Apress™
Books for Professionals by Professionals

Chapter One: "An Introduction to PHP"

A Programmer's Introduction to PHP 4.0

by William Jason Gilmore
ISBN # 1-893115-85-2

info@apress.com

CHAPTER 1

An Introduction to PHP

The past five years have been fantastic in terms of the explosive growth of the Internet and the new ways in which people are able to communicate with one another. Spearheading this phenomenon has been the World Wide Web (WWW), with thousands of new sites being launched daily and consumers being consistently offered numerous outstanding services via this new communications medium. With this exploding market has come a great need for new technologies and developers to learn these technologies. Chances are that if you are reading this paragraph, you are one of these Web developers or are soon to become one. Regardless of your profession, you've picked this book up because you've heard of the great new technology called *PHP*.

This chapter introduces the PHP language, discusses its history and capabilities, and provides the basic information you need to begin developing PHP-enabled sites. Several examples are provided throughout, hopefully serving to excite you about what PHP can offer you and your organization. You will learn how to install and configure the PHP software on both Linux/UNIX and Windows machines, and you will learn how to embed PHP in HTML. At the conclusion of the chapter, you will be ready to begin delving into the many important aspects of the PHP language. So light the fire, turn on your favorite jazz album, and curl up on the lazyboy; you are about to learn what will be one of the most exciting additions to your resume: PHP programming.

An Abbreviated History

PHP set its roots in 1995, when an independent software development contractor named Rasmus Lerdorf developed a Perl/CGI script that enabled him to know how many visitors were reading his online resume. His script performed two duties: logging visitor information and displaying the count of visitors to the Web page. Because the WWW as we know it today was still so young at that time, tools such as these were nonexistent, and they prompted emails inquiring about Lerdorf's scripts. Lerdorf thus began giving away his toolset, dubbed *Personal Home Page* (PHP), or *Hypertext Preprocessor*.

The clamor for the PHP toolset prompted Lerdorf to begin developing additions to PHP, one of which converted data entered in an HTML form into symbolic variables that allowed for their export to other systems. To accomplish this, he opted to continue development in C code rather than Perl. This addition to the

existing PHP toolset resulted in PHP 2.0, or PHP-FI (Personal Home Page—Form Interpreter). This 2.0 release was accompanied by a number of enhancements and improvements from programmers worldwide.

The new PHP release was extremely popular, and a core team of developers soon formed. They kept the original concept of incorporating code directly alongside HTML and rewrote the parsing engine, giving birth to PHP 3.0. By the 1997 release of version 3.0, over 50,000 users were using PHP to enhance their Web pages.

NOTE *1997 also saw the change of the words underlying the PHP abbreviation from Personal Home Page to Hypertext Preprocessor.*

Development continued at a hectic pace over the next two years, with hundreds of functions being added and the user count growing in leaps and bounds. At the onset of 1999, Netcraft (`http://www.netcraft.com`) reported a conservative estimate of a user base surpassing 1,000,000, making PHP one of the most popular scripting languages in the world.

Early 1999 saw the announcement of the upcoming PHP 4.0. Although one of PHP's strongest features was its proficiency at executing scripts, the developers had not intended that large-scale applications were going to be built using PHP. Thus they set out to build an even-more robust parsing engine, better known as Zend (`http://www.zend.com`). Development continued rapidly, culminating in the May 22, 2000, release of PHP 4.0.

In addition to the Zend processor, Zend technologies, based in Israel, offers the Zend optimizer, which increases even further the performance benefits of the Zend parsing engine. Available for download free of charge, the benchmarks have shown that the optimizer can result in a 40 to 100 percent overall performance gain. Check out the Zend site for more information.

At the time of this writing, according to Netcraft (`http://www.netcraft.com`), PHP is installed on over 3.6 million domains, making it one of the most popular scripting languages in the world. The future of PHP indeed looks bright, as major Web sites and personal users alike continue to embrace the product.

PHP is best summarized as an embedded server-side Web-scripting language that provides developers with the capability to quickly and efficiently build dynamic Web applications. PHP bears a close resemblance, both syntactically and grammatically, to the C programming language, although developers haven't been shy to integrate features from a multitude of languages, including Perl, Java, and C++. Several of these valuable borrowed features include regular expression parsing, powerful array-handling capabilities, an object-oriented methodology, and vast database support.

For writing applications that extend beyond the traditional, static methodology of Web page development (that is, HTML), PHP can also serve as a valuable tool for creating and managing dynamic content, embedded directly beside the

likes of JavaScript, Stylesheets, WML (Wireless Markup Language) and many other useful languages. Providing hundreds of predefined functions, PHP is capable of handling just about anything a developer can dream of. Extensive support is offered for graphic creation and manipulation, mathematical calculations, ecommerce, and burgeoning technologies such as Extensible Markup Language (XML), open database connectivity (ODBC), and Macromedia Shockwave. This vast range of capabilities eliminates the need for the tedious and costly integration of several third-party modules, making PHP the tool of choice for developers worldwide.

One of the main strengths of PHP is the fact that because it can be embedded directly alongside HTML code, there is no need to write a program that has many commands just to output the HTML. HTML and PHP can be used interchangeably as needed, working alongside one another in unison. With PHP, we can simply do the following:

```
<html>
<title><? print "Hello world!"; ?></title>
</html>
```

And `Hello world!` will be displayed in the Web page title bar. Interestingly, the single line print statement is enclosed in what are commonly known as PHP's escape characters (`<?…?>`) is a complete program. No need for lengthy prefacing code or inclusion of libraries; the only required code is what is needed to get the job done!

Of course, in order to execute a PHP script, you must first install and configure the PHP software on your server. This process is explained in "Downloading and Installing PHP/Apache," later in this chapter. Immediately preceding that section are a few excerpts from prominent users testifying to the power of PHP, followed by a detailed synopsis of the language and its history. However, before diving into the installation process, take a moment to read more about the characteristics of PHP that make it such a powerful language. This is the subject of the next section, aptly titled "Characteristics of PHP."

Characteristics of PHP

As you may have realized, the PHP language revolves around the central theme of practicality. PHP is about providing the programmer with the necessary tools to get the job done in a quick and efficient fashion. Five important characteristics make PHP's practical nature possible:

- Familiarity
- Simplicity

- Efficiency
- Security
- Flexibility

One final characteristic makes PHP particularly interesting: it's free!

Familiarity

Programmers from many backgrounds will find themselves already accustomed to the PHP language. Many of the language's constructs are borrowed from C and Perl, and in many cases PHP code is almost indistinguishable from that found in the typical C or Pascal program. This minimizes the learning curve considerably.

Simplicity

A PHP script can consist of 10,000 lines or one line: whatever you need to get the job done. There is no need to include libraries, special compilation directives, or anything of the sort. The PHP engine simply begins executing the code after the first escape sequence (`<?`) and continues until it passes the closing escape sequence (`?>`). If the code is syntactically correct, it will be executed exactly as it is displayed.

Efficiency

Efficiency is an extremely important consideration for working in a multiuser environment such as the WWW. PHP 4.0 introduced resource allocation mechanisms and more pronounced support for object-oriented programming, in addition to session management features. Reference counting has also been introduced in the latest version, eliminating unnecessary memory allocation.

Security

PHP provides developers and administrators with a flexible and efficient set of security safeguards. These safeguards can be divided into two frames of reference: system level and application level.

System-Level Security Safeguards

PHP furnishes a number of security mechanisms that administrators can manipulate, providing for the maximum amount of freedom and security when PHP is properly configured. PHP can be run in what is known as *safe mode*, which can

limit users' attempts to exploit the PHP implementation in many important ways. Limits can also be placed on maximum execution time and memory usage, which if not controlled can have adverse affects on server performance. Much as with a cgi-bin folder, administrators can also place restrictions on the locations in which users can view and execute PHP scripts and use PHP scripts to view guarded server information, such as the passwd file.

Application-Level Security Safeguards

Several trusted data encryption options are supported in PHP's predefined function set. PHP is also compatible with many third-party applications, allowing for easy-integration with secure ecommerce technologies. Another advantage is that the PHP source code is not viewable through the browser because the script is completely parsed before it is sent back to the requesting user. This benefit of PHP's server-side architecture prevents the loss of creative scripts to users at least knowledgeable enough to execute a 'View Source'.

Security is such an important issue that this book contains an entire chapter on the subject. Please read Chapter 16, "Security," for a thorough accounting of PHP's security features.

Flexibility

Because PHP is an embedded language, it is extremely flexible towards meeting the needs of the developer. Although PHP is generally touted as being used in conjunction solely with HTML, it can also be integrated alongside languages like JavaScript, WML, XML, and many others. Additionally, as with most other mainstream languages, wisely planned PHP applications can be easily expanded as needed.

Browser dependency is not an issue because PHP scripts are compiled entirely on the server side before being sent to the user. In fact, PHP scripts can be sent to just about any kind of device containing a browser, including cell phones, personal digital assistant (PDA) devices, pagers, laptops, not to mention the traditional PC. People who want to develop shell-based applications can also execute PHP from the command line.

Since PHP contains no server-specific code, users are not limited to a specific and perhaps unfamiliar Web server. Apache, Microsoft IIs, Netscape Enterprise Server, Stronghold, and Zeus are all fair game for PHP's server integration. Because of the various platforms that these servers operate on, PHP is largely platform independent, available for such platforms as UNIX, Solaris, FreeBSD, and Windows 95/98/NT.

Finally, PHP offers access to external components, such as Enterprise Java Beans and Win32 COM objects. These newly added features put PHP in the big league, truly enabling developers to scale PHP projects upward and outward as need be.

Free

The open source development strategy has gained considerable notoriety in the software industry. The prospect of releasing source code to the masses has resulted in undeniably positive outcomes for many projects, perhaps most notably Linux, although the success of the Apache project has certainly been a major contributor in proving the validity of the open source ideal. The same holds true for the developmental history of PHP, as users worldwide have been a huge factor in the advancement of the PHP project.

PHP's embracing of this open source strategy result in great performance gains for users, and the code is available free of charge. Additionally, an extremely receptive user community numbering in the thousands acts as "customer support," providing answers to even the most arcane questions in popular online discussion groups.

The next section, "User Affirmations," provides testimonies from three noted industry professionals. Each provides keen insight into why they find PHP such an appealing technology.

User Affirmations

"We have for a long time had a personal contact to some of the PHP developers and exchanged a lot of emails with them in the past. When the PHP developers have had any problems with MySQL related issues we have always been ready to help them solve them. We have also on some occasions added new features into MySQL just to get the PHP integration better. The result of this work is that MySQL works extremely well with PHP and we will ensure that it keeps that way!"

Michael "Monty" Widenius, MySQL Developer
`http://www.mysql.com`

"FAST used PHP to implement mp3.lycos.com for a number of reasons. The most important was time to market; PHP really lets you speed up the development. Another reason was speed, we went from 0 to 1.4 million page impressions in one day, and PHP coped just fine with this. The third reason was of course that I knew that if I found bugs in PHP during this '"stress test",' I could fix them myself since PHP is open source."

Stig Bakken, FAST Search & Transfer ASA
`http://www.fast.no`

"I've used PHP from the early days when it was PHP/FI 1.x. I loved having the ability to process forms and customize my pages on the fly with such an easy-to-use language. As my company's needs have evolved, so has PHP.

Today, PHP is extremely feature rich. We rely on it for just about every custom web site we develop, including 32bit.com and DevShed.com. We even use it at InfoWest to manage our customer service, account management and port monitoring.

PHP's evolution and acceptance is a textbook example of a successful open source project. Open-mindedness, community contribution, and a well-managed code-base have helped build PHP into a success few commercial entities have been able to emulate. I look forward to the future of PHP. I encourage any budding web developer to give PHP a spin. Like me, you may never want to give it up."

Randy Cosby
President, nGenuity, Inc.
DevShed (`http://www.devshed.com`)

An Introductory Example

Consider the example shown in Listing 1-1, which illustrates just how easily PHP can be integrated alongside HTML:

Listing 1-1: Dynamic PHP page creation

```
<?
// Set a few variables
$site_title = "PHP Recipes";
$bg_color = "white";
$user_name = "Chef Luigi";
?>

<html>
<head>
<title><? print $site_title; ?></title>
</head>
<body bgcolor="<? print $bg_color; ?>" >
<?
// Display an intro. message with date and user name.
print "
PHP Recipes | ".date("F d, Y")." <br>
Greetings, $user_name! <br>
";
?>
</body>
</html>
```

Figure 1-1 shows how the script appears when it is executed in the browser.

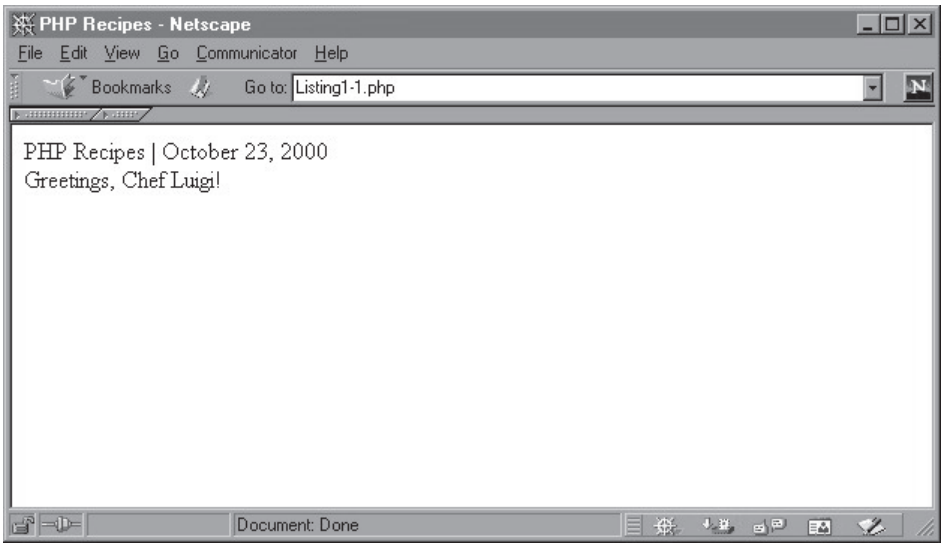

Figure 1-1. The script is executed in the browser.

Not too shabby, huh? I'm sure many a reader's mind is already churning with possibilities. However, before delving further into scripting issues, chances are you may need to install and configure PHP on your machine. This is the subject of the next few sections.

Downloading PHP/Apache

Before you proceed, I recommend that you take some time to download, install, and configure PHP and a Web server on your machine. Although PHP is compatible with a wide variety of Web servers, I'll assume that you will be using Apache, partly because it is currently the Web's most popular Web server and partly because it is the one most widely used with PHP. Regardless, the general installation process will not differ widely between Web servers.

You can download the PHP distribution from the official PHP site or from one of its many worldwide mirror sites. Go to `http://www.php.net` for the most recently updated mirror list. From here, you can download PHP in one of two formats:

- WIN32 Binary
- Source code

The Win32 binary is for Windows 95/98/NT/2000 users. While it is also possible to compile the source code on the Windows platform, for the large majority of users this won't be necessary. However, if you insist on doing so (incidentally, a process that is not discussed within this book), you'll need a recent Visual C++ compiler for doing so. Check out http://www.php.net/version4/win32build.php for more information on this process. The Win32 binary installation process is detailed later in this chapter.

For non-Windows users, you'll need to build the source code. While many beginners may shudder at this thought, it is actually a rather simple process, as you'll soon learn. For those of you interested to know whether or not PHP is offered in RPM (RedHat Package Manager) distribution format; it is, although these RPMs are not available via the official PHP site. Check the discussion groups (some of which are listed at the end of this chapter) for more information regarding distribution locations and instructions. The generalized build process is detailed later in this chapter.

Proceed to `http://www.php.net` and download the distribution that best suits your needs. Download times will vary with your connection type and speed. Additionally, the documentation is available for download. I strongly recommend downloading the most recent version.

> **TIP** *PHP 4.0.3 was the current stable version at the time of printing of this book. Of course, this version number is due to change along with the continued development of the PHP package. I recommend always downloading the most recent stable version of the product.*

If you haven't yet installed the Apache server, you will want to download the latest stable version of that as well. These packages are at http://www.apache.org/dist/binaries/, which contains directories for a plethora of operating systems. Download the one that is specific to your needs. Providing instructions regarding PHP configuration specifics for every available platform and Web server is out of the scope of this book. Therefore, I will concentrate on the Apache server. Regardless of the Web server you intend to use, I strongly recommend reading through the configuration sections later in this chapter to gain some insight into the generalized configuration issues that you may encounter.

Installation of new software can sometimes prove to be a daunting process for newcomers. However, the PHP developers have taken extra steps to make PHP installation relatively easy. The following sections highlight the steps you should take to install and configure PHP on both the non-Windows and the Win32 platforms.

NOTE *In later chapters I'll introduce the MySQL database server, using this popular product as the basis for illustrating Web/database integration. In order to experiment with these examples, you'll need to install the MySQL package, available at* `http://www.mysql.com`. *Like PHP, MySQL is available for both non-Windows and Windows platforms. Although I defer to the MySQL documentation due to its thorough installation instructions, you may be interested in taking a moment to read through the initial pages of Chapter 11, "Databases," for an introduction of the MySQL database server.*

Installation and Configuration

At this point, I'll assume that you have successfully downloaded PHP and Apache. The next step is deciding how you would like to install the distribution.For non-Windows machines, there three different ways to do so: CGI binary, static Apache module, and the dynamic Apache module. As a non-Windows user, chances are you will not want to build PHP as a CGI binary. Furthermore, there are several advantages to building PHP as a server module, therefore I'll concentrate solely on building PHP both as a static and a dynamic module. As it relates to installation, the main difference between the two is that any subsequent changes to the PHP static module will require the recompilation of both Apache and PHP, while changes to the PHP dynamic module only require the subsequent recompilation of just PHP and not the server.

For Windows machines, PHP can be installed as either a CGI binary or as a static Apache module. In this case, I'll concentrate upon the CGI binary, since a Windows-user might be more prone to use a Web server other than Apache, like Microsoft's Internet Information Server or Microsoft's Personal Web Server. The CGI version can easily be integrated into these servers. Although I illustrate the PHP/Apache Windows installation process, this process is very similar to that which would be used for the above-mentioned Web servers as well.

NOTE *Recall that PHP4 comes with support for a wide variety of Web servers, including AOL Server, Netscape Enterprise Server, Microsoft IIs, Zeus, and more. However, I will keep the installation process limited to that relating to Apache. For detailed instructions regarding how to install PHP with these other servers, check out the PHP documentation at* `http://www.php.net`.

Non-Windows

Regardless of the installation variation you choose, you'll need to begin by decompressing the distributions. This is accomplished in two easy steps:

1. Unzip the packages. Once done, you'll see that the files will be left with *.tar extensions:

```
gunzip apache_1.3.9.tar.gz
gunzip php-4.0.0.tar.gz
```

2. Untar the packages. This will unarchive the distributions:

```
tar -zxvf apache_1.3.x.tar
tar -zxvf php-4.0.x.tar
```

The installation procedure will pick up from this point.

Apache Module

Installing PHP as an Apache module is rather simple. I'll take you through each step here:

1. Change location to the Apache directory:

```
cd apache_1.3.x
```

2. Configure Apache. You can use any path you like. Keep in mind that a slash does *not* follow the pathname:

```
./configure –prefix=[path]
```

3. Change the location to the PHP directory and configure, build, and install the distribution. The option with-config-file-path specifies the directory that will contain PHP's configuration file. Generally, this path is set to be /usr/local/lib, but you can set it to be anything you wish:

```
./configure –with-apache=../apache_1.3.x –with-config-file-path=[config-path]
make
make install
```

4. Change back to the Apache directory. Now you will reconfigure, build, and install Apache. The other-configuration-options option refers to any special configuration options that you would like to pass along to the Apache Web server. This is beyond the scope of this book. I suggest checking out the Apache documentation for a complete explanation of these options:

```
./configure –activate-module=src/modules/php4/libphp4.a
–other-configuration-options
make
make install
```

5. The final step involves modifying Apache's httpd.conf file. Some of these modifications relate specifically to Apache, while others are necessary to ensure that PHP scripts can be recognized and sent to the Web server. First, locate the line that reads:

```
ServerName new.host.name
```

Change this line to read:

```
ServerName localhost
```

Next, locate the following two lines:

```
#AddType application/x-httpd-php .php .php4
#AddType application/x-httpd-php-source .phps
```

These lines need to be uncommented in order for PHP-enabled files to work correctly on the server. To uncomment these lines, simply remove the pound symbol (#) from the beginning of each line. Save the file and move up one directory. Start the Apache server using the following command:

```
./bin/apachectl start
```

Voilà! PHP and Apache are now ready for use. For testing purposes, insert the following code into a file and save the file as phpinfo.php to the Apache's document root directory. This is the directory called htdocs, located in the Apache installation directory.

```
<?
    php_info();
?>
```

Open this file up in a browser on the server. You should see a lengthy list of information regarding PHP's configuration. Congratulations, you've successfully installed PHP as an Apache Module.

Dynamic Apache Module

The Dynamic Module is useful because it allows you to upgrade your PHP distribution without having to recompile the Web server as well. Apache considers it just another one of its many modules, like ModuleRewrite or ModuleSpelling. This idea becomes particularly useful when you want to add some kind of support to PHP later, encryption, for example. All you have to do is reconfigure/compile PHP in accordance with the encryption support, and you can immediately begin using it in your Web applications. Here is the installation process:

1. Change location to the Apache directory:

   ```
   cd apache_1.3.x
   ```

2. Configure Apache. You can use any path you like. Keep in mind that a slash does *not* follow the pathname. The –other-configuration-options option refers to any special configuration options that you would like to pass along to the Apache Web server. This is beyond the scope of this book. I suggest checking out the Apache documentation for a complete explanation of these options:

   ```
   ./configure –prefix=[path] –enable-module=so –other-configuration-options
   ```

3. Build the Apache server. After typing **make**, you will see a bunch of messages scroll by. This is normal.

   ```
   make
   ```

4. Install the Apache server. After you type **make install**, another bunch of messages will scroll by. Again, this is normal. Once this has finished, you'll see a message stating that you have successfully installed the server.

   ```
   make install
   ```

5. Assuming no errors occurred, you're ready to modify Apache's "httpd.con" file. This file is located in the conf directory in the path that

you designated in step 4. Open this file in your favorite text editor. Locate the following line:

```
ServerName new.host.name
```

Modify this line to read:

```
ServerName localhost
```

6. Change location to the directory in which you downloaded PHP. Then, configure, make, and install PHP. You will need to specify the path directory pointing to the apxs file. This file can be found in the bin directory of the path you designated in step 4.

```
./configure –with-apxs=[path/to/apxs]
make
make install
```

7. Reopen Apache's httpd.conf file for another modification. In order for incoming requests for PHP-enabled files to be properly parsed, the file extension must coincide with the one as specified in the Apache server's configuration file, httpd.conf. This file contains a number of options, which can be modified at the administrator's discretion; a few of these options relate directly to PHP. Open the httpd.conf file in your favorite text editor. Towards the end of the file are two lines similar to the following:

```
#AddType application/x-httpd-php .php .php4
#AddType application/x-httpd-php-source .phps
```

8. You must uncomment these in order for PHP-enabled files to work correctly on the server. To uncomment these lines, simply remove the pound symbol (#) from the beginning of each line.

9. Save the file and move up one directory (to cd). Start the Apache server using the following command:

```
./bin/apachectl start
```

Voilà! PHP and Apache are now ready for use.

For testing purposes, insert the following code into a file and save the file as phpinfo.php to the Apache's document root directory. This is the directory called htdocs, located in the Apache installation directory.

```
<?
    php_info();
?>
```

Open this file up in a browser on the server. You should see a lengthy list of information regarding PHP's configuration. Congratulations, you've successfully installed the Dynamic Apache Module.

Installation on Windows 95/98/NT

If you have installed an application on the Windows operating system, you have probably found it to be very easy. Click a few buttons, agree to a few statements, and the application is installed. And so is the case with the installation of Apache and PHP on a Windows machine.

1. Double-click the Apache executable to begin the installation. You will be greeted with an installation wizard. Read attentively and accept the licensing agreement.

2. The wizard will suggest a default installation directory (C:\Program Files\Apache Group\Apache). This is fine, but you may want to shorten it to just C:\Apache\. However, it's up to you.

3. You will then be prompted for what name you would like to have appear in the Start menu. Enter whatever you want, or accept the default.

4. Next you will be prompted for the installation type. Just pick Typical. After you make your choice, the installation process is carried out.

5. Now it is time to modify the "httpd.conf" file, located in the conf directory, which is located in whatever directory you chose to install the Apache server in step 2. Open this file using your favorite text editor. You'll probably want to make at least three basic modifications:

Replace `yourname@yoursite.com` with the correct information.

```
ServerAdmin yourname@yoursite.com
```

Uncomment this line and place the correct server name. Just use localhost if you do not have an actual server name:

```
ServerName localhost
```

6. Attempt to start Apache to ensure that everything is working. At this point you need to make the differentiation as to the type of Windows OS you are using:

   If you're using Windows NT, choose "Install Apache as Service (NT Only)" from the Start menu. Then go to the Control Panel, open up the Services window, choose Apache, and click the "Start" button. Apache will start, and it will start automatically at every subsequent boot of the machine.

   If you're not using Windows NT, choose "Start Apache" from the Start menu. A small window will open. This window must be kept open in order for the server to run.

7. Finally, go to a browser installed on the server and enter **http://localhost/**. You should see a default page stating that the installation has been carried out correctly.

8. Now it's time to install PHP. Change the directory to wherever you downloaded the PHP package. Extract it to the directory of your choice using an unzipping application.

9. Go to that directory and look for a file entitled "php.ini-dist". Rename this file to php.ini and place it in the C:\Windows\ directory.

10. Go back to the PHP directory. Look for two more files, php4ts.dll and Mscvrt.dll. Place these files in the C:\Windows\System\ directory. You probably already have the Mscvrt.dll file, and you will be prompted to overwrite it. Don't overwrite the file or copy it.

11. Return to the Apache http.conf file, again opening it up in a text editor. There are a few more modifications that you need to make:

Look for this line:

```
ScriptAlias /cgi-bin/ "C:/Apache/cgi-bin/"
```

Directly below this line, add the following:

```
ScriptAlias /php4/ "C:/php4/"
```

Then search for "AddType". You will see the following two commented lines:

```
#AddType application/x-httpd-php3 .phtml
#AddType application/x-httpd-php3-source .phps
```

Directly below these lines, add the following:

```
AddType application/x-httpd-php .phtml .php
AddType application/x-httpd-php-source .phps
```

Keep scrolling down. You will find the following commented lines:

```
#
# Action lets you define media types that will execute a script whenever
# a matching file is called. This eliminates the need for repeated URL
# pathnames for oft-used CGI file processors.
# Format: Action media/type /cgi-script/location
# Format: Action handler-name /cgi-script/location
#
```

Below this, add the following:

```
Action application/x-httpd-php /php4/php.exe
```

12. Voilà! PHP and Apache are now ready for use.

For testing purposes, insert the following code into a file and save the file as "phpinfo.php" to the Apache's document root directory. This is the directory called htdocs located in whatever directory you specified in step 4.

```
<?
    php_info();
?>
```

CAUTION *Although successfully completing the steps outlined above does make it possible for the Web server/PHP configuration to be used for testing purposes, it does not imply that your Web server is accessible via the World Wide Web. Check out the official Apache site (*`http://www.apache.org`*) for information regarding this matter. Furthermore, although the preceding steps suffice to get the PHP package up and running, you will probably be interested in modifying PHP's configuration to best suit your needs. See "PHP Configuration," later in this chapter, for details.*

Open this file in a browser on the server. You should see a lengthy list of information regarding PHP's configuration.

PHP Configuration

Although PHP will correctly run given its default configuration setting, you can make quite a few modifications to fine-tune the installation to your needs. The php.ini file, copied by default into the /usr/local/lib/ directory during the installation process, contains all of these configuration settings.

Regardless of the platform and Web server used in conjunction with PHP, the php.ini file will contain the same default set of parameters, from which several

NOTE *The configuration file is entitled php3.ini in the 3.0 version but has been changed to php.ini in the 4.0 version.*

important characteristics of the PHP installation can be administered. This file contains all of the characteristics relevant to how your installation will act when PHP scripts are executed. The PHP engine reads the php.ini file when PHP starts up.

General Configuration Directives

Reiterating all of the configuration directives is beyond the scope of this book, but there are several directives worth mentioning, as most the developers may find them particularly useful. I'll mention other directives as appropriate in subsequent chapters.

short_open_tag [on | off]

The short_open_tag [on | off] configuration directive determines the use of the short PHP escape tags <?…?>, in addition to the default tags.

asp_tags [on | off]

The asp_tags [on | off] configuration directive determines the use of ASP style tags in addition to the default tags. ASP style tags are those that enclose PHP code as follows:

```
<%
print "This is PHP code.";
%>
```

precision [integer]

The precision [integer] configuration directive sets the number of significant digits displayed in floating point numbers.

safe_mode [on | off]

Turning on safe mode is a particularly good idea if you have several users on your system. Essentially, turning on safe mode eliminates the possibility that a user can use a PHP script to gain access to another file on the system, for example, the passwd file on a Linux machine. Safe_mode works solely on the CGI version of PHP. Check out Chapter 16 for more details regarding this matter.

max_execution_time [integer]

The max_execution_time [integer] configuration directive determines the maximum number of seconds that a given PHP script may execute. This prevents runaway scripts from eating up valuable system resources.

error_reporting [1-8]

The error_reporting [1-8] configuration directive gauges to what degree errors will be reported, if any. The higher the bit value, the more sensitive PHP will be to reporting errors:

BIT VALUE	REPORTING SENSITIVITY
1	normal errors
2	normal warnings
4	parser errors
8	notices

display_errors [on | off]

The display_errors [on | off] configuration directive display the errors in the browser.

log_errors

The log_errors configuration directive determines whether or not errors are logged to a file. If log_errors is turned on, the directive error_log designates which file the errors are logged to.

error_log [filename]

If log_errors is turned on, error_log designates the filename to which all errors should be logged.

magic_quotes_gpc

When magic_quotes_gpc is activated, all special characters contained in user or database data will automatically be escaped with the necessary backslash. By the way, "gpc" stands for "get/post/cookie".

Personally, I find it more efficient to keep magic_quotes_gpc turned off and to escape the special characters explicitly. Regardless of the way you ultimately decide to do it, there can be no compromise or your data may be corrupted. If magic_quotes_gpc is "on", then never physically escape special characters with a backslash; otherwise, make it a habit to always do so.

track_vars

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:

```
<?
// file: example.php
// author: WJ Gilmore
// date: August 24, 2000

print "An example with comments";
?>
```

Multiline Comments

PHP provides a mechanism for detailed comments that may take up more than one line. This type of comment is enclosed in C-style comments, denoted with an opening '/*' and '*/'.

```
<?
/*
    script: multi_comment_example.php
    purpose: Multiline comment example
    author: wj gilmore
    date: June 14, 2000
*/

print "A multiline comment can be found at the top of this script!";
?>
```

As you can see, multiline comments are useful when you need to provide a relatively lengthy summary of a script or a part of one.

What's Next?

This chapter brought you up to speed regarding several key aspects of PHP, namely:

- PHP's history and features
- Installation and configuration
- "Escaping" to PHP

- Commenting PHP code

These topics serve as the introduction to subsequent chapters, where you will learn more about the developmental issues regarding the PHP language. At the conclusion of the next chapter, you will know enough about PHP to begin writing your own programs. You will apply this knowledge by developing an events calendar that can be easily inserted into an existing Web page. This project will serve as the precursor for further development of the PHP Recipes Web application.