	Not	`!$a`	Returns TRUE if `$a` is not TRUE
Other			
@	Error off	`@my_function()`	Disables display of errors in expression
` `	Execute	`` ` date ` ``	Executes what is within the back ticks (these are not single quotes), like DOS commands

Table 5-3 PHP Operators

Listing 5-4 shows examples of the use of PHP variables and operators. The results this script returns are shown on the following page.

NOTE

The variable names in the following script, for example, `$a` or `$b`, will be replaced with the value contained in the variable when they are displayed with `echo` or `print`, unless you escape them (treat them as literal characters) by putting a backslash (\) in front of the variable. This is true *both* inside double quotes as well as outside of any quotation marks, and is called *interpolation*. If you use single quotes to enclose a variable name, the name, not the contents, will be displayed.



Listing 5-4 Examples of Variables and Operators

```
<html>
   <head>
      <title>Listing 5-4</title>
   </head>
   <body>
      <h1>Example of Variables and Operators</h1>
      <?php
         //Math
         $a = 2; $b = 5;
         echo "Add $a and $b to get  ", $a + $b, "<br />";
         echo "Multiply $a by $b to get  ", $a * $b, "<br />";
         echo "Divide  $a by $b to get  ", $a / $b, "<br />";
         echo "Modulus  $a by $b to get  ", $a % $b, "<br />";
         //Increment and Concatenate
         $a = 7;
         echo "Increment $a by 2 to get ", $a += 2, "<br />";
         echo "Increment $a by 1 to get ", ++$a, "<br />";
            $first_name = "Mary "; $last_name = "Smith";
         echo "Concatenate $first_name and $last_name to get  ",
            $first_name . $last_name, "<br />";
```

```
        //Comparison
        $a = "2"; $b = 2.0; $c = 4; $d = $a == $b; $e = $a === $b;
        echo "Test if $a and $b are equal (True = 1)  ", $a == $b,
           "<br />";
        echo "Test if $a and $b are identical (True = 1)  ",
           $a === $b, "<br />";
        echo "Test if $a is less than or equal to $c (True = 1)  ",
           $a <= $c, "<br />";
        //Logical
        echo "Test if \$d and \$e are TRUE (True = 1)  ", $d and $e,
           "<br />";
        echo "Test if \$d or \$e are TRUE (True = 1)  ", $d or $e,
           "<br />";
        //Execute
        echo ` date ` ; /* Executes the date DOS command as you would
                           in Windows' Command Prompt window. */
    ?>
  </body>
</html>
```

Statements and Expressions

PHP scripts contain either comments or statements. A *statement* is anything that is in between semicolons or the opening and closing PHP tags. Often, a statement is a single line of code ending in a semicolon, but you can have several statements on a single line, and you can have statements that take several lines. Most statements contain one or more expressions, but a few are only a single keyword, such as `break` or `else`.

Expressions are anything that has a value or evaluates to a value. *Values* are anything that can be assigned to a variable, so values can be any of the data types: integer, floating point, string, Boolean, array, or object. While the `NULL` data type is the absence of a value, it is still considered a value for this discussion.

Expressions can contain expressions, or said another way, expressions are building blocks that can be used to build other expressions. For example, `$a = 2` is three expressions, `2`, `$a`, and `$a = 2`.

Functions

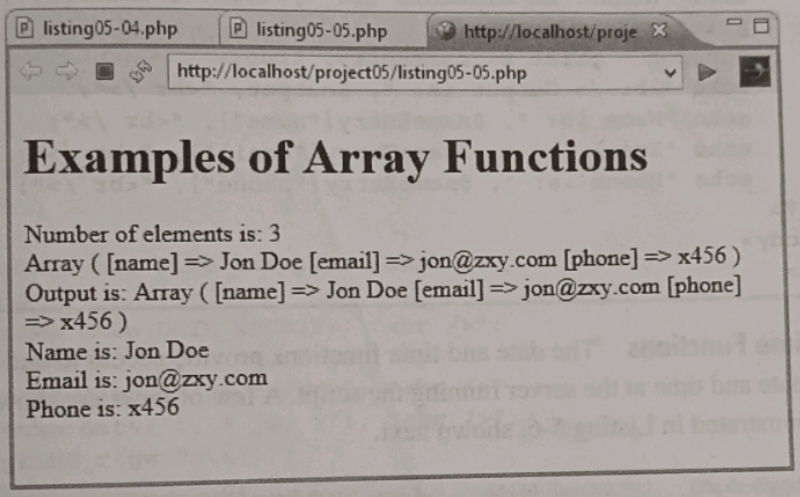
A *function* is a piece of script that does something and can be repeatedly called within a larger script. Some internal functions already exist, and some user-defined functions you write. Some functions require *arguments*, which are values that you pass to the function, which uses them to compute a return value. Other functions simply return a value when they are called.

Internal Functions

You can use an internal or predefined PHP function to do many different tasks. To explore the full set of PHP functions, see the online PHP Manual - Function Reference at php.net/manual/en/funcref.php.

The following sections describe a *few* of the more heavily used internal functions as examples in the given category. Again, these are only a small sample of the available functions.

Array Functions Array functions provide the means to work with arrays in a number of ways. Several of these are shown in Table 5-4 and are demonstrated in Listing 5-5.



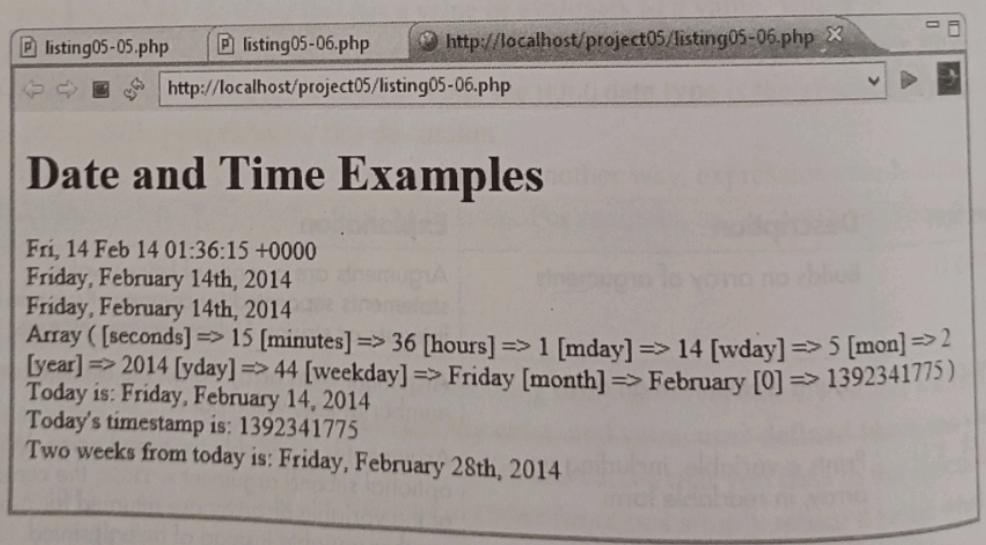
Function	Description	Explanation
`array()`	Builds an array of arguments	Arguments are a series of key => value statements separated by commas. Keys can be integers or strings. Values can be any type.
`count()`	Counts elements in an array	Argument is an array name; the elements are the number of key/value pairs.
`print_r()`	Prints a variable, including an array, in readable form	Argument is a variable (or array) name. With an optional second argument = `TRUE`, the contents of the variable or array are returned for use by another variable instead of being printed.

Table 5-4 Several Array Functions

Listing 5-5 Examples of Array Functions

```
<html>
   <head>
      <title>Listing 5-5</title>
   </head>
   <body>
      <h1>Examples of Array Functions</h1>
      <?php
         $nameEntry = array(
            "name" => "Jon Doe", "email" => "jon@zxy.com",
            "phone" => "x456" );
         echo "Number of elements is: ", count($nameEntry), "<br />";
         print_r($nameEntry);
         $output = print_r($nameEntry, true);
         echo "<br /> Output is: ", $output, "<br />";
         echo "Name is: ", $nameEntry["name"], "<br />";
         echo "Email is: ", $nameEntry["email"], "<br />";
         echo "Phone is: ", $nameEntry["phone"], "<br />";
      ?>
   </body>
</html>
```

Date/Time Functions The date and time functions provide access to and work with the current date and time at the server running the script. A few of these are shown in Table 5-5 and demonstrated in Listing 5-6, shown next.



Function	Description	Explanation
`date()`	Formats a date and time as a string	Requires a format string and optionally a timestamp if the local time is not desired
`getdate()`	An array of the date and time components	Takes an optional timestamp if the local time is not desired
`time()`	Current "Unix" timestamp	Returns the number of seconds since 1-1-1970, 00:00:00 GMT to the current moment at your server

Table 5-5 Some Date and Time Functions

Listing 5-6 Examples of Date and Time Functions

```
<html>
   <head>
   <title>Listing 5-6</title>
   </head>
   <body>
      <h1>Date and Time Examples</h1>
      <?php
         echo date(DATE_RFC822), "<br />";
         $date_format = "l, F jS, Y";
         echo date($date_format), "<br />";
         echo date("l, F jS, Y"), "<br />";
         print_r(getdate());
         $date_array = getdate();
         echo "<br />Today is: ", $date_array["weekday"],
            ", ", $date_array["month"], " ", $date_array["mday"],
            ", ", $date_array["year"], "<br />";
         echo "Today's timestamp is: ", time(), "<br />";
            //Seconds since 1-1-1970
         $twoweeks = (14 * 24 * 60 * 60);
            //seconds in 2 weeks
         $twoweeksFrNow = Time() + $twoweeks;
         echo "Two weeks from today is: ",
            date("l, F jS, Y", $twoweeksFrNow), "<br />";
      ?>
   </body>
</html>
```

NOTE

In the results of Listing 5-6, you see that **echo date**(DATE_RFC822), "**
**"; ends with +0000. This says that there is no offset (0000) from GMT (Greenwich Mean Time), which happens because I have not changed the default in Zend Apache server. PHP version 5.1.0 and later checks to see if a valid time zone has been set and will issue a warning if it hasn't. Most Internet hosting services set this, so if you want to use your user's time zone, you're okay. To be sure, though, it is a good idea to set your own time zone in each script (once per script is adequate) using the `date_default_timezone_set()` function with a local time zone constant (for example, mine is "America/Los_Angeles" for the Pacific time zone). You can find a list of supported time zone constants at php.net/manual/en/timezones.php (not every major city is included; you must look for cities in your time zone).

When working with dates, there are a large number of constants, formats, and functions within PHP that you can work with. Listing 5-6 demonstrates a couple of these:

- The `date()` function requires, at a minimum, a format string. The format string can be a predefined date/time constant, such as `DATE_RFC822`; a literal string, such as "l, F jS, Y" that uses format codes for the various parts of a date and time; or a variable with the format codes.
- Eleven date/time constants provide predefined formatting for the date and time. Four of the more common ones are shown next. You can see all 11 at php.net/manual/en/class.datetime.php.

Date and Time Constants

Atom: 2014-02-16T18:35:23+00:00
RFC822: Sun, 16 Feb 14 18:35:23 +0000
RFC850: Sunday, 16-Feb-14 18:35:23 UTC
RSS: Sun, 16 Feb 2014 18:35:23 +0000

- Thirty-five formatting codes can be used to format a date and time in PHP. The codes are case sensitive—a "d" and a "D" mean different formats. Here are four commonly used sets of formatting codes, which are explained in Table 5-6. You can see all of the PHP formatting codes at php.net/manual/en/function.date.php.

Date and Time Formatting Codes

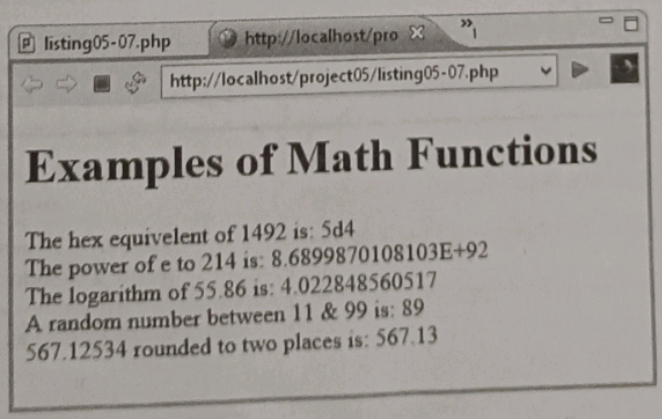
l, F jS, Y : Sunday, February 16th, 2014
D, M d, y : Sun, Feb 16, 14
n/j/y g:i:s a : 2/16/14 6:35:23 pm
Y-m-d G:i:s : 2014-02-16 18:35:23

- Within the literal format string (within the quote marks), you can include separator characters such as **, - / .** and **:**, as you can see in the previous illustration.

Code	Explanation
D	Three-character day of the week (Mon – Sun)
d	Two-character day of the month with a leading zero (01 – 31)
j	One- or two-character day of the month (1 – 31)
l	Full text day of the week (Sunday – Saturday)
S	Suffix for day of the month, used with code j (st, nd, rd, th)
F	Full text month (January – December)
M	Three-character month (Jan – Dec)
m	Two-character numeric month with a leading zero (01 – 12)
n	One- or two-character numeric month (1 – 12)
Y	Four-digit year (for example, 1986 or 2014)
y	Two-digit year (for example, 86 or 14)
a	am or pm
A	AM or PM
g	One- or two-digit hours in 12-hour format (1 – 12)
G	One- or two-digit hours in 24-hour format (0 – 23
h	Two-digit hours in 12-hour format with leading zeros (01 – 12)
H	Two-digit hours in 24-hour format with leading zeros (00 – 23)
i	Two-digit minutes with leading zeros (00 – 59)
s	Two-digit seconds with leading zeros (00 – 59)

Table 5-6 Explanation of Some of the Date/Time Formatting Codes

Math Functions Math functions allow you to perform mathematical routines. Some of these are shown in Table 5-7 and are demonstrated in Listing 5-7.



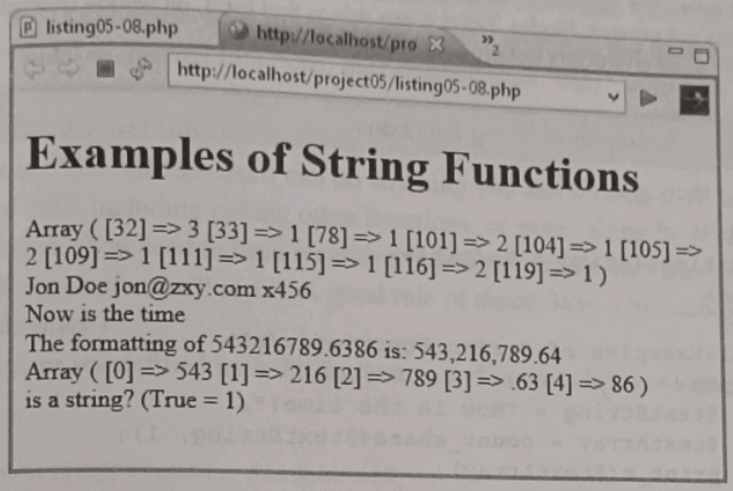
Function	Description	Explanation
`dechex()`	Converts a decimal integer to a hexadecimal number	Returns a string representing the hexadecimal number.
`exp()`	Calculates powers of e	Returns a floating point number of e to the power of the floating point argument.
`log()`	Calculates the natural logarithm	Returns a floating point number of the logarithm of the floating point argument. An optional argument can specify a different base other than the default of e.
`rand()`	Generates a random number	Without an argument, this generates an integer between 0 and 32,768. Optionally, minimum and maximum integers may be specified, separated by a comma.
`round()`	Rounds a number to number of decimal digits	Returns a floating point number after arithmetically rounding a floating point number. Optionally, the precision or number of decimal digits may be specified.

Table 5-7 A Few Math Functions

Listing 5-7 Examples of Math Functions

```
<html>
   <head>
      <title>Listing 5-7</title>
   </head>
   <body>
      <h1>Examples of Math Functions</h1>
      <?php
         echo "The hex equivalent of 1492 is: ", dechex(1492), "<br />";
         echo "The power of e to 214 is: ", exp(214), "<br />";
         echo "The logarithm of 55.86 is: ", log(55.86), "<br />";
         echo "A random number between 11 & 99 is: ", rand(11,99),
            "<br />";
         echo "567.12534 rounded to two places is: ", round(567.12534, 2),
            "<br />";
      ?>
   </body>
</html>
```

String Functions String functions allow you to work with strings. You have already seen two string functions, `echo()` and `print()`. Several others are shown in Table 5-8, demonstrated in Listing 5-8, and shown next.



Function	Description	Explanation
`count_chars()`	Counts occurrence of characters in a string	Arguments are the string and optionally a mode, which determines if the return is an array or a string and what is to be counted. See Note following the illustration.
`implode()` or `join()`	Joins array elements to form a string	Arguments are the array and optionally the character, which defaults to a space, to be placed between elements.
`is_string()`	Determines if a value is a string	Argument is a variable, which is evaluated and returns TRUE if it is a string.
`ltrim()`	String formed by removing white space and specified characters on left	Arguments are the string and optionally a list of characters to remove. Without a list, spaces, tabs, carriage returns, newlines, and vertical tabs are removed.
`number_format()`	String formed by formatting a number with a decimal point and thousands separator	A floating point number is formatted with an optional integer number of decimal digits (1, 2, or 4, not 3). Optionally, the decimal point and thousands separator characters can be specified.
`str_split()`	Splits a string into array elements	Arguments are the string and optionally the number of characters in each element, with a default of 1.
`strval()`	Converts a numeric variable to a string	Argument must be an integer or floating point number.

Table 5-8 Examples of String Functions

NOTE

The `count_chars()` function has five modes (0 through 4). Modes 0 through 2 are arrays that return the count of 0, all 255 characters; 1, all characters present; and 2, all characters not present. Modes 3 and 4 are strings that list 3, all unique characters present; and 4, all characters not present.

Listing 5-8 Examples of String Functions

```
<html>
   <head>
      <title>Listing 5-8</title>
   </head>
   <body>
      <h1>Examples of String Functions</h1>
      <?php
         $textString = "Now is the time!";
         $textArray = count_chars($textString, 1);
         print_r($textArray);
         $nameEntry = array(
            "name" => "Jon Doe", "email" => "jon@zxy.com",
            "phone" => "x456"
            );
         echo "<br />", implode("  ",$nameEntry), "<br />";
         $aLongString = "          Now is the time";
         echo ltrim($aLongString), "<br />";
         echo "The formatting of 543216789.6386 is: ",
            number_format(543216789.6386, 2), "<br />";
         print_r(str_split(543216789.6386, 3)) ;
         echo "<br />", $newString, "  is a string? (True = 1) ",
            is_string($newString) , "<br />";
      ?>
   </body>
</html>
```

NOTE

In the next chapter, you'll see a much better display of `count_chars()`.

User-Defined Functions

As you write PHP scripts, you'll often find that you want to repeatedly use the same code. You could simply copy the code to all the places you want to use it, but if you want to change that code, you would have to change it everywhere it was copied. The solution for this is to create a function containing the code you want to repeat and simply call that function everywhere you want to use it. Functions also let you segment or compartmentalize your code, making it easier to debug and maintain.