THE EXPERT'S VOICE® IN WEB DEVELOPMENT

SECOND EDITION

PHP for Absolute Beginners

EVERYTHING YOU NEED TO KNOW TO GET STARTED IN PHP

Thomas Blom Hansen and Jason Lengstorf

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

Modern web development relies on the successful integration of several technologies. Content is mostly formatted as HTML. With server-side technologies, you can create highly dynamic web applications. PHP is the single most used server-side scripting language for delivering browser-based web applications. PHP is the backbone of online giants such as Facebook, Flickr, and Yahoo.

There are other server-side languages available for web application development, but PHP is the workhorse of the Internet. For an absolute beginner, it should be comforting to know that PHP is a relatively easy language to learn. You can do many things with a little PHP. Also, there is a thriving, friendly community supporting PHP. It will be easy to get help with your own PHP projects.

Who Should Read This Book

This book is intended for those who know some HTML and CSS. It is for those who are ready to take their web developer skills to the next level. You will learn to generate HTML and CSS dynamically, using PHP and MySQL. You will learn the difference between client-side and server-side scripting through hands-on experience with PHP/MySQL and JavaScript code projects. Emphasis will be on getting up and running with PHP, but you will also get to use some MySQL and some JavaScript in your projects. By the end of the book, you will have created a number of PHP-driven projects, including the following:

- A personal portfolio site with dynamic navigation
- A dynamic image gallery where users can upload images through an HTML form
- A personal blogging system, complete with a login and an administration module

In the process, you will become acquainted with such topics as object-oriented programming, design patterns, progressive enhancement, and database design. You will not get to learn everything there is to know about PHP, but you will be off to a good start.

How to Read This Book

This book is divided into two main parts. Part I will quickly get you started writing PHP for small, dynamic projects. You will be introduced to a relatively small subset of PHP–just enough for you to develop entry-level web applications. Part I will also teach you the basic vocabulary of PHP.

Part II is a long hands-on project. You will be guided through the development of the aforementioned personal blogging system, starting from scratch. Part II will show you how to use your PHP vocabulary to design dynamic, database-driven web applications.

PART I

PHP/MySQL Basics

You will learn how to set up a PHP/MySQL development environment, the basics of PHP and how to connect PHP to a MySQL database.

CHAPTER 1

Setting Up a PHP Development Environment

Getting a working development environment put together can be intimidating, especially for the absolute beginner. To follow along with the project in this book, you'll need to have access to a working installation of Apache, PHP, and MySQL, preferably on your local machine. It's always desirable to test locally, both for speed and security. Doing this both shelters your work-in-progress from the open Internet and decreases the amount of time spent uploading files to an FTP server and waiting for pages to reload.

Why You Need Apache, MySQL, and PHP

PHP is a powerful scripting language that can be run by itself in the command line of any computer with PHP installed. However, PHP alone isn't sufficient for building dynamic web sites. To use PHP on a web site, you need a server that can process PHP scripts. Apache is a free web server that, once installed on a computer, allows developers to test PHP scripts locally; this makes it an invaluable piece of your local development environment.

Additionally, web sites developed with PHP often rely on information stored in a database, so it can be modified quickly and easily. This is a significant difference between a PHP site and an HTML site. This is where a relational database management system such as MySQL comes into play. This book's examples rely on MySQL. I chose this database because PHP provides native support for it, and because MySQL is a free, open source project.

Note An open source project is available for free to end users and ships with the code required to create that software. Users are free to inspect, modify, and improve the code, albeit with certain conditions attached. The Open Source Initiative lists ten key provisions that define open source software. You can view this list at `www.opensource.org/docs/osd`.

PHP is a general-purpose scripting language that was originally conceived by Rasmus Lerdorf in 1995. Lerdorf created PHP to satisfy the need for an easy way to process data when creating pages for the World Wide Web.

Note PHP was born out of Rasmus Lerdorf's desire to create a script that would keep track of how many visits his online résumé received. Due to the wild popularity of the script he created, Lerdorf continued developing the language. Over time, other developers joined him in creating the software. Today, PHP is one of the most popular scripting languages in use on the Internet.

PHP originally stood for *Personal Home Page* and was released as a free, open source project. Over time, the language was reworked to meet the needs of its users. In 1997, PHP was renamedPHP: *Hypertext Preprocessor*, as it is known currently. At the time I write this, PHP 5.5.7 is the current stable version. Older versions of PHP are still in use on many servers.

How PHP Works

PHP is generally used as a server-side scripting language; it is especially well-suited for creating dynamic web pages. The scripting language features integrated support for interfacing with databases, such as MySQL, which makes it a prime candidate for building all manner of web applications, from simple personal web sites to complex enterprise-level applications.

HTML is parsed by a browser when a page loads. Browsers cannot process PHP at all. PHP is processed by the machine that serves the document (this machine is referred to as a server). All PHP code in the document is processed by the server before the document is sent to the visitor's browser. Because PHP is processed by a server, it is a *server-side scripting language*.

With PHP, you can create *dynamic* web pages—web pages that can change according to conditions. For example: When I log in to my Facebook account, I see my content. When you log in to your Facebook account, you see your content. We would be loading the same resource (`www.facebook.com`), but we would be served different content *dynamically*. This would be impossible with HTML web documents, because they are *static*, meaning they can't change. Every user would see exactly the same HTML page. The rest of this book explores some of the things you can achieve with dynamic web pages.

PHP is an interpreted language, which is another great advantage for PHP programmers. Many programming languages require that you compile files into machine code before they can be run, which is a time-consuming process. Bypassing the need to compile means you're able to edit and test code much more quickly.

Because PHP is a server-side language, running PHP scripts requires a server. To develop PHP projects on your local machine means installing a server on your local machine. The examples in this book rely on the Apache Web Server to deliver your web pages.

Apache and What It Does

Apache is the most popular web server software on the Web; it hosts nearly half of all web sites that exist today. Apache is an open source project that runs on virtually all available operating systems. Apache is a community-driven project, with many developers contributing to its progress. Apache's open source roots also means that the software is available free of charge, which probably contributes heavily to Apache's overwhelming popularity relative to its competitors, including Microsoft's IIS and Google's GWS, among others.

On the Apache HTTP Server Project web site (`http://httpd.apache.org`), Apache HTTP Server is described as "an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards."

As with all web servers, Apache accepts an HTTP request and serves an HTTP response. The World Wide Web is founded on web servers, and every web site you visit demonstrates the functionality of web servers. I've already mentioned that while HTML can be processed by a web browser, server-side scripting languages such as PHP have to be handled by a web server. Due to its overwhelming popularity, Apache is used for testing purposes throughout this book.

Storing Info with MySQL

MySQL is a relational database management system (RDBMS). Essentially, this means that MySQL allows users to store information in a table-based structure, using rows and columns to organize different pieces of data. There are many other relational database management systems. The examples in this book rely on MySQL to store the information you'll use in your PHP scripts, from blog entries to administrator information. This approach has great advantages, which we will explore in detail.

■ **Note** *Blog* is short for *weblog*, which is an online journal produced by an individual or a business.

Installing PHP, Apache, and MySQL

One of the biggest hurdles for new programmers is starting. Before you can write your first line of PHP, you must download Apache and PHP, and usually MySQL, and then fight through installation instructions that are full of technical jargon you might not understand yet. This experience can leave many developers feeling unsure of themselves, doubting whether they've installed the required software correctly.

In my own case, this hurdle kept me from learning programming for months, even though I desperately wanted to move beyond plain ole HTML. I unsuccessfully attempted to install PHP on my local machine not once, but three different times before I was able to run my first PHP command successfully.

Fortunately, the development community has responded to the frustration of beginning developers with several options that take all the pain out of setting up your development environment, whether you create applications for Windows, Mac, or Linux machines. These options include all-in-one solutions for setting up Apache, MySQL, and PHP installations.

The most common all-in-one solution is a program called XAMPP (`www.apachefriends.org/en/xampp.html`), which rolls Apache, MySQL, PHP, and a few other useful tools together into one easy installer. XAMPP is free and available for Windows, Mac, and Linux. This book assumes that you will use it as your development environment.

■ **Note** Most Linux distributions ship with one flavor or another of the LAMP stack (Linux-specific software that functions similarly to XAMPP) bundled in by default. Certain versions of Mac OS X also have PHP and Apache installed by default.

Installing XAMPP

Enough background. You're now ready to install XAMPP on your development machine. This process should take about five minutes and is completely painless.

Step 1: Download XAMPP

Your first task is to obtain a copy of the XAMPP software. Head over to the XAMPP site (`www.apachefriends.org/en/xampp.html`) and download the latest version (1.8.3 at publication time).

Step 2: Open the Installer and Follow the Instructions

After downloading XAMPP, find the newly downloaded installer and run it. You should be greeted with a screen similar to the one shown in Figure 1-1.

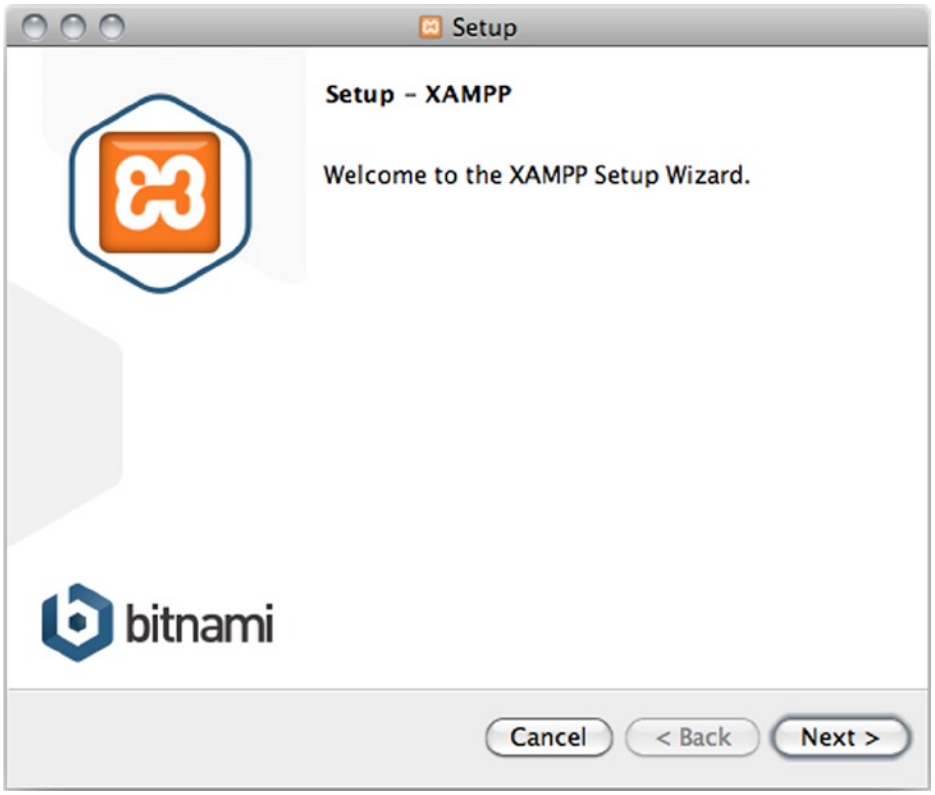

Figure 1-1. *The introductory screen for the XAMPP installer on Mac OS X*

■ **Note** All screenshots used in this book were taken on a computer running Mac OS X 10.6.8. Your installation might differ slightly, if you use a different operating system. XAMPP for Windows offers additional options, such as the ability to install Apache, MySQL, and Filezilla (an FTP server) as services. This is unnecessary and will consume computer resources, even when they are not being used, so it's probably best to leave these services off. Additionally, Windows users should keep the `c:\xampp` install directory for the sake of following this book's examples more easily.

Click the Next button to move to the next screen (see Figure 1-2), where you can choose which components to install. Just go with the default selection. The XAMPP installer will guide you through the installation process. Figures 1-3 through 1-5 show the remaining steps.

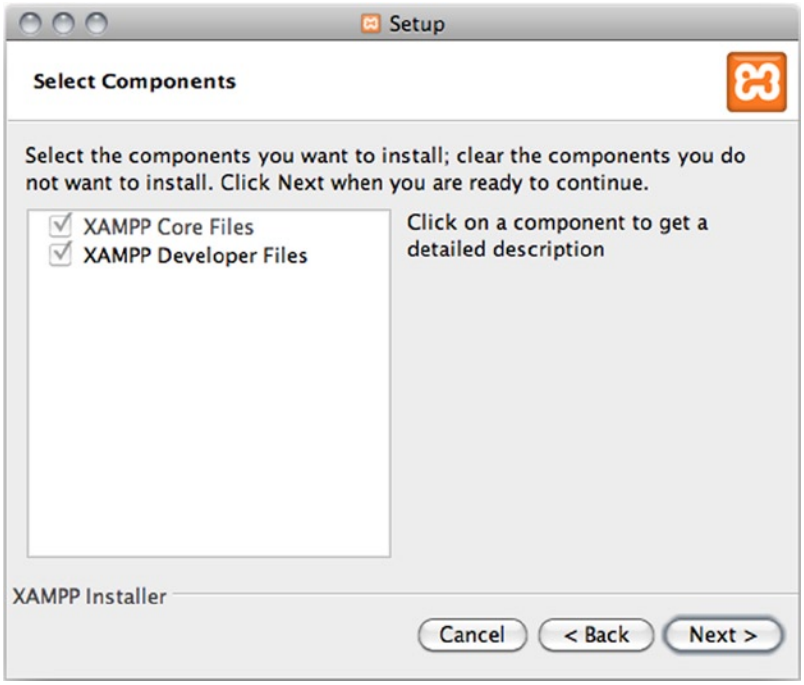

Figure 1-2. *Select components to install*

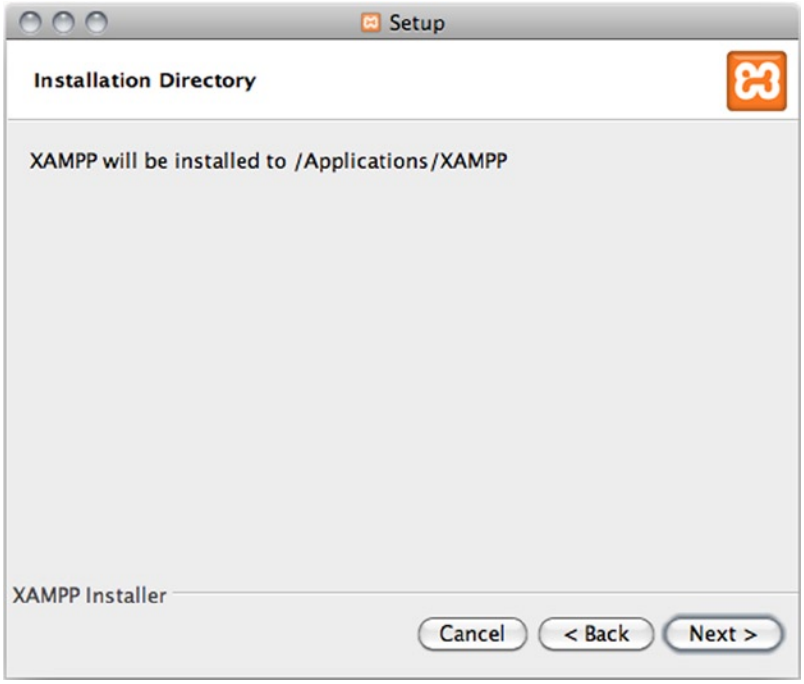

Figure 1-3. *XAMPP installation directory*

Figure 1-4. *You don't have to learn more about BitNami at this point*

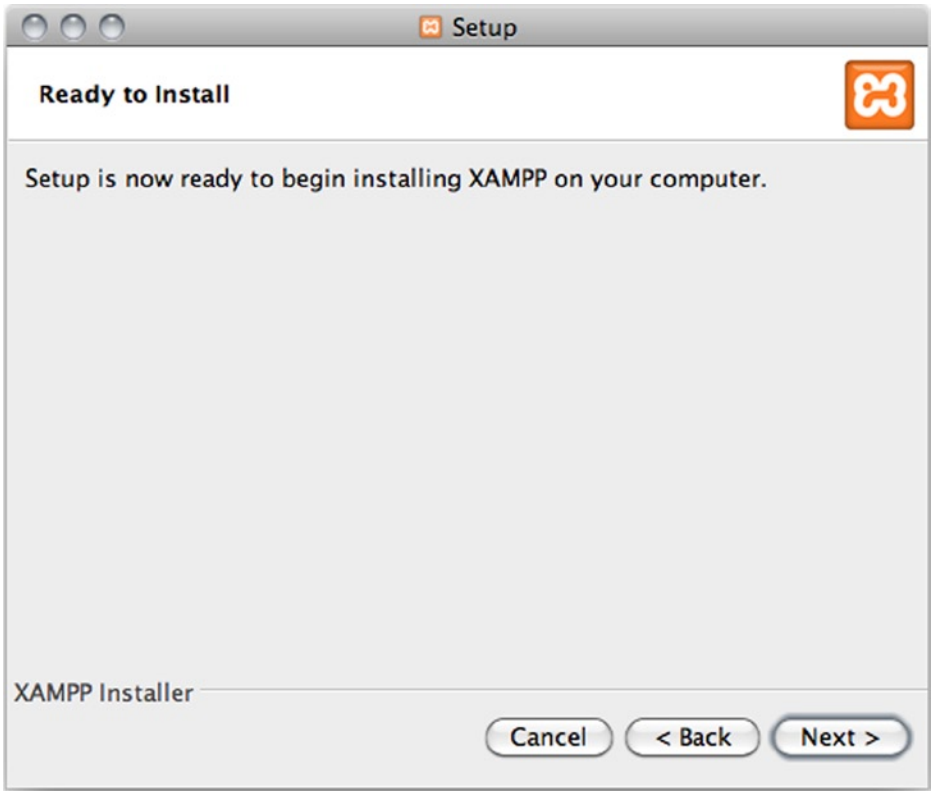

Figure 1-5. *When you're ready to install, click Next*

Installation requires a minute or two to complete, whereupon the installer displays the final screen (see Figure 1-6), which confirms that the installation was successful.

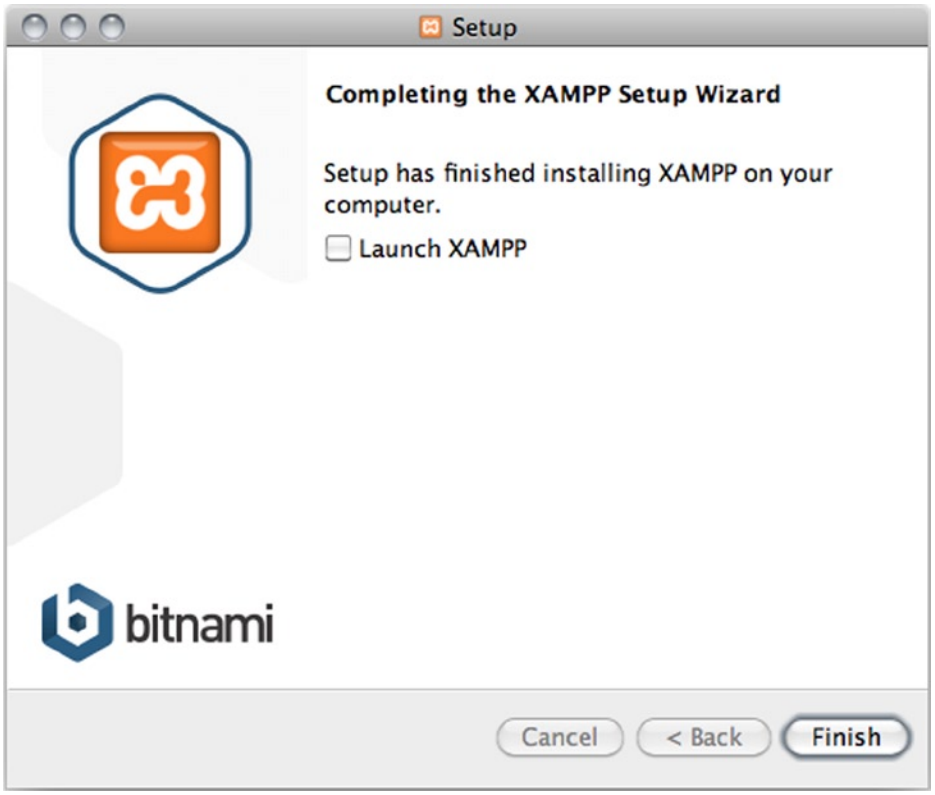

Figure 1-6. *Installation is complete*

Step 3: Test XAMPP to Ensure Proper Installation

So far, you've used the XAMPP wizard to install Apache, PHP, and MySQL. The next step is to activate Apache, so you can write some PHP.

Open the XAMPP Control Panel

You can activate the just-installed applications by navigating to the newly installed XAMPP folder and opening the XAMPP manager (see Figure 1-7).

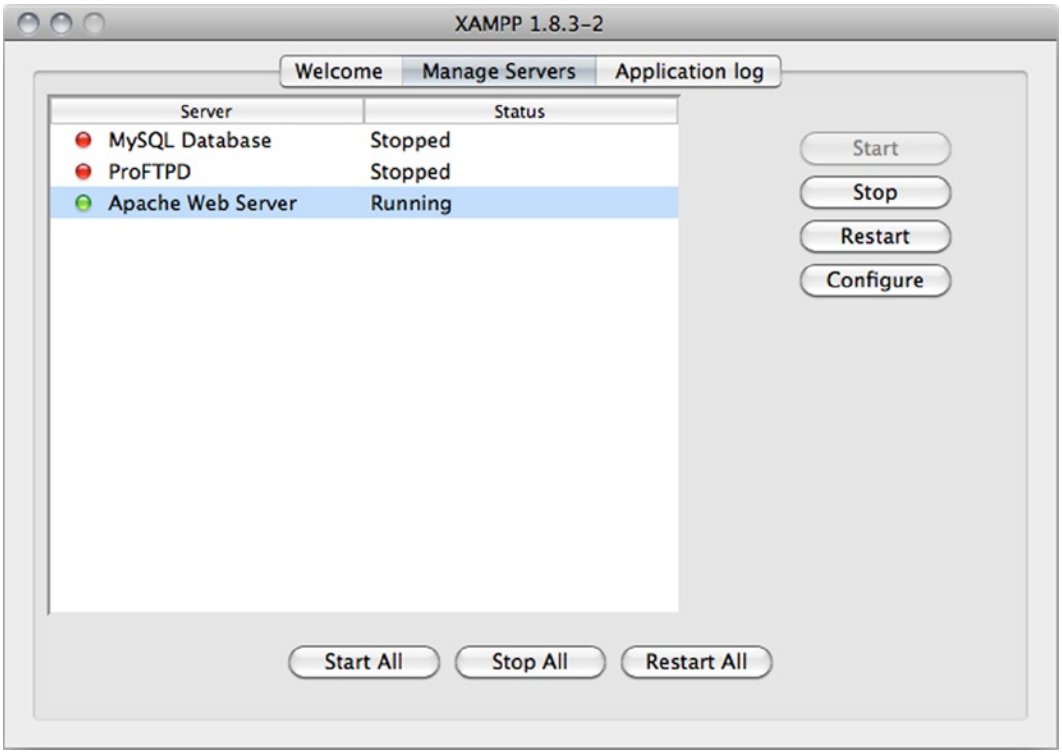

Figure 1-7. *The XAMPP manager shows that the local Apache Web Server is running*

■ **Note** When opening the XAMPP control panel, you may be prompted for your password. This has no effect on the services themselves and should not affect the projects covered in this book.

Activating Apache, PHP, and MySQL on your development machine is as simple as clicking the Start button next to Apache in the XAMPP manager. You might be prompted to confirm that the server is allowed to run on your computer, and you might be required to enter your system password. After you do this, the Status should indicate that Apache is running, as shown in Figure 1-7.

■ **Note** There is an FTP (file transfer protocol) option available in XAMPP. FTP provides a method for moving files between networks. The examples in this book don't require this option, so there is no need to activate it in the XAMPP control panel. The first few chapters don't even require a MySQL database.

What If Apache Isn't Running?

Sometimes, XAMPP Apache Server doesn't run, even if you try to start it. The most common problem is that it conflicts with some other service using the same port on your computer. Check if you have Skype or Messenger or some similar networking service running. Shut Skype completely down, and if you're lucky, your Apache can run.

If it still doesn't run, you could turn to the Internet for help. The XAMPP online community is extremely helpful, and most installation issues have been addressed in the Apache Friends forum at `https://community.apachefriends.org/f/viewforum.php?f=34`. You could also turn to search or ask at `http://stackoverflow.com/`.

Verify That Apache and PHP Are Running

It's a simple matter to check whether Apache is running properly on your development machine. Simply open a browser and go to the following address: `http://localhost`. If everything has gone correctly, you'll be redirected to `http://localhost/xampp/splash.php` (see Figure 1-8).

Figure 1-8. *Check in your browser that your Apache Web Server is running*

If this screen loads, you've installed Apache and PHP on your development machine successfully! The address, `http://localhost`, is an alias for the current computer you're working on. When using XAMPP, navigating to `http://localhost` in a browser tells the server to open the root web directory. This is the `htdocs` folder contained in the XAMPP install directory. Another way to use your server to access the root web directory on your local machine is to navigate to the IP address—a numerical identifier assigned to any device connected to a computer network—that serves as the "home" address for all HTTP servers: `http://127.0.0.1`.

Choosing a PHP Editor

Your development machine is now running all the necessary programs for programming with PHP. The next step is to decide how you're going to write your scripts. PHP scripts are text-based, so you have myriad options, ranging from the simple `Notepad.exe` and text-edit programs to highly specialized integrated development environments (IDEs).

Most experienced PHP developers use an IDE, because they offer many benefits. Many beginners have some difficulties using an IDE, perhaps because IDEs have so many features that beginners are simply left confused.

You can probably write PHP code using whichever program you have used for writing HTML and CSS. There are some features you should expect from a good editor.

- *Syntax highlighting*: This is the ability to recognize certain words in a programming language, such as variables, control structures, and various other special text. This special text is highlighted or otherwise differentiated to make scanning your code much easier.
- *Built-in function references*: When you enter the name of a function or an object method, this feature displays available parameters, as well as the file that declares the function, a short description of what the function does, and a more in-depth breakdown of parameters and return values. This feature proves invaluable when dealing with large libraries, and it can save you trips to the PHP manual to check the order of parameters or acceptable arguments for a function.
- *Auto-complete features*: This feature adds available PHP keywords to a drop-down list, allowing you to select the intended keyword from the list quickly and easily, saving you the effort of remembering and typing it out every time. When it comes to productivity, every second counts, and this feature is a great way to contribute to saved time.
- *Code folding*: This feature lets you collapse snippets of code, making your workspace clutter-free and your code easy to navigate.
- *Auto-indent*: This automatically indents the code you write in a consistent manner. Such indented code is vastly easier to read for human readers, because indentation indicates relationships between code blocks.
- *Built-in ftp*: You need ftp to upload your PHP files to an online web server when you want to publish your project on the World Wide Web. You can use a stand-alone ftp program, but if it is built into your IDE, you can upload an entire project with a single click.

You have many good IDEs and editors to choose from. NetBeans and Eclipse PDT are both excellent, free IDEs. Try one or both, if you want to get used to the tools professional developers often gravitate to. Beginners may find it easier to start with a simpler editor. I really like Komodo Edit: It is as easy to use as any editor, and it provides most of the features just listed out of the box, including excellent auto-complete for PHP and many other languages. Following are the download links for the three PHP editors just mentioned:

- Get NetBeans from `https://netbeans.org/downloads/`.
- Get Eclipse PDT from `http://projects.eclipse.org/projects/tools.pdt`.
- Get Komodo Edit from `www.activestate.com/komodo-edit/downloads`.

I will use Komodo Edit for the examples in this book. You should have no difficulties following the examples with any other editor. If you decide to use an IDE, you will have to consult online documentation to learn how to set up a new project in your chosen IDE.

Creating Your First PHP File

With everything set up and running as it should, it is time to take the plunge and write your first PHP script. As a server-side scripting language, PHP requires a web server such as Apache to run. You have just installed Apache on your local computer, so your system is ready.

Apache will interpret any PHP files saved inside a folder called `htdocs`. You can find it inside your XAMPP installation in `XAMPP/xamppfiles/htdocs`.

You'll be making many PHP files soon, so it is a good idea to keep them organized. Create a new folder inside `htdocs` and call it `ch1`.

Now open Komodo Edit, or whichever editor or IDE you have decided to use. From Komodo Edit, you can select File ➤ New ➤ New File from the main menu. In the new file you write the following:

```
<?php

echo "Hello from PHP";
```

In Komodo Edit, select File ➤ Save to see the file saving dialog box (Figure 1-9). Save the new file as `test.php`, in `htdocs/ch1`; set format to All Files; and then click Save.

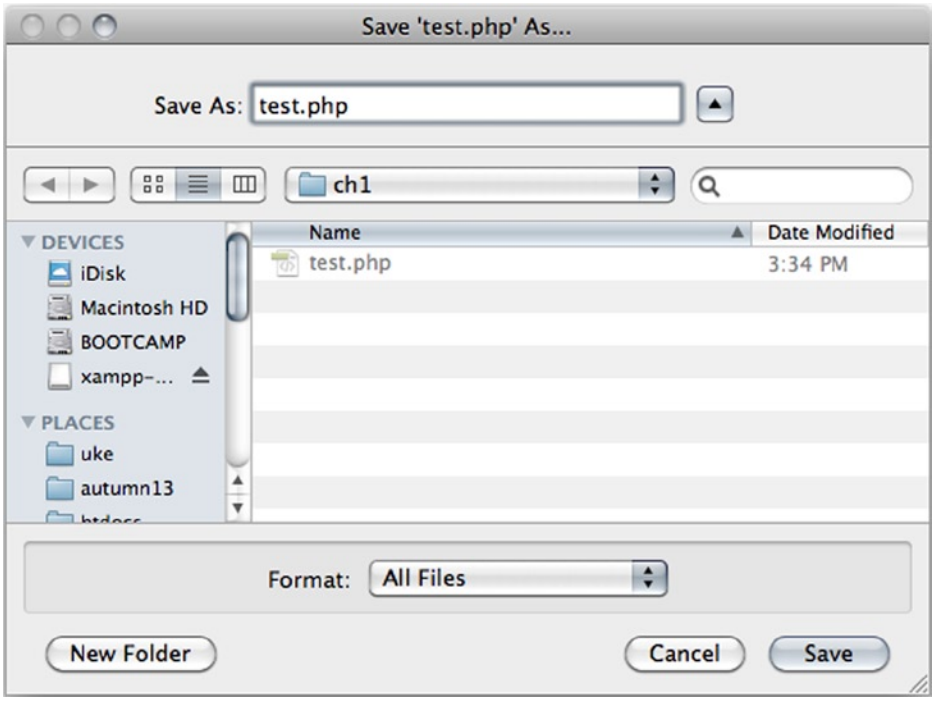

Figure 1-9. *The Save As dialog box from Komodo Edit*

Running Your First PHP Script

The next step is to get Apache to process your PHP script. That happens automatically, if you request the script through a browser. So, open a web browser and navigate to `http://localhost/ch1/test.php` and marvel at the PHP-generated output you should see in your browser (Figure 1-10). You have successfully created and executed your first PHP script!

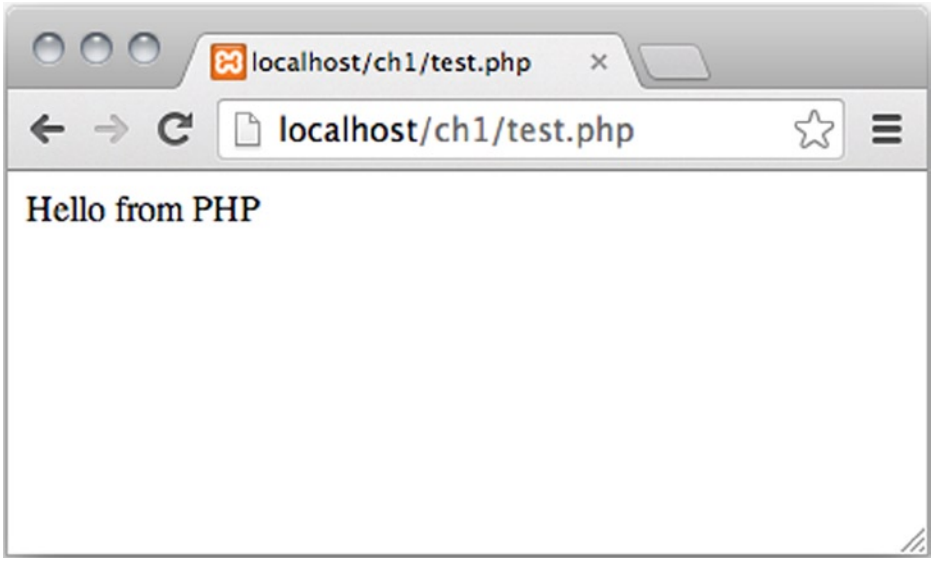

Figure 1-10. *Seeing the output from* `test.php` *in the Chrome web browser*

Summary

In this chapter, you learned a little bit about PHP, MySQL, and Apache. You found out what they are and what role they play in the development of dynamic web sites. You also learned a quick and easy way to install a fully functional development environment on your local computer, by installing XAMPP and Komodo Edit.

In the next chapter, you'll learn a small but potent subset of PHP, including variables, objects, and some native language constructs and statements. Nearly everything you learn will be tested in your new development environment, so keep XAMPP's Apache Server open and running.

CHAPTER 2

Understanding PHP: Language Basics

So far, you've bypassed the old, cumbersome method of creating a development environment, and you're now ready to start writing code.

But where do you start? In this chapter, I'll cover the steps you need to follow to start using PHP in the creation of powerful, dynamic web applications. You'll also begin to develop the basic skills you need to create your blog. In addition, you'll learn how to accomplish several tasks, including how to do the following:

- Embed PHP in web pages
- Send data as output to the browser
- Add comments in your code
- Use variables
- Work with PHP errors
- Create an HTML5 template
- Use objects
- Concatenate strings
- Access URL variables with the `$_GET` superglobal
- Declare a class definition
- Embed dynamic CSS

By the end of this chapter, you will have seen some basic PHP that will allow you to create, store, manipulate, and output data. You will have used those skills to develop a bare-bones version of a personal portfolio web site.

Note This chapter discusses basic aspects of the PHP language, but not in complete detail. For clarification, more examples, or for concept reinforcement, visit the PHP manual at `www.php.net/manual/en/` and search the function in the field where it says "search for _____ in the function list." Alternatively, you can access information about many PHP functions by navigating to `http://php.net/function_name`. Don't forget to read the comments, because many of your fellow programmers offer insight, tips, and even additional functions in their commentary.

Embedding PHP Scripts

In Chapter 1, when I talked about Apache and web servers in general, I mentioned how a server will process PHP in a file before sending that file to the browser. But you might be curious as to how the server knows where to look for PHP.

By default, servers look for PHP only in files that end with the `.php` extension. But a `.php` file can contain elements that aren't part of your PHP script, and searching the entire file for potential scripts is confusing and resource-intensive. To solve this issue, all PHP scripts need to be contained with PHP delimiters. To begin a PHP script, you include the opening delimiter `<?php` and start coding. To finish, you simply add `?>` to the end of the script. Anything outside of these delimiters will be treated as HTML or plain text.

You can see this in action. Start by creating a new folder `ch2` in `/xampp/htdocs/`. Next, create a new file, `test.php`, with Komodo Edit. Write the following code:

```
<p>Static Text</p>
<?php
echo "<p>This text was generated by PHP!</p>";
?>
<p>This text was not.</p>
```

Save the file, navigate to `http://localhost/ch2/test.php` in your browser, and you should see the following output in your browser:

```
Static Text
This text was generated by PHP!
This text was not.
```

As you can see, the text inside the PHP delimiters , was handled as a script, but the text outside was rendered as regular HTML. There is no limit to how many blocks of PHP you can include in a page, so the following snippet is completely valid:

```
<?php
echo "<p>This is some text.</p>";
?>
<p>Some of this text is static, <?php echo "but this sure isn't!"; ?></p>
<?php echo "<p>"; ?>
This text is enclosed in paragraph tags that were generated by PHP.
<?php echo "</p>"; ?>
```

The preceding code snippet outputs the following to the browser:

```
This is some text.
Some of this text is static, but this sure isn't!
This text is enclosed in paragraph tags that were generated by PHP.
```

If you write a PHP script that holds nothing but PHP, you don't have to end the PHP delimiter. You only have to mark the ending of a PHP code block, if you are going to write something that is *not* PHP in the file.

Using echo

Take an extra look at the use of `echo` in the preceeding code examples. PHP's `echo` is a so-called *language construct*—the basic syntactic units PHP is made of. The `echo` statement is probably the most common approach for outputting text from PHP to the browser. That is all `echo` does. It sends output to the browser.

Notice that the output strings are delimited with double quotes in the preceding code example. The initial double quote indicates the beginning of a string of characters. The second double quote marks the end of the string to output. In PHP, you must delimit any strings you are using in your code. The string delimiters tell PHP when a string of characters begin and end, something PHP needs to know in order to process your code.

■ **Note** *String* is a geeky word for "text." Because computers are not human, they don't really see texts, much less words. They see *strings* of characters.

What Is a Variable?

A variable is a keyword or phrase that acts as an identifier for a value stored in a system's memory. This is useful, because it allows us to write programs that will perform a set of actions on a variable value, which means you can change the output of the program simply by changing the variable, rather than changing the program itself.

Storing Values in a Variable

It is quite straightforward to store a value in a variable. In one single line, you can declare a new variable and assign a value to it:

```
<?php
$myName = "Thomas";
$friendsName = "Brennan";
echo "<p>I am $myName and I have a friend called $friendsName.</p>";
```

If you type the preceding lines into your `test.php` file and load it in your browser, you should see an output such as the following:

```
I am Thomas and I have a friend called Brennan.
```

Perhaps you will notice that the preceding code holds nothing but PHP. Consequently, there is no need to mark the end of the PHP block with a PHP delimiter. You can add `?>` at the end, if you like; it'll make no difference.

A Variable Is a Placeholder

Variables are used extensively in programming. It is a basic concept you must come to understand. There is an important lesson to be learned from the example preceding. When you read the PHP code, you see variable names:

```
echo "<p>I am $myName and I have a friend called $friendsName.</p>";
```

You can see the output from PHP in the browser. You can see that PHP replaces the variable names with string values. For example, when you see `$myName`, PHP sees *Thomas*. When you see `$friendsName`, PHP sees *Brennan*.

A variable is a placeholder for a specific value. PHP doesn't even notice the variable; it sees the value stored inside. Metaphorically, you could understand a variable as a container—a cup, for example. I have a cup right next to my computer, and I can put all sorts of things inside it: coffee, a pencil, or some loose change. PHP variables are like that. PHP sees what is contained, not the container.

■ **Note** In technical terms, PHP variables are *passed by value*, as opposed to *passed by reference*.

Valid PHP Variable Names

In PHP, all variables must begin with a dollar sign character ($). There are some further restrictions on valid variable names, but if you simply use alphabetical characters only, you will encounter no problems with invalid variable names. So, avoid whitespace characters, numbers, and special characters such as !"#€%&/.

■ **Note** You can actually use numbers in variable names but not in initial positions. So, `$1a` is an invalid variable name, whereas `$a1` is perfectly valid.

Displaying PHP Errors

On your journey toward learning PHP, you are bound to produce some errors. It is easy to think that you have done something bad when you have written some erroneous PHP. In a sense, it is, of course, bad. You would probably prefer to write perfect PHP from the very start.

In another sense, errors are a very good thing. Many such errors present a learning opportunity. If you really understand the cause of an error, you are less likely to repeat it, and even if you do repeat it, you can easily correct the error if you understand it.

PHP error messages are not always displayed—it depends on how your development environment is set up. If you write the following two lines of PHP at the beginning of your scripts, all error messages will be displayed. Let's produce an error:

```
<?php
//these two lines tell PHP to show errors in the browser
error_reporting( E_ALL );
ini_set( "display_errors", 1 );

//here comes the error
echo "This string never ends;
```

Do you see the error? There is only one string delimiter. To write valid PHP, you must wrap your strings in string delimiters, for example, double quotes. In the preceding example, the end delimiter is missing, so PHP cannot see where the output ends. If you run the code, you will see an error message in your browser, as follows:

Parse error: syntax error, unexpected $end, expecting T_VARIABLE or T_DOLLAR_OPEN_CURLY_BRACES or T_CURLY_OPEN in**/Applications/XAMPP/xamppfiles/htdocs/ch2/test.php** on line **4**

Error messages are friendly but not always as precise as you might prefer. When PHP is unable to process your code, an error is triggered. PHP will make an educated guess about what the problem might be. In the preceding example, PHP has encountered an "unexpected end" on line 4. There is a "bug" in your script. Please debug the script by adding the missing double quote.

I recommend you make a habit of forcing error messages to display and try to read all error messages you come across. If you encounter an error message you don't understand, you can always search the Internet for an explanation. A site such as `www.stackoverflow.com` is very likely to have an explanation for your particular error message.

Creating an HTML5 Page with PHP

PHP is a wonderful language for creating dynamic HTML pages. With a tiny bit of PHP, you can create a valid HTML5 page with variable content in memory and have PHP output the created page to the browser. Let's make a bare-bones skeleton for a personal portfolio site. Create a new PHP file called `index.php` in `XAMPP/htdocs/ch2`:

```
<?php
error_reporting( E_ALL );
ini_set( "display_errors", 1 );

$title = "Test title";
$content = "<h1>Hello World</h1>";
$page = "
<!DOCTYPE html>
<html>
<head>
<title>$title</title>
<meta http-equiv='Content-Type' content='text/html;charset=utf-8'/>
</head>
<body>
$content
</body>
</html>";
echo $page;
```

If you save and load `http://localhost/ch2/index.php` in your browser, you should see a well-formed HTML5 page with a title and a heading. It's a good habit to inspect the source code of your PHP-generated HTML pages. Do it, and you should see that the variables have been replaced by their corresponding values by PHP. The HTML source code should look like the following:

```
<!DOCTYPE html>
<html>
<head>
<title>Test title</title>
<meta http-equiv='Content-Type' content='text/html;charset=utf-8' />
</head>
<body>
Hello World</h1>
</body>
</html>
```

Including a Simple Page Template

Creating a valid HTML5 page with PHP is a very, very common task. You should have few problems understanding the preceding code. Let's try to create the same output in a way that's easier to reuse in other projects. If you can reuse your code in other projects, you can develop solutions faster and more efficient. Let's keep the HTML5 page template in a separate file.

Create a new folder called `templates` in your existing PHP project. Create a new PHP file called `page.php` in the templates folder, as follows:

```
<?php
return "<!DOCTYPE html>
<html>
<head>
<title>$title</title>
<meta http-equiv='Content-Type' content='text/html;charset=utf-8'/>
</head>
<body>
$content
</body>
</html>";
```

Returning Values

The `return` statement in PHP is very useful. It simply stops execution of the script. Any value indicated immediately after the `return` statement will be returned. In the preceding example, a valid HTML5 page will be returned.

Including the Template

To use the template from your index, you will have to load the script into PHP's memory. You can do that with another PHP statement: `include_once`. Update your `index.php` file, as follows:

```
<?php
//complete code for index.php
error_reporting( E_ALL );
ini_set( "display_errors", 1 );
$title = "Test title";
$content = "<h1>Hello World</h1>";
//indicate the relative path to the file to include
$page = include_once "templates/page.php";
echo $page;
```

The output of the preceding code will be identical to that you had when you first created the page. There are no functional changes, but there are some aesthetic changes in code architecture. A reusable page template is now kept in a separate file. The template is included into `index.php`, when needed. We're really splitting different parts of the code into different files. The result is that more of the code becomes readily reusable in other projects. This process of separating different parts is also known as *separation of concerns*.