CHAPTER 3

Installing and Configuring PHP

In the last of the three installation-related chapters, you will acquire, install, and configure PHP and make some basic changes to your Apache installation. In this chapter, you will learn

- How to install PHP with Apache on Linux/Unix
- How to install PHP with Apache server on Windows
- How to test your PHP installation
- How to find help when things go wrong
- The basics of the PHP language

Current and Future Versions of PHP

The installation instructions in this chapter refer to PHP version 4.3.3, which is the current version of the software. The PHP Group uses minor release numbers for updates containing security enhancements or bug fixes. Minor releases do not follow a set release schedule; when enhancements or fixes are added to the code and thoroughly tested, the PHP Group will releases a new version, with a new minor version number.

It is possible that by the time you purchase this book, the minor version number will have changed, to 4.3.4 or beyond. If that is the case, you should read the list of changes at `http://www.php.net/ChangeLog-4.php` for any changes regarding the installation or configuration process, which makes up the bulk of this chapter.

Although it is unlikely that any installation instructions will change between minor version updates, you should get in the habit of always checking the changelog of software that you install and maintain. If a minor version change does occur during the time you are reading this book, but no installation changes are noted in the

changelog, simply make a mental note and substitute the new version number wherever it appears in the installation instructions and accompanying figures.

By the Way

For instructions on installing PHP from the CD, please refer to Appendix A, "Installing MySQL, Apache, and PHP from the CD-ROM."

Building PHP on Linux/Unix with Apache

In this section, we will look at one way of installing PHP with Apache on Linux/Unix. The process is more or less the same for any Unix operating system. Although you might be able to find pre-built versions of PHP for your system, compiling PHP from the source gives you greater control.

To download the PHP distribution files, go to the home of PHP, `http://www.php.net/`, and follow the link to the Downloads section. Grab the latest version of the source code—for this example, we are using 4.3.3. Your distribution will be named something similar to `php-`*`version`*`.tar.gz`, where *`version`* is the most recent release number. This archive will be a compressed tar file, so you will need to unpack it:

```
# gunzip < php-version.tar.gz ¦ tar xvf –
```

Keep the downloaded file in a directory reserved for source files, such as `/usr/src/` or `/usr/local/src/`. After your distribution is unpacked, you should move to the PHP distribution directory:

```
# cd php-version
```

Within your distribution directory, you will find a script called `configure`. This script accepts additional information that is provided when the `configure` script is run from the command line. These command-line arguments control the features that PHP will support. In this example, we will include the basic options you need to use to install PHP with Apache and MySQL support. We will discuss some of the available `configure` options later in the chapter, and throughout the book as they become relevant.

```
# ./configure  --prefix=/usr/local/php --with-mysql \
--with-apxs2=/usr/local/apache2/bin/apxs
```

Once the `configure` script has run, you will be returned to the prompt after receiving informational notes from the PHP Group:

```
+--------------------------------------------------------------------+
| License:                                                           |
| This software is subject to the PHP License, available in this     |
| distribution in the file LICENSE.  By continuing this installation |
| process, you are bound by the terms of this license agreement.     |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.                            |
+--------------------------------------------------------------------+
|                          *** NOTE ***                              |
|            The default for register_globals is now OFF!            |
|                                                                    |
| If your application relies on register_globals being ON, you       |
| should explicitly set it to on in your php.ini file.               |
| Note that you are strongly encouraged to read                      |
| http://www.php.net/manual/en/security.registerglobals.php          |
| about the implications of having register_globals set to on, and   |
| avoid using it if possible.                                        |
+--------------------------------------------------------------------+

Thank you for using PHP.
```

Next, issue the `make` command, followed by the `make install` command. These commands should end the process of PHP compilation and installation and return you to your prompt.

You will need to ensure that two very important files are copied to their correct locations. First, issue the following command to copy the distributed version of `php.ini` to its default location. You will learn more about `php.ini` later in this chapter.

```
# cp php.ini-dist /usr/local/lib/php.ini
```

Next, copy the PHP shared object file to its proper place in the Apache installation directory, if it has not already been placed there by the installation process:

```
# cp libs/libphp4.so /usr/local/apache2/modules/
```

You should now be able to configure and run Apache, but let's cover some additional configuration options before heading on to the "Integrating PHP with Apache on Linux/Unix" section.

Additional Configuration Options

When we ran the `configure` script, we included some command-line arguments that determined some features that the PHP engine will include. The `configure` script itself gives you a list of available options, including the ones we used. From the PHP distribution directory, type the following:

```
# ./configure  --help
```

This produces a long list, so you might want to add it to a file and read it at your leisure:

```
# ./configure  --help  > configoptions.txt
```

If you discover additional functionality you wish to add to PHP after it has been installed, simply run the configuration and build process again. Doing so will create a new version of `libphp4.so` and place it in the Apache directory structure. All you have to do is restart Apache in order for the new file to be loaded.

Integrating PHP with Apache on Linux/Unix

To ensure that PHP and Apache get along with one another, you need to check for—and potentially add—a few items to the `httpd.conf` configuration file. First, look for a line like the following:

```
LoadModule php4_module        modules/libphp4.so
```

If this line is not present, or only appears with a pound sign (`#`) at the beginning of the line, you must add the line or remove the `#`. This line tells Apache to use the PHP shared object file that was created by the PHP build process (`libphp4.so`).

Next, look for this section:

```
#
# AddType allows you to add to or override the MIME configuration
# file mime.types for specific file types.
#
```

Add the following line:

```
AddType application/x-httpd-php .php .phtml .html
```

This ensures that the PHP engine will parse files that end with the `.php`, `.phtml`, and `.html` extensions. Your selection of filenames might differ, and you might want to add `.php3` as an extension, for backward compatibility with any very old scripts you may have.

Save this file, and then restart Apache. When you look in your `error_log`, you should see something like the following line:

```
[Sun Sep 28 10:42:47 2002] [notice] Apache/2.0.47 (Unix) PHP/4.3.3 configured
```

PHP is now part of the Apache Web server. If you want to learn how to install PHP on a Windows platform, keep reading. Otherwise, you can skip ahead to the "Testing Your Installation" section.

Installing PHP Files on Windows

Unlike building and installing PHP on Linux/Unix, installing PHP on Windows requires nothing more than downloading the distribution and moving a few files around. To download the PHP distribution files, go to the home of PHP, `http://www.php.net/`, and follow the link to the Downloads section. Grab the latest version of the zip package (*not* the installer!) from the Windows Binaries section—for this example, we are using 4.3.3. Your distribution will be named something like to `php-version.zip`, where *`version`* is the most recent release number.

Once the file is downloaded to your system, double-click on it to launch your unzipping software. The distribution is packed up with pathnames already in place, so extract the files to the root of your drive, where it will create a directory called `php-version-Win32`, and place all the files and subdirectories under that new directory.

Now that you have all the basic PHP distribution files, you just need to move a few of them around:

1. In the PHP installation directory, find the `php.ini-dist` file and rename it `php.ini`.
2. Copy the `php.ini` file to `C:\WINDOWS\` or wherever you usually put your `*.ini` files.
3. Copy the `php4ts.dll` file to `C:\WINDOWS\SYSTEM\` or wherever you usually put your `*.dll` files.

To get a basic version of PHP working with Apache, you'll need to make a few minor modifications to the Apache configuration file.

Integrating PHP with Apache on Windows

To ensure that PHP and Apache get along with one another, you need to add a few items to the `httpd.conf` configuration file in the Apache directory on your computer. First, find a section that looks like this:

```
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule access_module modules/mod_access.so
LoadModule actions_module modules/mod_actions.so
LoadModule alias_module modules/mod_alias.so
LoadModule asis_module modules/mod_asis.so
LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
```

```
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so
#LoadModule dav_module modules/mod_dav.so
#LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
#LoadModule file_cache_module modules/mod_file_cache.so
#LoadModule headers_module modules/mod_headers.so
```

At the end of this section, add the following:

```
LoadModule php4_module c:/php-version/sapi/php4apache2.dll
```

Next, look for this section:

```
#
# AddType allows you to add to or override the MIME configuration
# file mime.types for specific file types.
#
```

Add the following lines:

```
AddType application/x-httpd-php .php .phtml .html
```

This ensures that the PHP engine will parse files that end with the `.php`, `.phtml`, and `.html` extensions. Your selection of filenames might differ, and you might want to add `.php3` as an extension, for backward compatibility with any very old scripts you might have.

Save this file, and then restart Apache. The server should start without warning, PHP is now part of the Apache Web server.

`php.ini` Basics

After you have compiled or installed PHP, you can still change its behavior with the `php.ini` file. On Unix systems, the default location for this file is `/usr/local/php/lib`, or the `lib` subdirectory of the PHP installation location you used at configuration time. On a Windows system, this file should be in the Windows directory.

Directives in the `php.ini` file come in two forms: values and flags. Value directives take the form of a directive name and a value separated by an equal sign. Possible values vary from directive to directive. Flag directives take the form of a

directive name and a positive or negative term separated by an equal sign. Positive terms include `1`, `On`, `Yes`, and `True`. Negative terms include `0`, `Off`, `No`, and `False`. Whitespace is ignored.

You can change your `php.ini` settings at any time, but after you do, you'll need to restart the server for the changes to take effect. At some point, take time to read through the `php.ini` file on your own to see the types of things that can be configured.

Testing Your Installation

The simplest way to test your PHP installation is to create a small test script utilizing the `phpinfo()` function. This function will produce a long list of configuration information. Open a text editor and type the following line:

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

Save this file as `phpinfo.php` and place it in the document root of your Web server—the `htdocs` subdirectory of your Apache installation. Access this file via your Web browser and you should see something like Figure 3.1 or Figure 3.2.

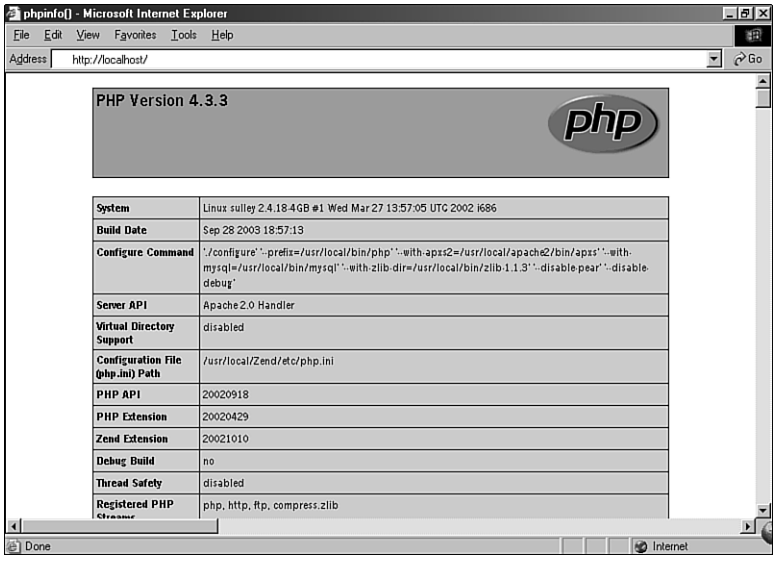

FIGURE 3.1
The results of `phpinfo()` on a Linux/Unix system.

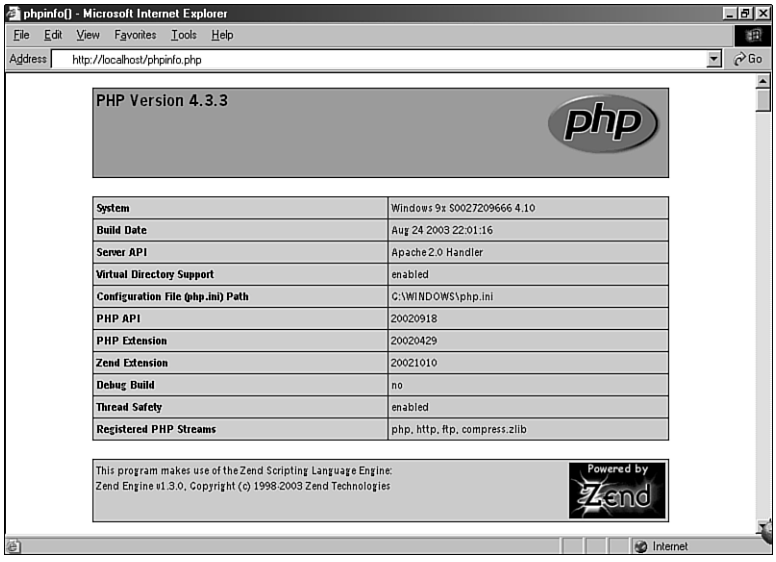

FIGURE 3.2
The results of `phpinfo()` on a Windows system.

Getting Installation Help

Help is always at hand on the Internet, particularly for problems concerning open source software. Wait a moment before you click the send button, however. No matter how intractable your installation, configuration, or programming problem might seem, chances are you are not alone. Someone has probably already answered your question.

When you hit a brick wall, your first recourse should be to the official PHP site at `http://www.php.net/` (particularly the annotated manual at `http://www.php.net/manual/`).

If you still can't find your answer, don't forget that the PHP site is searchable. The advice you are seeking may be lurking in a press release or a Frequently Asked Questions file. You can also search the mailing list archives at `http://www.php.net/search.php`. These archives represent a huge information resource with contributions from many of the great minds in the PHP community. Spend some time trying out a few keyword combinations.

If you are still convinced that your problem has not been addressed, you might well be doing the PHP community a service by exposing it. You can join the PHP mailing lists at `http://www.php.net/support.php`. Although these lists often have high volume, you can learn a lot from them. If you are serious about PHP

scripting, you should certainly subscribe to at least a digest list. Once you've subscribed to the list that matches your concerns, you might consider posting your problem.

When you post a question, it is a good idea to include as much information as possible (without writing a novel). The following items are often pertinent:

- Your operating system
- The version of PHP you are running or installing
- The configuration options you chose
- Any output from the `configure` or `make` commands that preceded an installation failure
- A reasonably complete example of the code that is causing problems

Why all these cautions about posting a question to a mailing list? First, developing research skills will stand you in good stead. A good researcher can generally solve a problem quickly and efficiently. Posting a naive question to a technical list often results in a wait rewarded only by a message or two referring you to the archives where you should have begun your search for answers in the first place.

Second, remember that a mailing list is not analogous to a technical support call center. No one is paid to answer your questions. Despite this, you have access to an impressive pool of talent and knowledge, including that of some of the creators of PHP itself. A good question and its answer will be archived to help other coders. Asking a question that has been answered several times just adds more noise.

Having said this, don't be afraid to post a problem to the list. PHP developers are a civilized and helpful breed, and by bringing a problem to the attention of the community, you might be helping others to solve the same problem.

The Basics of PHP Scripts

Let's jump straight in with a PHP script. To begin, open your favorite text editor. Like HTML documents, PHP files are made up of plain text. You can create them with any text editor, such as Notepad on Windows, Simple Text and BBEdit on Mac OS, or vi and Emacs on Unix operating systems. Most popular HTML editors provide at least some support for PHP.

Did you Know?

Keith Edmunds maintains a handy list of PHP-friendly editors at `http://phpeditors.linuxbackup.co.uk/`.

Type in the example in Listing 3.1 and save the file, calling it something like `first.php`.

LISTING 3.1 A Simple PHP Script

```
1: <?php
2:     echo "Hello Web!";
3: ?>
```

If you are not working directly on the machine that will be serving your PHP script, you will probably need to use an FTP client such as WS-FTP for Windows or Fetch for Mac OS to upload your saved document to the server.

Once the document is in place, you should be able to access it via your browser. If all has gone well, you should see the script's output. Figure 3.3 shows the output from the `first.php` script.

FIGURE 3.3
Success: the output from Listing 3.1.

Beginning and Ending a Block of PHP Statements

When writing PHP, you need to inform the PHP engine that you want it to execute your commands. If you don't do this, the code you write will be mistaken for HTML and will be output to the browser. You can designate your code as PHP

with special tags that mark the beginning and end of PHP code blocks. Table 3.1 shows four such PHP delimiter tags.

TABLE 3.1 PHP Start and End Tags

Tag Style	Start Tag	End Tag
Standard tags	`<?php`	`?>`
Short tags	`<?`	`?>`
ASP tags	`<%`	`%>`
Script tags	`<SCRIPT LANGUAGE="php">`	`</SCRIPT>`

Of the tags in Table 3.1, only the standard and script tags are guaranteed to work on any configuration. The short and ASP-style tags must be explicitly enabled in your `php.ini`.

To activate recognition for short tags, you must make sure that the `short_open_tag` switch is set to `On` in `php.ini`:

```
short_open_tag  =  On;
```

Short tags are enabled by default, so you need to edit `php.ini` only if you want to disable them.

To activate recognition for the ASP-style tags, you must enable the `asp_tags` setting:

```
asp_tags = On;
```

After you have edited `php.ini`, you should be able to use any of the four styles in your scripts. This is largely a matter of preference, although if you intend to include XML in your script, you should disable the short tags (`<? ?>`) and work with the standard tags (`<?php ?>`).

Watch Out!

The character sequence `<?` tells an XML parser to expect a processing instruction and is therefore frequently included in XML documents. If you include XML in your script and have short tags enabled, the PHP engine is likely to confuse XML processing instructions and PHP start tags. Disable short tags if you intend to incorporate XML in your document.

Let's run through some of the ways in which you can legally write the code in Listing 3.1. You can use any of the four PHP start and end tags that you have seen:

```
<?
echo"Hello Web!";
?>

<?php
echo "Hello Web!";
?>

<%
echo "Hello Web!";
%>

<SCRIPT LANGUAGE="php">
echo "Hello Web!";
</SCRIPT>
```

You can also put single lines of code in PHP on the same line as the PHP start and end tags:

```
<? echo "Hello Web!"; ?>
```

Now that you know how to define a block of PHP code, let's take a closer look at the code in Listing 3.1 itself.

The `echo` Statement and `print()` Function

Simply, the `echo` statement is used to output data. In most cases, anything output by `echo` ends up viewable in the browser. You could also have used the `print()` function in place of the `echo` statement. Using `echo` or `print()` is a matter of taste; when you look at other people's scripts, you might see either use.

Whereas `echo` is a statement, `print()` is a function. A *function* is a command that performs an action, usually modified in some way by data provided for it. Data sent to a function is almost always placed in parentheses after the function name. In this case, you could have sent the `print()` function a collection of characters, or a *string*. Strings must always be enclosed in quotation marks, either single or double. For example:

```
<?
print("Hello Web!");
?>
```

The only line of code in Listing 3.1 ended with a semicolon. The semicolon informs the PHP engine that you have completed a statement.

A *statement* represents an instruction to the PHP engine. Broadly, it is to PHP what a sentence is to written or spoken English. A sentence should usually end with a period; a statement should usually end with a semicolon. Exceptions to this include statements that enclose other statements, and statements that end a block of code. In most cases, however, failure to end a statement with a semicolon will confuse the PHP engine and result in an error.

Combining HTML and PHP

The script in Listing 3.1 is pure PHP. You can incorporate this into an HTML document by simply adding HTML outside the PHP start and end tags, as shown in Listing 3.2.

LISTING 3.2 A PHP Script Incorporated into HTML

```
 1: <html>
 2: <head>
 3: <title>Listing 3.2 A PHP script including HTML</title>
 4: </head>
 5: <body>
 6: <b>
 7: <?php
 8:     echo "hello world";
 9: ?>
10: </b>
11: </body>
12: </html>
```

As you can see, incorporating PHP code into a predominantly HTML document is simply a matter of typing in the code. The PHP engine ignores everything outside the PHP open and close tags. If you were to view Listing 3.2 with a browser, as shown in Figure 3.4, you would see the string `hello world` in bold. If you were to view the document source, as shown in Figure 3.5, the listing would look exactly like a normal HTML document.

You can include as many blocks of PHP code as you need in a single document, interspersing them with HTML as required. Although you can have multiple blocks of code in a single document, they combine to form a single script. Any variables defined in the first block will usually be available to subsequent blocks.

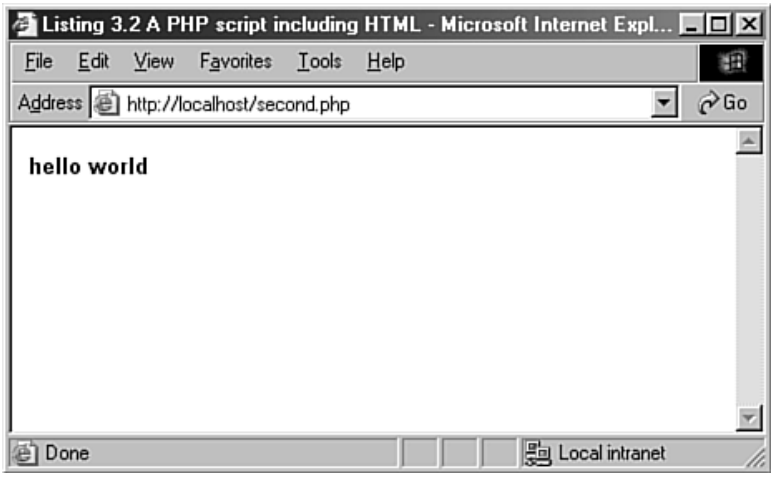

FIGURE 3.4
The output of Listing 3.2 as viewed in a browser.

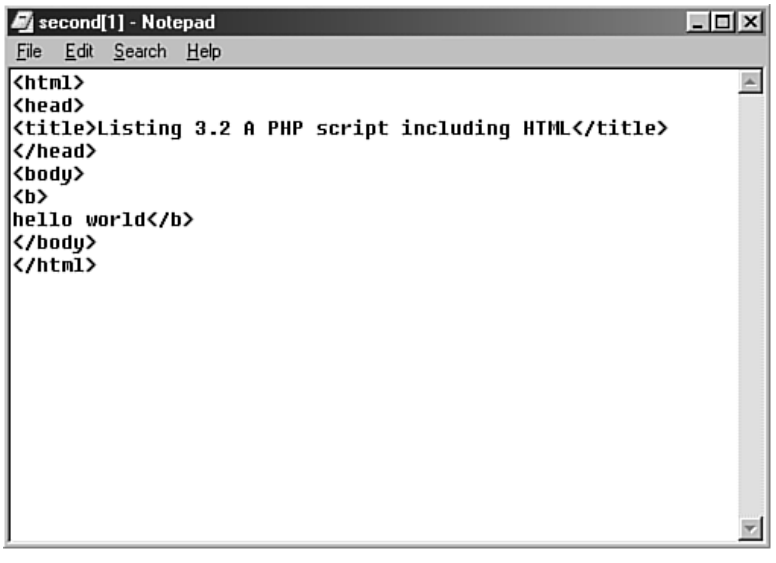

FIGURE 3.5
The output of Listing 3.2 as HTML source code.

Adding Comments to PHP Code

Code that seems clear at the time of writing can seem like a hopeless tangle when you try to amend it six months later. Adding comments to your code as you write can save you time later on and make it easier for other programmers to work with your code.

A *comment* is text in a script that is ignored by the PHP engine. Comments can be used to make code more readable or to annotate a script.

Single-line comments begin with two forward slashes (`//`) or a single hash sign (`#`). The PHP engine ignores all text between these marks and either the end of the line or the PHP close tag:

```
// this is a comment
#  this is another comment
```

Multiline comments begin with a forward slash followed by an asterisk (`/*`) and end with an asterisk followed by a forward slash (`*/`):

```
/*
this is a comment
none of this will
be parsed by the
PHP engine
*/
```

Summary

In this chapter, you learned how to install and configure PHP for use with Apache on either Linux/Unix or Windows. You learned that various `configure` options in the Linux/Unix build script can change the features that are supported. You learned about `php.ini` and how to change the values of its directives. Using the `phpinfo()` function, you tested your installation and produced a list of its configuration values. You created a simple PHP script using a text editor. You examined four sets of tags that you can use to begin and end blocks of PHP code. Finally, you learned how to use the `echo` statement or `print()` function to send data to the browser, and you brought HTML and PHP together into the same script. In the next chapter, you will use these skills to test some of the fundamental building blocks of the PHP language, including variables, data types, and operators.

Q&A

Q. ***You have covered an installation for Linux/Unix or Windows, and the Apache Web server. Does this mean that the material presented in this book will not apply to my server and operating system?***

A. No. One of PHP's great strengths is that it runs on multiple platforms. You can find installation instructions for different Web servers and configuration directives for database support in the PHP manual. Although the examples throughout this book are specifically geared toward the combination of PHP, MySQL, and Apache, only slight modifications would be needed to work with the examples using different Web servers or databases.

Q. ***Which are the best start and end tags to use?***

A. It is largely a matter of preference. For the sake of portability, the standard tags (`<?php ?>`) are probably the safest bet. Short tags are enabled by default and have the virtue of brevity, but with the increasing popularity of XML, it is safest to avoid them.

Q. ***What editors should I avoid when creating PHP code?***

A. Do not use word processors that format text for printing (such as Word, for example). Even if you save files created using this type of editor in plain text format, hidden characters are likely to creep into your code.

Q. ***When should I comment my code?***

A. Once again, this is a matter of preference. Some short scripts will be self-explanatory, even after a long interval. For scripts of any length or complexity, you should comment your code. This often saves you time and frustration in the long run.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1. From a Linux/Unix operating system, how would you get help on configuration options (the options that you pass to the `configure` script in your PHP distribution)?

2. What line should you add to the Apache configuration file to ensure that the `.php` extension is recognized?
3. What is PHP's configuration file called?
4. Can a person browsing your Web site read the source code of PHP script you have successfully installed?

Answers

1. You can get help on configuration options by calling the `configure` script in the PHP distribution folder and passing it the `--help` argument:

   ```
   ./configure --help
   ```

2. The line

   ```
   AddType application/x-httpd-php .php
   ```

   ensures that Apache will treat files ending with the `.php` extension as PHP scripts.
3. PHP's configuration file is called `php.ini`.
4. No, the user will see only the output of your script.

Activities

1. Install PHP on your system. If it is already in place, review your `php.ini` file and check your configuration.
2. Familiarize yourself with the process of creating, uploading, and running PHP scripts. In particular, create your own "hello world" script. Add HTML code to it, and add additional blocks of PHP. Experiment with the different PHP delimiter tags. Which ones are enabled in your configuration? Take a look at your `php.ini` file to confirm your findings. Don't forget to add some comments to your code.