Learning PHP 5

By David Sklar

Publisher: O'Reilly
Pub Date: June 2004
ISBN: 0-596-00560-1
Pages: 368

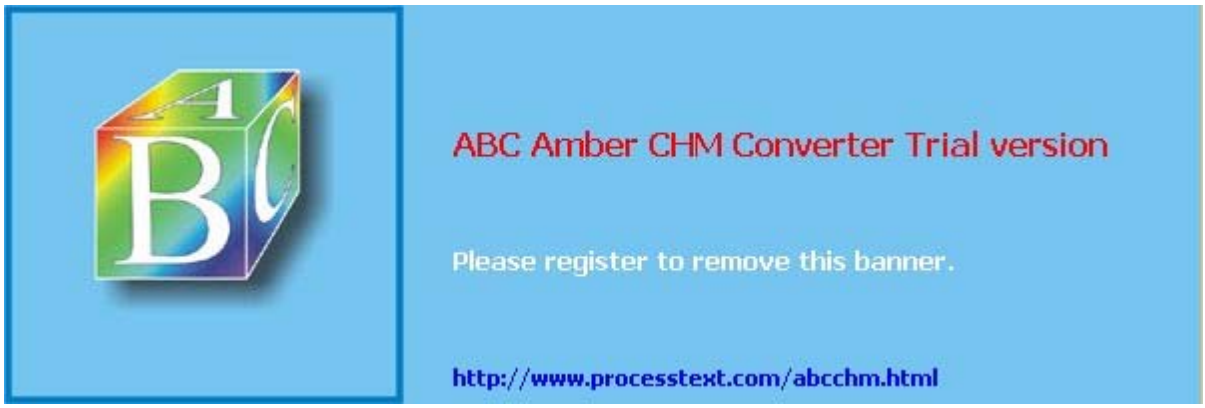
Learning PHP 5 is the ideal tutorial for graphic designers, bloggers, and other web crafters who want a thorough but non-intimidating way to understand the code that makes web sites dynamic. The book begins with an introduction to PHP, then moves to more advanced features: language basics, arrays and functions, web forms, connecting to databases, and much more. Complete with exercises to make sure the lessons stick, this book offers the ideal classroom learning experience whether you're in a classroom or on your own.

Learning PHP 5

By David Sklar

Publisher: O'Reilly
Pub Date: June 2004
ISBN: 0-596-00560-1
Pages: 368

PREV < Day Day Up > NEXT



Printed in the United States of America.

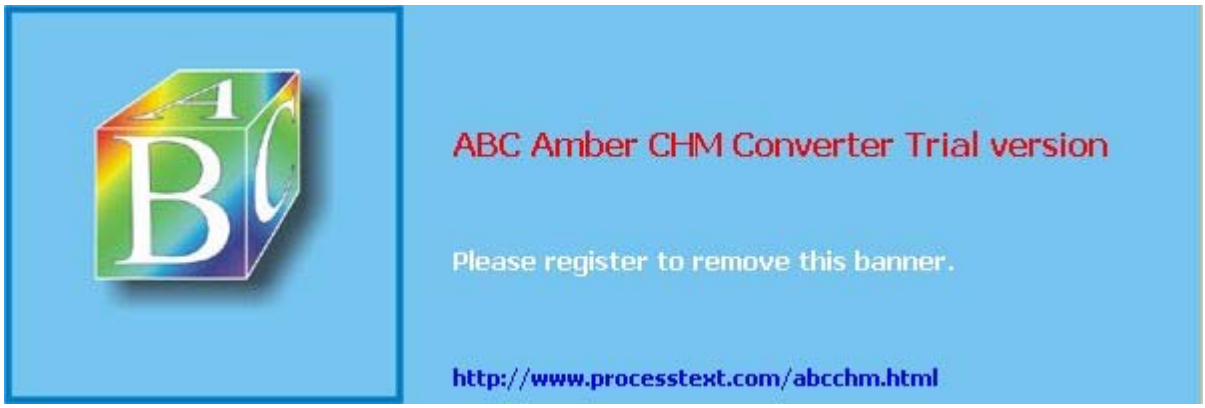
Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Dedication

To Jacob, who can look forward to so much learning.

PREV < Day Day Up > NEXT



Preface

Boring web sites are *static*. Interesting web sites are *dynamic*. That is, their content changes. A giant static HTML page listing the names, pictures, descriptions, and prices of all 1,000 products a company has for sale is hard to use and takes forever to load. A dynamic web product catalog that lets you search and filter those products so you see only the six items that meet your price and category criteria is more useful, faster, and much more likely to close a sale.

The PHP programming language makes it easy to build dynamic web sites. Whatever interactive excitement you want to create? such as a product catalog, a blog, a photo album, or an event calendar? PHP is up to the task. And after reading this book, you'll be up to the task of building that dynamic web site, too.

PREV < Day Day Up > NEXT

Who This Book Is For

This book is for:

- A hobbyist who wants to create an interactive web site for himself, his family, or a nonprofit organization.
- A web site builder who wants to use the PHP setup provided by an ISP or hosting provider.
- A small business owner who wants to put her company on the Web.
- A page designer who wants to communicate better with her developer co-workers.
- A JavaScript whiz who wants to build server-side programs that complement her client-side code.
- A blogger or HTML jockey who wants to easily add dynamic features to her site.
- A Perl, ASP, or ColdFusion programmer who wants to get up to speed with PHP.
- Anybody who wants a straightforward, jargon-free introduction to one of the most popular programming languages for building an interactive web site.

PHP's gentle learning curve and approachable syntax make it an ideal "gateway" language for the nontechnical web professional. *Learning PHP 5* is aimed at both this interested, intelligent, but not necessarily technical individual as well as at programmers familiar with another language who want to learn PHP.

Aside from basic computer literacy (knowing how to type, moving files around, surfing the Web), the only assumption that this book makes about you is that you're acquainted with HTML. You don't need to be an HTML master, but you should be comfortable with the HTML tags that populate a basic web page such as `<html>`, `<head>`, `<body>`, `<p>`, `<a>`, and `<br>`. If you're not familiar with HTML, read *HTML & XHTML: The Definitive Guide*, Fifth Edition, by Bill Kennedy and Chuck Musciano (O'Reilly).

Contents of This Book

This book is designed so that you start at the beginning and work through the chapters in order. For the most part, each chapter depends on material in the previous chapters. Chapter 2, through Chapter 12 and Appendix B, each end with exercises that test your understanding of the content in the chapter.

Chapter 1, provides some general background on PHP and how it interacts with your web browser and a web server. It also shows some PHP programs and what they do to give you an idea of what PHP programs look like. Especially if you're new to programming or building dynamic web sites, it is important to read Chapter 1.

The next four chapters give you a grounding in the fundamentals of PHP. Before you can write great literature, you need to learn a little grammar and some vocabulary. That's what these chapters are for. (Don't worry? you'll learn enough PHP grammar and vocabulary right away to start writing some short programs, if not great literature.) Chapter 2 shows you how to work with different kinds of data such as pieces of text and numbers. This is important because the web pages that your PHP programs generate are just big pieces of text. Chapter 3, describes the PHP commands with which your programs can make decisions. These decisions are at the heart of the "dynamic" in "dynamic web site." The concepts in Chapter 3 are what you use, for example, to display only items in a product catalog that fall between two prices a user enters in a web form.

Chapter 4, introduces *arrays*, which are collections of a bunch of individual numbers or pieces of text. Many frequent activities in PHP programs, such as processing submitted web form parameters or examining information pulled out of a database, involve using arrays. As you write more complicated programs, you'll find yourself wanting to repeat similar tasks. *Functions*, discussed in Chapter 5, help you reuse pieces of your programs.

The three chapters after that cover three essential tasks in building a dynamic web site: dealing with forms, databases, and users. Chapter 6, supplies the details on working with web forms. These are the primary way that users interact with your web site. Chapter 7, discusses databases. A database holds the information that your web site displays, such as a product catalog or event calendar. This chapter shows you how to make your PHP programs talk to a database. With the techniques in Chapter 8, your web site can do user-specific things such as display sensitive information to authorized people only or tell someone how many new message board posts have been created since she last logged in.

Then, the next three chapters examine three other areas you're likely to encounter when building your web site. Chapter 9, highlights the steps you need to take, for example, to display a monthly calendar or to allow users to input a date or time from a web form. Chapter 10, describes the PHP commands for interacting with files on your own computer or elsewhere on the Internet. Chapter 11, supplies the basics for dealing with XML documents in your PHP programs, whether you need to generate one for another program to consume or you've been provided with one to use in your own program.

Chapter 12 and Chapter 13 each stand on their own. Chapter 12, furnishes some approaches for understanding the error messages that the PHP interpreter generates and hunting down problems in your programs. While it partially depends on earlier material, it may be worthwhile to skip ahead and peruse Chapter 12 as you're working through the book.

Chapter 13 serves a taste of many additional capabilities of PHP, such as generating images, running code written in other languages, and making Flash movies. After you've gotten comfortable with the core PHP concepts explained in Chapter 1 through Chapter 12, visit Chapter 13 for lots of new things to learn.

The three appendixes provide supplementary material. To run PHP programs, you need to have a copy of the PHP interpreter installed on your computer (or have an account with a web-hosting provider that supports PHP). Appendix A, helps you get up and running, whether you are using Windows, OS X, or Linux.

Many text-processing tasks in PHP, such as validating submitted form parameters or parsing an HTML document, are made easier by using *regular expressions*, a powerful but initially inscrutable pattern matching syntax. Appendix B, explains the basics of regular expressions so that you can use them in your programs if you choose.

Other Resources

The online annotated PHP Manual (http://www.php.net/manual) is a great resource for exploring PHP's extensive function library. Plenty of user-contributed comments offer helpful advice and sample code, too. Additionally, there are many PHP mailing lists covering installation, programming, extending PHP, and various other topics. You can learn about and subscribe to these mailing lists at http://www.php.net/mailing-lists.php. A read-only web interface to the mailing lists is at http://news.php.net. Also worth exploring is the PHP Presentation System archive at http://talks.php.net. This is a collection of presentations about PHP that have been delivered at various conferences.

After you're comfortable with the material in this book, the following books about PHP are good next steps:

- *Programming PHP*, by Rasmus Lerdorf and Kevin Tatroe (O'Reilly). A more detailed and technical look at how to write PHP programs. Includes information on generating graphics and PDFs.
- *PHP Cookbook*, by David Sklar and Adam Trachtenberg (O'Reilly). A comprehensive collection of common PHP programming problems and their solutions.
- *Essential PHP Tools*, by David Sklar (Apress). Examples and explanations about many popular PHP add-on libraries and modules including HTML_QuickForm, SOAP, and the Smarty templating system.
- *Upgrading to PHP 5*, by Adam Trachtenberg (O'Reilly). A comprehensive look at the new features of PHP 5, including coverage of features for XML handling and object-oriented programming.

These books are helpful for learning about databases, SQL, and MySQL:

- *Web Database Applications with PHP & MySQL*, by David Lane and Hugh E. Williams (O'Reilly). How to make PHP and MySQL sing in harmony to make a robust dynamic web site.
- *SQL in a Nutshell*, by Kevin E. Kline (O'Reilly). The essentials you need to know to write SQL queries. Covers the SQL dialects used by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL.
- *MySQL Cookbook*, by Paul DuBois (O'Reilly). A comprehensive collection of common MySQL tasks.
- *MySQL Reference Manual* (http://dev.mysql.com/doc/mysql). The ultimate source for information about MySQL's features and SQL dialect.

These books are helpful for learning about HTML and HTTP:

- *HTML & XHTML: The Definitive Guide*, by Bill Kennedy and Chuck Musciano (O'Reilly). If you've got a question about HTML, this book answers it.
- *Dynamic HTML: The Definitive Reference*, by Danny Goodman (O'Reilly). Full of useful information you need if you're using JavaScript or Dynamic HTML as part of the web pages your PHP programs output.
- *HTTP Developer's Handbook*, by Chris Shiflett (Sams Publishing). With this book, you'll better understand how your web browser and a web server communicate with each other.

These books are helpful for learning about security and cryptography:

- *Web Security, Privacy & Commerce*, by Simson Garfinkel (O'Reilly). A readable and complete overview of the various aspects of web-related security and privacy.
- *Practical Unix & Internet Security*, by Simson Garfinkel, Alan Schwartz, and Gene Spafford (O'Reilly). A classic exploration of all facets of computer security.
- *Applied Cryptography*, by Bruce Schneier (John Wiley & Sons). The nitty-gritty on how

Conventions Used in This Book

The following programming and typesetting conventions are used in this book.

Programming Conventions

The code examples in this book are designed to work with PHP 5.0.0. They were tested with PHP 5.0.0RC2, which was the most up-to-date version of PHP 5 available at the time of publication. Almost all of the code in the book works with PHP 4.3 as well. The PHP 5-specific features discussed in the book are as follows:

- Chapter 7: the mysqli functions
- Chapter 10: the `file_put_contents( )` function
- Chapter 11: the SimpleXML module
- Chapter 12: the `E_STRICT` error-reporting level
- Chapter 13: some new features related to classes and objects, the advanced XML processing functions, the bundled SQLite database, and the Perl extension

Typographical Conventions

The following typographical conventions are used in this book:

Italic

Indicates new terms, example URLs, example email addresses, filenames, file extensions, pathnames, and directories.

`Constant width`

Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

`Constant width italic`

Shows text that should be replaced with user-supplied values.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

Typing some of the example programs in the book yourself is instructive when you are getting started. However, if your fingers get weary, you can download all of the code examples from http://www.oreilly.com/catalog/learnphp5.

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact the publisher for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Learning PHP 5* by David Sklar Copyright 2004 O'Reilly Media, Inc., 0-596-00560-1." If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact the publisher at permissions@oreilly.com.

PREV < Day Day Up > NEXT

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway
North Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

There is a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/learnphp5

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

Or you can contact the author directly via his web site:

http://www.sklar.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Acknowledgments

This book is the end result of the hard work of many people. Thank you to:

- The many programmers, testers, documentation writers, bug fixers, and other folks whose time, talent, and devotion have made PHP the first-class development platform that it is today. Without them, I'd have nothing to write about.
- The Apple WWPM Hardware Placement Lab for the loan of an iBook, and to Adam Trachtenberg, George Schlossnagle, and Jeremy Zawodny for advice on some code examples.
- My diligent reviewers: Griffin Cherry, Florence Leroy, Mark Oglia, and Stewart Ugelow. They caught plenty of mistakes, turned confusing explanations into clear ones, and otherwise made this book far better than it would have been without them.
- Robert Romano, who turned my blocky diagrams and rustic pencil sketches into high-quality figures and illustrations.
- Tatiana Diaz, who funneled all of my random questions to the right people, kept me on schedule, and ultimately made sure that whatever needed to get done, was done.
- Nat Torkington, whose editorial guidance and helpful suggestions improved every part of the book. Without Nat's feedback, this book would be twice as long and half as readable as it is.

For a better fate than wisdom, thank you also to Susannah, with whom I enjoy ignoring the syntax of things.

Chapter 1. Orientation and First Steps

There are lots of great reasons to write computer programs in PHP. Maybe you want to learn PHP because you need to put together a small web site for yourself that has some interactive elements. Perhaps PHP is being used where you work and you have to get up to speed. This chapter provides context for how PHP fits into the puzzle of web site construction: what it can do and why it's so good at what it does. You'll also get your first look at the PHP language and see it in action.

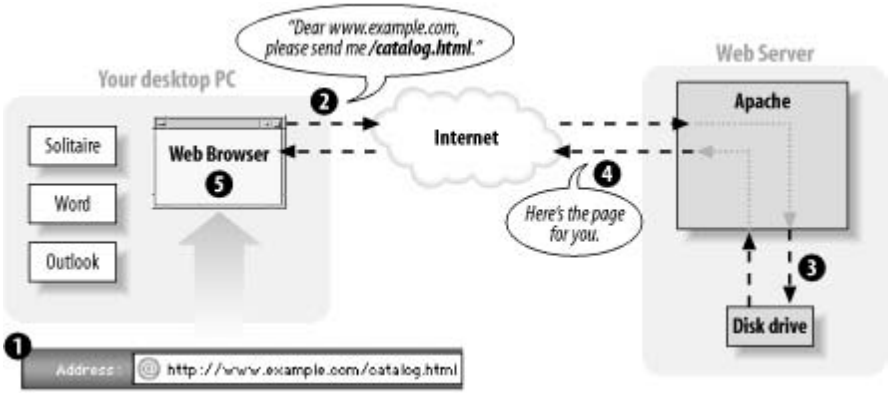
PREV < Day Day Up > NEXT

1.1 PHP's Place in the Web World

PHP is a programming language that's used mostly for building web sites. Instead of a PHP program running on a desktop computer for the use of one person, it typically runs on a web server and is accessed by lots of people using web browsers on their own computers. This section explains how PHP fits into the interaction between a web browser and a web server.

When you sit down at your computer and pull up a web page using a browser such as Internet Explorer or Mozilla, you cause a little conversation to happen over the Internet between your computer and another computer. This conversation and how it makes a web page appear on your screen is illustrated in Figure 1-1.

Figure 1-1. Client and server communication without PHP



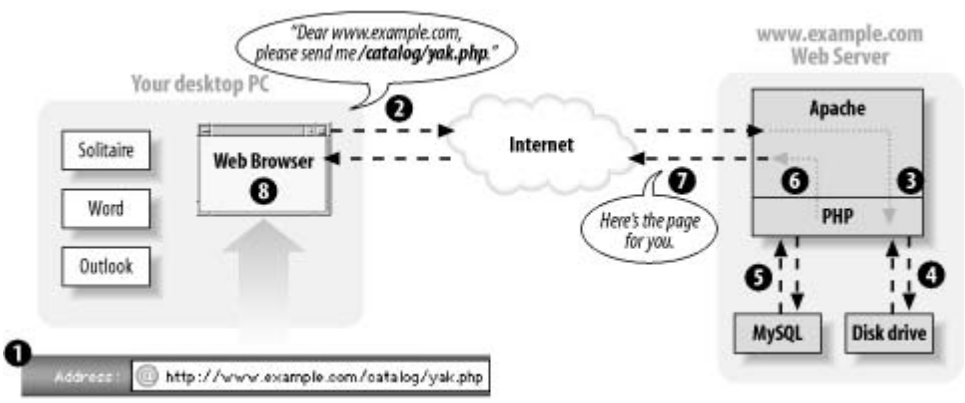
Here's what's happening in the numbered steps of the diagram:

1. You type *www.example.com/catalog.html* into the location bar of Internet Explorer.
2. Internet Explorer sends a message over the Internet to the computer named `www.example.com` asking for the */catalog.html* page.
3. Apache, a program running on the `www.example.com` computer, gets the message and reads the *catalog.html* file from the disk drive.
4. Apache sends the contents of the file back to your computer over the Internet as a response to Internet Explorer's request.
5. Internet Explorer displays the page on the screen, following the instructions of the HTML tags in the page.

Every time a browser asks for *http://www.example.com/catalog.html*, the web server sends back the contents of the same *catalog.html* file. The only time the response from the web server changes is if someone edits the file on the server.

When PHP is involved, however, the server does more work for its half of the conversation. Figure 1-2 shows what happens when a web browser asks for a page that is generated by PHP.

Figure 1-2. Client and server communication with PHP



1.2 What's So Great About PHP?

You may be attracted to PHP because it's free, because it's easy to learn, or because your boss told you that you need to start working on a PHP project next week. Since you're going to use PHP, you need to know a little bit about what makes it special. The next time someone asks you "What's so great about PHP?", use this section as the basis for your answer.

1.2.1 PHP Is Free (as in Money)

You don't have to pay anyone to use PHP. Whether you run the PHP interpreter on a beat-up 10-year-old PC in your basement or in a room full of million-dollar "enterprise-class" servers, there are no licensing fees, support fees, maintenance fees, upgrade fees, or any other kind of charge.

Most Linux distributions come with PHP already installed. If yours doesn't, or you are using another operating system such as Windows, you can download PHP from http://www.php.net/. Appendix A has detailed instructions on how to install PHP.

1.2.2 PHP Is Free (as in Speech)

As an open source project, PHP makes its innards available for anyone to inspect. If it doesn't do what you want, or you're just curious about why a feature works the way it does, you can poke around in the guts of the PHP interpreter (written in the C programming language) to see what's what. Even if you don't have the technical expertise to do that, you can get someone who does to do the investigating for you. Most people can't fix their own cars, but it's nice to be able to take your car to a mechanic who can pop open the hood and fix it.

1.2.3 PHP Is Cross-Platform

You can use PHP with a web server computer that runs Windows, Mac OS X, Linux, Solaris, and many other versions of Unix. Plus, if you switch web server operating systems, you generally don't have to change any of your PHP programs. Just copy them from your Windows server to your Unix server, and they will still work.

While Apache is the most popular web server program used with PHP, you can also use Microsoft Internet Information Server and any other web server that supports the CGI standard. PHP also works with a large number of databases including MySQL, Oracle, Microsoft SQL Server, Sybase, and PostgreSQL. In addition, it supports the ODBC standard for database interaction.

If all the acronyms in the last paragraph freak you out, don't worry. It boils down to this: whatever system you're using, PHP probably runs on it just fine and works with whatever database you are already using.

1.2.4 PHP Is Widely Used

As of March 2004, PHP is installed on more than 15 million different web sites, from countless tiny personal home pages to giants like Yahoo!. There are many books, magazines, and web sites devoted to teaching PHP and exploring what you can do with it. There are companies that provide support and training for PHP. In short, if you are a PHP user, you are not alone.

1.2.5 PHP Hides Its Complexity

You can build powerful e-commerce engines in PHP that handle millions of customers. You can also build a small site that automatically maintains links to a changing list of articles or press releases. When you're using PHP for a simpler project, it doesn't get in your way with concerns that are only relevant in a massive system. When you need advanced features such as caching, custom libraries, or dynamic image generation, they are available. If you don't need them, you don't have to worry about them. You can just focus on the basics of handling user input and displaying output.

1.2.6 PHP Is Built for Web Programming

Unlike most other programming languages, PHP was created from the ground up for generating

1.3 PHP in Action

Ready for your first taste of PHP? This section contains a few program listings and explanations of what they do. If you don't understand everything going on in each listing, don't worry! That's what the rest of the book is for. Read these listings to get a sense of what PHP programs look like and an outline of how they work. Don't sweat the details yet.

When given a program to run, the PHP interpreter pays attention only to the parts of the program between PHP start and end tags. Whatever's outside those tags is printed with no modification. This makes it easy to embed small bits of PHP in pages that mostly contain HTML. The PHP interpreter runs the commands between `<?php` (the PHP start tag) and `?>` (the PHP end tag). PHP pages typically live in files whose names end in *.php*. Example 1-1 shows a page with one PHP command.

Example 1-1. Hello, World!

```
<html>
<head><title>PHP says hello</title></head>
<body>
<b>
<?php
print "Hello, World!";
?>
</b>
</body>
</html>
```

The output of Example 1-1 is:

```
<html>
<head><title>PHP says hello</title></head>
<body>
<b>
Hello, World!
</b>
</body>
</html>
```

In your web browser, this looks like Figure 1-3.

Figure 1-3. Saying hello with PHP

Printing a message that never changes is not a very exciting use of PHP, however. You could have included the "Hello, World!" message in a plain HTML page with the same result. More useful is printing dynamic data ? i.e., information that changes. One of the most common sources of information for PHP programs is the user: the browser displays a form, the user enters information into that and hits the "submit" button, the browser sends that information to the server, and the server finally passes it on to the PHP interpreter where it is available to your program.

Example 1-2 is an HTML form with no PHP. The form consists simply of a text box named `user` and a Submit button. The form submits to *sayhello.php*, specified via the `<form>` tag's `action` attribute.

Example 1-2. HTML form for submitting data

```
<form method="POST" action="sayhello.php">
```

1.4 Basic Rules of PHP Programs

This section lays out some ground rules about the structure of PHP programs. More foundational than the basics such as "how do I print something" or "how do I add two numbers", these proto-basics are the equivalent of someone telling you that you should read pages in this book from top to bottom and left to right, or that what's important on the page are the black squiggles, not the large white areas.

If you've had a little experience with PHP already or you're the kind of person that prefers playing with all the buttons on your new DVD player before going back and reading in the manual about how the buttons actually work, feel free to skip ahead to Chapter 2 now and flip back here later. If you forge ahead to write some PHP programs of your own, and they're behaving unexpectedly or the PHP interpreter complains of "parse errors" when it tries to run your program, revisit this section for a refresher.

1.4.1 Start and End Tags

Each of the examples you've already seen in this chapter uses `<?php` as the PHP start tag and `?>` as the PHP end tag. The PHP interpreter ignores anything outside of those tags. Text before the start tag or after the end tag is printed with no interference from the PHP interpreter.

A PHP program can have multiple start and end tag pairs, as shown in Example 1-8.

Example 1-8. Multiple start and end tags

```
Five plus five is:
<?php print 5 + 5; ?>
<p>
Four plus four is:
<?php
 print 4 + 4;
?>
<p>
<img src="vacation.jpg" alt="My Vacation">
```

The PHP source code inside each set of `<?php ?>` tags is processed by the PHP interpreter, and the rest of the page is printed as is. Example 1-8 prints:

```
Five plus five is:
10<p>
Four plus four is:
8<p>
<img src="vacation.jpg" alt="My Vacation">
```

Some older PHP programs use `<?` as a start tag instead of `<?php`. The `<?` is called the *short open tag*, since it's shorter than `<?php`. It's usually better to use the regular `<?php` open tag since it's guaranteed to work on any server running the PHP interpreter. The short tag can be turned on or off with a PHP configuration setting. Appendix A shows you how to modify your PHP configuration to control which open tags are valid in your programs.

The rest of the examples in this chapter all begin with the `<?php` start tag and end with `?>`. In subsequent chapters, not all the examples have start and end tags ? but remember, your programs need them for the PHP interpreter to recognize your code.

1.4.2 Whitespace and Case-Sensitivity

Like all PHP programs, the examples in this section consist of a series of statements, each of which end with a semicolon. You can put multiple PHP statements on the same line of a program as long as they are separated with a semicolon. You can put as many blank lines between statements as you want. The PHP interpreter ignores them. The semicolon tells the interpreter that one statement is over and another is about to begin. No whitespace at all or lots and lots of whitespace between statements doesn't affect the program's execution. (*Whitespace* is programmer-speak for blank-looking characters such as space, tab, and

1.5 Chapter Summary

Chapter 1 covers:

- PHP's usage by a web server to create a response or document to send back to the browser.
- PHP as a server-side language, meaning it runs on the web server. This is in contrast to a client-side language such as JavaScript.
- What you sign up for when you decide to use PHP: it's free (in terms of money and speech), cross-platform, popular, and designed for web programming.
- How PHP programs that print information, process forms, and talk to a database appear.
- Some basics of the structure of PHP programs, such as the PHP start and end tags (`<?php` and `?>`), whitespace, case-sensitivity, and comments.

PREV < Day Day Up > NEXT

Chapter 2. Working with Text and Numbers

PHP can work with different types of data. In this chapter, you'll learn about individual values such as numbers and single pieces of text. You'll learn how to put text and numbers in your programs, as well as some of the limitations the PHP interpreter puts on those values and some common tricks for manipulating them.

Most PHP programs spend a lot of time handling text because they spend a lot of time generating HTML and working with information in a database. HTML is just a specially formatted kind of text, and information in a database, such as a username, a product description, or an address is a piece of text, too. Slicing and dicing text easily means you can build dynamic web pages easily.

In Chapter 1, you saw variables in action, but this chapter teaches you more about them. A variable is a named container that holds a value. The value that a variable holds can change as a program runs. When you access data submitted from a form or exchange data with a database, you use variables. In real life, a variable is something such as your checking account balance. As time goes on, the value that the phrase "checking account balance" refers to fluctuates. In a PHP program, a variable might hold the value of a submitted form parameter. Each time the program runs, the value of the submitted form parameter can be different. But whatever the value, you can always refer to it by the same name. This chapter also explains in more detail what variables are: how you create them and do things such as change their values or print them.

2.1 Text

When they're used in computer programs, pieces of text are called *strings*. This is because they consist of individual characters, strung together. Strings can contain letters, numbers, punctuation, spaces, tabs, or any other characters. Some examples of strings are `I would like 1 bowl of soup`, and `"Is it too hot?" he asked`, and `There's no spoon!`. A string can even contain the contents of a binary file such as an image or a sound. The only limit to the length of a string in a PHP program is the amount of memory your computer has.

2.1.1 Defining Text Strings

There are a few ways to indicate a string in a PHP program. The simplest is to surround the string with single quotes:

```
print 'I would like a bowl of soup.';
print 'chicken';
print '06520';
print '"I am eating dinner," he growled.';
```

Since the string consists of everything inside the single quotes, that's what is printed:

```
I would like a bowl of soup.chicken06520"I am eating dinner," he growled.
```

The output of those four `print` statements appears all on one line. No linebreaks are added by `print`.[1]

[1] You may also see `echo` used in some PHP programs to print text. It works just like `print`.

The single quotes aren't part of the string. They are *delimiters*, which tell the PHP interpreter where the start and end of the string is. If you want to include a single quote inside a string surrounded with single quotes, put a backslash (`\`) before the single quote inside the string:

```
print 'We\'ll each have a bowl of soup.';
```

The `\'` sequence is turned into ' inside the string, so what is printed is:

```
We'll each have a bowl of soup.
```

The backslash tells the PHP interpreter to treat the following character as a literal single quote instead of the single quote that means "end of string." This is called *escaping*, and the backslash is called the *escape character*. An escape character tells the system to do something special with the character that comes after it. Inside a single-quoted string, a single quote usually means "end of string." Preceding the single quote with a backslash changes its meaning to a literal single quote character.

Curly Quotes and Text Editors

Word processors often automatically turn straight quotes like ' and " into curly quotes like ‘, ’, “, and ”. The PHP interpreter only understands straight quotes as string delimiters. If you're writing PHP programs in a word processor or text editor that puts curly quotes in your programs, you have two choices: tell your word processor to stop it or use a different one. A program such as emacs, vi, BBEdit, or Windows Notepad leaves your quotes alone.

The escape character can itself be escaped. To include a literal backslash character in a string, put a back slash before it:

```
print 'Use a \\ to escape in a string';
```

This prints:

```
Use a \ to escape in a string
```

2.2 Numbers

Numbers in PHP are expressed using familiar notation, although you can't use commas or any other characters to group thousands. You don't have to do anything special to use a number with a decimal part as compared to an integer. Example 2-15 lists some valid numbers in PHP.

Example 2-15. Numbers

```
print 56;
print 56.3;
print 56.30;
print 0.774422;
print 16777.216;
print 0;
print -213;
print 1298317;
print -9912111;
print -12.52222;
print 0.00;
```

2.2.1 Using Different Kinds of Numbers

Internally, the PHP interpreter makes a distinction between numbers with a decimal part and those without one. The former are called *floating-point* numbers and the latter are called *integers*. Floating-point numbers take their name from the fact that the decimal point can "float" around to represent different amounts of precision.

The PHP interpreter uses the math facilities of your operating system to represent numbers so the largest and smallest numbers you can use, as well as the number of decimal places you can have in a floating-point number, vary on different systems.

One distinction between the PHP interpreter's internal representation of integers and floating-point numbers is the exactness of how they're stored. The integer 47 is stored as exactly 47. The floating-point number 46.3 could be stored as 46.2999999. This affects the correct technique of how to compare numbers. Section 3.3 explains comparisons and shows how to properly compare floating-point numbers.

2.2.2 Arithmetic Operators

Doing math in PHP is a lot like doing math in elementary school, except it's much faster. Some basic operations between numbers are shown in Example 2-16.

Example 2-16. Math operations

```
print 2 + 2;
print 17 - 3.5;
print 10 / 3;
print 6 * 9;
```

The output of Example 2-16 is:

```
4
13.5
3.3333333333333
54
```

In addition to the plus sign (`+`) for addition, the minus sign (`-`) for subtraction, the forward slash (`/`) for division, and the asterisk (`*`) for multiplication, PHP also supports the percent sign (`%`) for modulus division. This returns the remainder of a division operation:

```
print 17 % 3;
```

This prints:

```
2
```

2.3 Variables

Variables hold the data that your program manipulates while it runs, such as information about a user that you've loaded from a database or entries that have been typed into an HTML form. In PHP, variables are denoted by `$` followed by the variable's name. To assign a value to a variable, use an equals sign (`=`). This is known as the assignment operator.

```
$plates = 5;
$dinner = 'Beef Chow-Fun';
$cost_of_dinner = 8.95;
$cost_of_lunch = $cost_of_dinner;
```

Assignment works with here documents as well:

```
$page_header = <<<HTML_HEADER
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
HTML_HEADER;

$page_footer = <<<HTML_FOOTER
</body>
</html>
HTML_FOOTER;
```

Variable names must begin with letter or an underscore. The rest of the characters in the variable name may be letters, numbers, or an underscore. Table 2-2 lists some acceptable variable names.

Table 2-2. Acceptable variable names

Acceptable
`$size`
`$drinkSize`
`$my_drink_size`
`$_drinks`
`$drink4you2`

Table 2-3 lists some unacceptable variable names and what's wrong with them.

Table 2-3. Unacceptable variable names

Variable name	Flaw
`$2hot4u`	Begins with a number
`$drink-size`	Unacceptable character: `-`
`$drinkmaster@example.com`	Unacceptable characters: `@` and .
`$drink'lots`	Unacceptable character: '

2.4 Chapter Summary

Chapter 2 covers:

- Defining strings in your programs three different ways: with single quotes, with double quotes, and as a here document.
- Escaping: what it is and what characters need to be escaped in each kind of string.
- Validating a string by checking its length, removing leading and trailing whitespace from it, or comparing it to another string.
- Formatting a string with `printf( )`.
- Manipulating the case of a string with `strtolower( )`, `strtoupper( )`, or `ucwords( )`.
- Selecting part of a string with `substr( )`.
- Changing part of a string with `str_replace( )`.
- Defining numbers in your programs.
- Doing math with numbers.
- Storing values in variables.
- Naming variables appropriately.
- Using combined operators with variables.
- Using increment and decrement operators with variables.
- Interpolating variables in strings.

2.5 Exercises

1. Find the errors in this PHP program:

```
<? php
print 'How are you?';
print 'I'm fine.';
??>
```

5. Write a PHP program that computes the total cost of this restaurant meal: two hamburgers at $4.95 each, one chocolate milk shake at $1.95, and one cola at 85 cents. The sales tax rate is 7.5%, and you left a pre-tax tip of 16%.

6. Modify your solution to the previous exercise to print out a formatted bill. For each item in the meal, print the price, quantity, and total cost. Print the pre-tax food and drink total, the post-tax total, and the total with tax and tip. Make sure that prices in your output are vertically aligned.

7. Write a PHP program that sets the variable `$first_name` to your first name and `$last_name` to your last name. Print out a string containing your first and last name separated by a space. Also print out the length of that string.

8. Write a PHP program that uses the increment operator (`++`) and the combined multiplication operator (`*=`) to print out the numbers from 1 to 5 and powers of 2 from 2 (2^1) to 32 (2^5).

9. Add comments to the PHP programs you've written for the other exercises. Try both single and multiline comments. After you've added the comments, run the programs to make sure they work properly and your comment syntax is correct.

Chapter 3. Making Decisions and Repeating Yourself

Chapter 2 covered the basics of how to represent data in PHP programs. A program full of data is only half complete, though. The other piece of the puzzle is using that data to control how the program runs, taking actions such as:

- If an administrative user is logged in, print a special menu.
- Print a different page header if it's after three o'clock.
- Notify a user if new messages have been posted since she last logged in.

All of these actions have something in common: they make decisions about whether a certain logical condition involving data is true or false. In the first action, the logical condition is "Is an administrative user logged in?" If the condition is true (yes, an administrative user is logged in), then a special menu is printed. The same kind of thing happens in the next example. If the condition "is it after three o'clock?" is true, then a different page header is printed. Likewise, if "Have new messages been posted since the user last logged in?" is true, then the user is notified.

When making decisions, the PHP interpreter boils down an expression into `true` or `false`. Section 3.1 explains how the interpreter decides which expressions and values are `true` and which are `false`.

Those `true` and `false` values are used by language constructs such as `if( )` to decide whether to run certain statements in a program. The ins and outs of `if( )` are detailed later in this chapter in Section 3.2. Use `if( )` and similar constructs any time the outcome of a program depends on some changing conditions.

While `true` and `false` are the cornerstones of decision making, usually you want to ask more complicated questions, such as "is this user at least 21 years old?" or "does this user have a monthly subscription to the web site or enough money in their account to buy a daily pass?" Section 3.3, later in this chapter, explains PHP's comparison and logical operators. These help you express whatever kind of decision you need to make in a program, such as seeing whether numbers or strings are greater than or less than each other. You can also chain together decisions into a larger decision that depends on its pieces.

Decision making is also used in programs when you want to repeatedly execute certain statements ? you need a way to indicate when the repetition should stop. Frequently, this is determined by a simple counter, such as "repeat 10 times." This is like asking the question "Have I repeated 10 times yet?" If so, then the program continues. If not, the action is repeated again. Determining when to stop can be more complicated, too ? for example, "show another math question to a student until 6 questions have been answered correctly." Section 3.4, later in this chapter, introduces PHP's `while( )` and `for( )` constructs, with which you can implement these kinds of loops.

3.1 Understanding true and false

Every expression in a PHP program has a truth value: `true` or `false`. Sometimes that truth value is important because you use it in a calculation, but sometimes you ignore it. Understanding how expressions evaluate to `true` or to `false` is an important part of understanding PHP.

Most scalar values are `true`. All integers and floating-point numbers (except for 0 and 0.0) are `true`. All strings are `true` except for two: a string containing nothing at all and a string containing only the character `0`. These four values are `false`. The special constant `false` also evaluates to `false`. Everything else is `true`.[1]

[1] An empty array is also `false`. This is discussed in Chapter 4.

A variable equal to one of the five `false` values, or a function that returns one of those values also evaluates to `false`. Every other expression evaluates to `true`.

Figuring out the truth value of an expression has two steps. First, figure out the actual value of the expression. Then, check whether that value is `true` or `false`. Some expressions have common sense values. The value of a mathematical expression is what you'd get by doing the math with paper and pencil. For example, `7 * 6` equals `42`. Since 42 is `true`, the expression `7 * 6` is `true`. The expression `5 - 6 + 1` equals `0`. Since `0` is `false`, the expression `5 - 6 + 1` is `false`.

The same is true with string concatenation. The value of an expression that concatenates two strings is the new, combined string. The expression '`jacob`' . '`@example.com`' equals the string `jacob@example.com`, which is `true`.

The value of an assignment operation is the value being assigned. The expression `$price = 5` evaluates to `5`, since that's what's being assigned to `$price`. Because assignment produces a result, you can chain assignment operations together to assign the same value to multiple variables:

```
$price = $quantity = 5;
```

This expression means "set `$price` equal to the result of setting `$quantity` equal to `5`." When this expression is evaluated, the integer `5` is assigned to the variable `$quantity`. The result of that assignment expression is `5`, the value being assigned. Then, that result (`5`) is assigned to the variable `$price`. Both `$price` and `$quantity` are set to `5`.

3.2 Making Decisions

With the `if( )` construct, you can have statements in your program that are only run if certain conditions are `true`. This lets your program take different actions depending on the circumstances. For example, you can check that a user has entered valid information in a web form before letting her see sensitive data.

The `if( )` construct runs a block of code if its test expression is `true`. This is demonstrated in Example 3-1.

Example 3-1. Making a decision with if( )

```
if ($logged_in) {
   print "Welcome aboard, trusted user.";
}
```

The `if( )` construct finds the truth value of the expression inside its parentheses (the *test expression*). If the expression evaluates to `true`, then the statements inside the curly braces after the `if( )` are run. If the expression isn't `true`, then the program continues with the statements after the curly braces. In this case, the test expression is just the variable `$logged_in`. If `$logged_in` is `true` (or has a value such as `5`, `-12.6`, or `Grass Carp`, that evaluates to `true`), then `Welcome aboard, trusted user`. is printed.

You can have as many statements as you want in the code block inside the curly braces. However, you need to terminate each of them with a semicolon. This is the same rule that applies to code outside an `if( )` statement. You don't, however, need a semicolon after the closing curly brace that encloses the code block. You also don't put a semicolon after the opening curly brace. Example 3-2 shows an `if( )` clause that runs multiple statements when its test expression is `true`.

Example 3-2. Multiple statements in an if( ) code block

```
print "This is always printed.";
if ($logged_in) {
    print "Welcome aboard, trusted user.";
    print 'This is only printed if $logged_in is true.';
}
print "This is also always printed.";
```

To run different statements when the `if( )` test expression is `false`, add an `else` clause to your `if( )` statement. This is shown in Example 3-3.

Example 3-3. Using else with if( )

```
if ($logged_in) {
    print "Welcome aboard, trusted user.";
} else {
    print "Howdy, stranger.";
}
```

In Example 3-3, the first `print` statement is only executed when the `if( )` test expression (the variable `$logged_in`) is `true`. The second `print` statement, inside the `else` clause, is only run when the test expression is `false`.

The `if( )` and `else` constructs are extended further with the `elseif( )` construct. You can pair one or more `elseif( )` clauses with an `if( )` to test multiple conditions separately. Example 3-4 demonstrates `elseif( )`.

Example 3-4. Using elseif( )

```
if ($logged_in) {
    // This runs if $logged_in is true
    print "Welcome aboard, trusted user.";
} elseif ($new_messages) {
```

3.3 Building Complicated Decisions

The comparison and logical operators in PHP help you put together more complicated expressions on which an `if( )` construct can decide. These operators let you compare values, negate values, and chain together multiple expressions inside one `if( )` statement.

The equality operator is `= =`. It returns `true` if the two values you test with it are equal. The values can be variables or literals. Some uses of the equality operator are shown in Example 3-6.

Example 3-6. The equality operator

```
if ($new_messages == 10) {
    print "You have ten new messages.";
}

if ($new_messages == $max_messages) {
    print "You have the maximum number of messages.";
}

if ($dinner == 'Braised Scallops') {
    print "Yum! I love seafood.";
}
```

The opposite of the equality operator is `!=`. It returns `true` if the two values that you test with it are not equal. See Example 3-7.

Assignment Versus Comparison

Be careful not to use `=` when you mean `= =`. A single equals sign assigns a value and returns the value assigned. Two equals signs test for equality and return `true` if the values are equal. If you leave off the second equals sign, you usually get an `if( )` test that is always `true`, as in the following:

```
if ($new_messages = 12) {
    print "It seems you now have twelve new messages.";
}
```

Instead of testing whether `$new_messages` equals `12`, the code shown here sets `$new_messages` to `12`. This assignment returns `12`, the value being assigned. The `if( )` test expression is always `true`, no matter what the value of `$new_messages`. Additionally, the value of `$new_messages` is overwritten. One way to avoid using `=` instead of `= =` is to put the variable on the right side of the comparison and the literal on the left side, as in the following:

```
if (12 == $new_messages) {
    print "You have twelve new messages.";
}
```

The test expression above may look a little funny, but it gives you some insurance if you accidentally use `=` instead of `= =`. With one equals sign, the test expression is `12 = $new_messages`, which means "assign the value of `$new_messages` to `12`." This doesn't make any sense: you can't change the value of `12`. If the PHP interpreter sees this in your program, it reports a parse error and the program doesn't run. The parse error alerts you to the missing `=`. With the literal on the righthand side of the expression, the code is parseable by the interpreter, so it doesn't report an error.

Example 3-7. The not-equals operator

```
if ($new_messages != 10) {
    print "You don't have ten new messages.";
```

3.4 Repeating Yourself

When a computer program does something repeatedly, it's called *looping*. This happens a lot ? for example, when you want to retrieve a set of rows from a database, print rows of an HTML table, or print elements in an HTML `<select>` menu. The two looping constructs discussed in this section are `while( )` and `for( )`. Their specifics differ but they each require you to specify the two essential attributes of any loop: what code to execute repeatedly and when to stop. The code to execute is a code block just like what goes inside the curly braces after an `if( )` construct. The condition for stopping the loop is a logical expression just like an `if( )` construct's test expression.

The `while( )` construct is like a repeating `if( )`. You provide an expression to `while( )`, just like to `if( )`. If the expression is `true`, then a code block is executed. Unlike `if( )`, however, `while( )` checks the expression again after executing the code block. If it's still `true`, then the code block is executed again (and again, and again, as long as the expression is `true`.) Once the expression is `false`, program execution continues with the lines after the code block. As you have probably guessed, your code block should do something that changes the outcome of the test expression so that the loop doesn't go on forever.

Example 3-16 uses `while( )` to print out an HTML form `<select>` menu with 10 choices.

Example 3-16. Printing a <select> menu with while( )

```
$i = 1;
print '<select name="people">';
while ($i <= 10) {
    print "<option>$i</option>\n";
    $i++;
}
print '</select>';
```

Example 3-16 prints:

```
<select name="people"><option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>6</option>
<option>7</option>
<option>8</option>
<option>9</option>
<option>10</option>
</select>
```

Before the `while( )` loop runs, the code sets `$i` to `1` and prints the opening `<select>` tag. The test expression compares `$i` to `10`. As long as `$i` is less than or equal to 10, the two statements in the code block are executed. The first prints out an `<option>` tag for the `<select>` menu, and the second increments `$i`. If you didn't increment `$i` inside the `while( )` loop, Example 3-16 would print out `<option>1</option>` forever.

After the code block prints `<option>10</option>`, the `$i++` line makes `$i` equal to 11. Then the test expression (`$i <= 10`) is evaluated. Since it's not `true` (11 is not less than or equal to 10), the program continues past the `while( )` loop's code block and prints out the closing `</select>` tag.

The `for( )` construct also provides a way for you to execute the same statements multiple times. Example 3-17 uses `for( )` to print out the same HTML form `<select>` menu as Example 3-16.

Example 3-17. Printing a <select> menu with for( )

```
print '<select name="people">';
for ($i = 1; $i <= 10; $i++) {
```

3.5 Chapter Summary

Chapter 3 covers:

- Evaluating an expression's truth value: `true` or `false`.
- Making a decision with `if( )`.
- Extending `if( )` with `else`.
- Extending `if( )` with `elseif( )`.
- Putting multiple statements inside an `if( )`, `elseif( )`, or `else` code block.
- Using the equality (`= =`) and not-equals (`!=`) operators in test expressions.
- Distinguishing between assignment (`=`) and equality comparison (`= =`).
- Using the less-than (`<`), greater-than (`>`), less-than-or-equal-to (`<=`), and greater-than-or-equal-to (`>=`) operators in test expressions.
- Comparing two floating-point numbers with `abs( )`.
- Comparing two strings with operators.
- Comparing two strings with `strcmp( )` or `strcasecmp( )`.
- Using the negation operator (`!`) in test expressions.
- Using the logical operators (`&&` and `||`) to build more complicated test expressions.
- Repeating a code block with `while( )`.
- Repeating a code block with `for( )`.

3.6 Exercises

1. Without using a PHP program to evaluate them, determine whether each of these expressions is `true` or `false`:

    a. `100.00 - 100`

    b. `"zero"`

    c. `"false"`

    d. `0 + "true"`

    e. `0.000`

    f. `"0.0"`

    g. `strcmp("false","False")`

2. Without running it through the PHP interpreter, figure out what this program prints.

```
$age = 12;
$shoe_size = 13;
if ($age > $shoe_size) {
    print "Message 1.";
} elseif (($shoe_size++) && ($age > 20)) {
    print "Message 2.";
} else {
    print "Message 3.";
}
print "Age: $age. Shoe Size: $shoe_size";
```

12. Use `while( )` to print out a table of Fahrenheit and Celsius temperature equivalents from -50 degrees F to 50 degrees F in 5-degree increments. On the Fahrenheit temperature scale, water freezes at 32 degrees and boils at 212 degrees. On the Celsius scale, water freezes at 0 degrees and boils at 100 degrees. So, to convert from Fahrenheit to Celsius, you subtract 32 from the temperature, multiply by 5, and divide by 9. To convert from Celsius to Fahrenheit, you multiply by 9, divide by 5, and then add 32.

13. Modify your answer to Exercise 3 to use `for( )` instead of `while( )`.

Chapter 4. Working with Arrays

Arrays are collections of related values, such as the data submitted from a form, the names of students in a class, or the populations of a list of cities. In Chapter 2, you learned that a variable is a named container that holds a value. An array is a container that holds multiple values, each distinct from the rest.

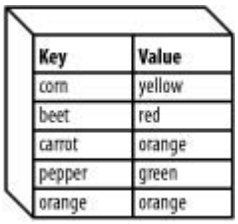
This chapter shows you how to work with arrays. Section 4.1, next, goes over fundamentals such as how to create arrays and manipulate their elements. Frequently, you'll want to do something with each element in an array, such as print it or inspect it for certain conditions. Section 4.2 explains how to do these things with the `foreach( )` and `for( )` constructs. Section 4.3 introduces the `implode( )` and `explode( )` functions, which turn arrays into strings and strings into arrays. Another kind of array modification is sorting, which is discussed in Section 4.4. Last, Section 4.5 explores arrays that themselves contain other arrays.

Chapter 6 shows you how to process form data, which the PHP interpreter automatically puts into an array for you. When you retrieve information from a database as described in Chapter 7, that data is often packaged into an array.

4.1 Array Basics

An array is made up of *elements*. Each element has a *key* and a *value*. An array holding information about the colors of vegetables has vegetable names for keys and colors for values, shown in Figure 4-1.

Figure 4-1. Keys and values



An array can only have one element with a given key. In the vegetable color array, there can't be another element with the key `corn` even if its value is `blue`. However, the same value can appear many times in one array. You can have orange carrots, orange tangerines, and orange oranges.

Any string or number value can be an array element key such as `corn`, `4`, `-36`, or `Salt Baked Squid`. Arrays and other nonscalar[1] values can't be keys, but they can be element values. An element value can be a string, a number, `true`, or `false`; it can also be another array.

[1] *Scalar* describes data that has a single value: a number, a piece of text, true, or false. Complex data types such as arrays, which hold multiple values, are not scalars.

4.1.1 Creating an Array

To create an array, assign a value to a particular array key. Array keys are denoted with square brackets, as shown in Example 4-1.

Example 4-1. Creating arrays

```
// An array called $vegetables with string keys
$vegetables['corn'] = 'yellow';
$vegetables['beet'] = 'red';
$vegetables['carrot'] = 'orange';

// An array called $dinner with numeric keys
$dinner[0] = 'Sweet Corn and Asparagus';
$dinner[1] = 'Lemon Chicken';
$dinner[2] = 'Braised Bamboo Fungus';

// An array called $computers with numeric and string keys
$computers['trs-80'] = 'Radio Shack';
$computers[2600] = 'Atari';
$computers['Adam'] = 'Coleco';
```

The array keys and values in Example 4-1 are strings (such as `corn`, `Braised Bamboo Fungus`, and `Coleco`) and numbers (such as `0`, `1`, and `2600`). They are written just like other strings and numbers in PHP programs: with quotes around the strings but not around the numbers.

You can also create an array using the `array( )` language construct. Example 4-2 creates the same arrays as Example 4-1.

Example 4-2. Creating arrays with array( )

```
$vegetables = array('corn' => 'yellow',
                    'beet' => 'red',
                    'carrot' => 'orange');

$dinner = array(0 => 'Sweet Corn and Asparagus',
                1 => 'Lemon Chicken',
```

4.2 Looping Through Arrays

One of the most common things to do with an array is to consider each element in the array individually and process it somehow. This may involve incorporating it into a row of an HTML table or adding its value to a running total.

The easiest way to iterate through each element of an array is with `foreach( )`. The `foreach( )` construct lets you run a code block once for each element in an array. Example 4-7 uses `foreach( )` to print an HTML table containing each element in an array.

Example 4-7. Looping with foreach( )

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');
print "<table>\n";
foreach ($meal as $key => $value) {
    print "<tr><td>$key</td><td>$value</td></tr>\n";
}
print '</table>';
```

Example 4-7 prints:

```
<table>
<tr><td>breakfast</td><td>Walnut Bun</td></tr>
<tr><td>lunch</td><td>Cashew Nuts and White Mushrooms</td></tr>
<tr><td>snack</td><td>Dried Mulberries</td></tr>
<tr><td>dinner</td><td>Eggplant with Chili Sauce</td></tr>
</table>
```

For each element in `$meal`, `foreach( )` copies the key of the element into `$key` and the value into `$value`. Then, it runs the code inside the curly braces. In Example 4-7, that code prints `$key` and `$value` with some HTML to make a table row. You can use whatever variable names you want for the key and value inside the code block. If the variable names were in use before the `foreach( )`, though, they're overwritten with values from the array.

When you're using `foreach( )` to print out data in an HTML table, often you want to apply alternating colors or styles to each table row. This is easy to do when you store the alternating color values in a separate array. Then, switch a variable between `0` and `1` each time through the `foreach( )` to print the appropriate color. Example 4-8 alternates between the two color values in its `$row_color` array.

Example 4-8. Alternating table row colors

```
$row_color = array('red','green');
$color_index = 0;
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');
print "<table>\n";
foreach ($meal as $key => $value) {
    print '<tr bgcolor="' . $row_color[$color_index] . '">';
    print "<td>$key</td><td>$value</td></tr>\n";
    // This switches $color_index between 0 and 1
    $color_index = 1 - $color_index;
}
print '</table>';
```

Example 4-8 prints:

```
<table>
<tr bgcolor="red"><td>breakfast</td><td>Walnut Bun</td></tr>
```

4.3 Modifying Arrays

You can operate on individual array elements just like regular scalar variables, using arithmetic, logical, and other operators. Example 4-17 shows some operations on array elements.

Example 4-17. Operating on array elements

```
$dishes['Beef Chow Foon'] = 12;
$dishes['Beef Chow Foon']++;
$dishes['Roast Duck'] = 3;

$dishes['total'] = $dishes['Beef Chow Foon'] + $dishes['Roast Duck'];

if ($dishes['total']> 15) {
    print "You ate a lot: ";
}

print 'You ate ' . $dishes['Beef Chow Foon'] . ' dishes of Beef Chow Foon.';
```

Example 4-17 prints:

```
You ate a lot: You ate 13 dishes of Beef Chow Foon.
```

Interpolating array element values in double-quoted strings or here documents is similar to interpolating numbers or strings. The easiest way is to include the array element in the string, but don't put quotes around the element key. This is shown in Example 4-18.

Example 4-18. Interpolating array element values in double-quoted strings

```
$meals['breakfast'] = 'Walnut Bun';
$meals['lunch'] = 'Eggplant with Chili Sauce';
$amounts = array(3, 6);

print "For breakfast, I'd like $meals[breakfast] and for lunch, ";
print "I'd like $meals[lunch]. I want $amounts[0] at breakfast and ";
print "$amounts[1] at lunch.";
```

Example 4-18 prints:

```
For breakfast, I'd like Walnut Bun and for lunch,
I'd like Eggplant with Chili Sauce. I want 3 at breakfast and
6 at lunch.
```

The interpolation in Example 4-18 works only with array keys that consist exclusively of letters, numbers, and underscores. If you have an array key that has whitespace or other punctuation in it, interpolate it with curly braces, as demonstrated in Example 4-19.

Example 4-19. Interpolating array element values with curly braces

```
$meals['Walnut Bun'] = '$3.95';
$hosts['www.example.com'] = 'web site';

print "A Walnut Bun costs {$meals['Walnut Bun']}.";
print "www.example.com is a {$hosts['www.example.com']}.";
```

Example 4-19 prints:

```
A Walnut Bun costs $3.95.
www.example.com is a web site.
```

In a double-quoted string or here document, an expression inside curly braces is evaluated and then its value is put into the string. In Example 4-19, the expressions used are lone array

4.4 Sorting Arrays

There are several ways to sort arrays. Which function to use depends on how you want to sort your array and what kind of array it is.

The `sort( )` function sorts an array by its element values. It should only be used on numeric arrays, because it resets the keys of the array when it sorts. Example 4-23 shows some arrays before and after sorting.

Example 4-23. Sorting with sort( )

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');

print "Before Sorting:\n";
foreach ($dinner as $key => $value) {
    print " \$dinner: $key $value\n";
}
foreach ($meal as $key => $value) {
    print "   \$meal: $key $value\n";
}

sort($dinner);
sort($meal);

print "After Sorting:\n";
foreach ($dinner as $key => $value) {
    print " \$dinner: $key $value\n";
}
foreach ($meal as $key => $value) {
    print "   \$meal: $key $value\n";
}
```

Example 4-23 prints:

```
Before Sorting:
 $dinner: 0 Sweet Corn and Asparagus
 $dinner: 1 Lemon Chicken
 $dinner: 2 Braised Bamboo Fungus
   $meal: breakfast Walnut Bun
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
After Sorting:
 $dinner: 0 Braised Bamboo Fungus
 $dinner: 1 Lemon Chicken
 $dinner: 2 Sweet Corn and Asparagus
   $meal: 0 Cashew Nuts and White Mushrooms
   $meal: 1 Dried Mulberries
   $meal: 2 Eggplant with Chili Sauce
   $meal: 3 Walnut Bun
```

Both arrays have been rearranged in ascending order by element value. The first value in `$dinner` is now `Braised Bamboo Fungus`, and the first value in `$meal` is `Cashew Nuts and White Mushrooms`. The keys in `$dinner` haven't changed because it was a numeric array before we sorted it. The keys in `$meal`, however, have been replaced by numbers from `0` to `3`.

To sort an associative array by element value, use `asort( )`. This keeps keys together with their values. Example 4-24 shows the `$meal` array from Example 4-23 sorted with `asort( )`.

4.5 Using Multidimensional Arrays

As mentioned earlier in Section 4.1, the value of an array element can be another array. This is useful when you want to store data that has a more complicated structure than just a key and a single value. A standard key/value pair is fine for matching up a meal name (such as `breakfast` or `lunch`) with a single dish (such as `Walnut Bun` or `Chicken withCashew Nuts`), but what about when each meal consists of more than one dish? Then, element values should be arrays, not strings.

Use the `array( )` construct to create arrays that have more arrays as element values, as shown in Example 4-27.

Example 4-27. Creating multidimensional arrays with array( )

```
$meals = array('breakfast' => array('Walnut Bun','Coffee'),
               'lunch'     => array('Cashew Nuts', 'White Mushrooms'),
               'snack'     => array('Dried Mulberries','Salted Sesame Crab'));

$lunches = array( array('Chicken','Eggplant','Rice'),
                  array('Beef','Scallions','Noodles'),
                  array('Eggplant','Tofu'));

$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese'  => array('hot' => 'mustard',
                                     'pepper-salty' => 'prickly ash'));
```

Access elements in these arrays of arrays by using more sets of square brackets to identify elements. Each set of square brackets goes one level into the entire array. Example 4-28 demonstrates how to access elements of the arrays defined in Example 4-27.

Example 4-28. Accessing multidimensional array elements

```
print $meals['lunch'][1];            // White Mushrooms
print $meals['snack'][0];            // Dried Mulberries
print $lunches[0][0];                // Chicken
print $lunches[2][1];                // Tofu
print $flavors['Japanese']['salty']  // soy sauce
print $flavors['Chinese']['hot'];    // mustard
```

Each level of an array is called a *dimension*. Before this section, all the arrays in this chapter are *one-dimensional arrays*. They each have one level of keys. Arrays such as `$meals`, `$lunches`, and `$flavors`, shown in Example 4-28, are called *multidimensional arrays* because they each have more than one dimension.

You can also create or modify multidimensional arrays with the square bracket syntax. Example 4-29 shows some multidimensional array manipulation.

Example 4-29. Manipulating multidimensional arrays

```
$prices['dinner']['Sweet Corn and Asparagus'] = 12.50;
$prices['lunch']['Cashew Nuts and White Mushrooms'] = 4.95;
$prices['dinner']['Braised Bamboo Fungus'] = 8.95;

$prices['dinner']['total'] = $prices['dinner']['Sweet Corn and Asparagus'] +
                             $prices['dinner']['Braised Bamboo Fungus'];

$specials[0][0] = 'Chestnut Bun';
$specials[0][1] = 'Walnut Bun';
$specials[0][2] = 'Peanut Bun';
$specials[1][0] = 'Chestnut Salad';
$specials[1][1] = 'Walnut Salad';
// Leaving out the index adds it to the end of the array
// This creates $specials[1][2]
$specials[1][] = 'Peanut Salad';
```

4.6 Chapter Summary

Chapter 4 covers:

- Understanding the components of an array: elements, keys, and values.
- Defining an array in your programs two ways: with `array( )` and with square brackets.
- Understanding the shortcuts PHP provides for arrays with numeric keys.
- Counting the number of elements in an array.
- Visiting each element of an array with `foreach( )`.
- Alternating table row colors with `foreach( )` and an array of color values.
- Modifying array element values inside a `foreach( )` code block.
- Visiting each element of a numeric array with `for( )`.
- Alternating table row colors with `for( )` and the modulus operator (`%`).
- Understanding the order in which `foreach( )` and `for( )` visit array elements.
- Checking for an array element with a particular key.
- Checking for an array element with a particular value.
- Interpolating array element values in strings.
- Removing an element from an array.
- Generating a string from an array with `implode( )`.
- Generating an array from a string with `explode( )`.
- Sorting an array with `sort( )`, `asort( )`, or `ksort( )`.
- Sorting an array in reverse.
- Defining a multidimensional array.
- Accessing individual elements of a multidimensional array.
- Visiting each element in a multidimensional array with `foreach( )` or `for( )`.
- Interpolating multidimensional array elements in a string.

4.7 Exercises

1. According to the U.S. Census Bureau, the 10 largest American cities (by population) in 2000 were as follows:
   - New York, NY (8,008,278 people)
   - Los Angeles, CA (3,694,820)
   - Chicago, IL (2,896,016)
   - Houston, TX (1,953,631)
   - Philadelphia, PA (1,517,550)
   - Phoenix, AZ (1,321,045)
   - San Diego, CA (1,223,400)
   - Dallas, TX (1,188,580)
   - San Antonio, TX (1,144,646)
   - Detroit, MI (951,270)

   Define an array (or arrays) that holds this information about locations and population. Print a table of locations and population information that includes the total population in all 10 cities.

2. Modify your solution to the previous exercise so that the rows in result table are ordered by population. Then modify your solution so that the rows are ordered by city name.

3. Modify your solution to the first exercise so that the table also contains rows that hold state population totals for each state represented in the list of cities.

4. For each of the following kinds of information, state how you would store it in an array and then give sample code that creates such an array with a few elements. For example, for the first item, you might say, "An associative array whose key is the student's name and whose value is an associative array of grade and ID number," as in the following:

5. ```
   $students = array('James D. McCawley' => array('grade' => 'A+','id' => 271231),
                     'Buwei Yang Chao' => array('grade' => 'A', 'id' => 818211));
   ```

   - The grades and ID numbers of students in a class.
   - How many of each item in a store inventory is in stock.
   - School lunches for a week ? the different parts of each meal (entree, side dish, drink, etc.) and the cost for each day.
   - The names of people in your family.
   - The names, ages, and relationship to you of people in your family.
   ```

Chapter 5. Functions

When you're writing computer programs, laziness is a virtue. Reusing code you've already written makes it easier to do as little work as possible. Functions are the key to code reuse. A *function* is a named set of statements that you can execute just by invoking the function name instead of retyping the statements. This saves time and prevents errors. Plus, functions make it easier to use code that other people have written (as you've discovered by using the built-in functions written by the authors of the PHP interpreter).

The basics of defining your own functions and using them are laid out in Section 5.1. When you call a function, you can hand it some values with which to operate. For example, if you write a function to check whether a user is allowed to access the current web page, you would need to provide the username and the current web page name to the function. These values are called *arguments*. Section 5.2 explains how to write functions that accept arguments and how to use the arguments from inside the function.

Some functions are one-way streets. You may pass them arguments, but you don't get anything back. A `print_header( )` function that prints the top of an HTML page may take an argument containing the page title, but it doesn't give you any information after it executes. It just displays output. Most functions move information in two directions. The access control function mentioned above is an example of this. The function gives you back a value: `true` (access granted) or `false` (access denied). This value is called the *return value*. You can use the return value of a function like any other value or variable. Return values are discussed in Section 5.3.

The statements inside a function can use variables just like statements outside a function. However, the variables inside a function and outside a function live in two separate worlds. The PHP interpreter treats a variable called `$name` inside a function and a variable called `$name` outside a function as two unrelated variables. Section 5.4 explains the rules about which variables are usable in which parts of your programs. It's important to understand these rules ? get them wrong and your code relies on uninitialized or incorrect variables. That's a bug that is hard to track down.

5.1 Declaring and Calling Functions

To create a new function, use the `function` keyword, followed by the function name and then, inside curly braces, the function body. Example 5-1 declares a new function called `page_header( )`.[1]

[1] Strictly speaking, the parentheses aren't part of the function name, but it's good practice to include them when referring to functions. Doing so helps you to distinguish functions from variables and other language constructs.

Example 5-1. Declaring a function

```
function page_header( ) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}
```

Function names follow the same rules as variable names: they must begin with a letter or an underscore, and the rest of the characters in the name can be letters, numbers, or underscores. The PHP interpreter doesn't prevent you from having a variable and a function with the same name, but you should avoid it if you can. Many things with similar names makes for programs that are hard to understand.

The `page_header( )` function defined in Example 5-1 can be called just like a built-in function. Example 5-2 uses `page_header( )` to print a complete page.

Example 5-2. Calling a function

```
page_header( );
print "Welcome, $user";
print "</body></html>";
```

Functions can be defined before or after they are called. The PHP interpreter reads the entire program file and takes care of all the function definitions before it runs any of the commands in the file. The `page_header( )` and `page_footer( )` functions in Example 5-3 both execute successfully, even though `page_header( )` is defined before it is called and `page_footer( )` is defined after it is called.

Example 5-3. Defining functions before or after calling them

```
function page_header( ) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}

page_header( );
print "Welcome, $user";
page_footer( );

function page_footer( ) {
    print '<hr>Thanks for visiting.';
    print '</body></html>';
}
```

5.2 Passing Arguments to Functions

While some functions (such as `page_header( )` in the previous section) always do the same thing, other functions operate on input that can change. The input values supplied to a function are called *arguments*. Arguments add to the power of functions because they make functions more flexible. You can modify `page_header( )` to take an argument that holds the page color. The modified function declaration is shown in Example 5-4.

Example 5-4. Declaring a function with an argument

```
function page_header2($color) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

In the function declaration, you add `$color` between the parentheses after the function name. This lets the code inside the function use a variable called `$color`, which holds the value passed to the function when it is called. For example, you can call the function like this:

```
page_header2('cc00cc');
```

This sets `$color` to `cc00cc` inside `page_header2( )`, so it prints:

```
<html><head><title>Welcome to my site</title></head><body bgcolor="#cc00cc">
```

When you define a function that takes an argument as in Example 5-4, you must pass an argument to the function when you call it. If you call the function without a value for the argument, the PHP interpreter complains with a warning. For example, if you call `page_header2( )` like this:

```
page_header2( );
```

The interpreter prints a message that looks like this:

```
PHP Warning:  Missing argument 1 for page_header2( )
```

To avoid this warning, define a function to take an optional argument by specifying a default in the function declaration. If a value is supplied when the function is called, then the function uses the supplied value. If a value is not supplied when the function is called, then the function uses the default value. To specify a default value, put it after the argument name. Example 5-5 sets the default value for `$color` to `cc3399`.

Example 5-5. Specifying a default value

```
function page_header3($color = 'cc3399') {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

Calling `page_header3('336699')` produces the same results as calling `page_header2('336699')`. When the body of each function executes, `$color` has the value `336699`, which is the color printed out for the `bgcolor` attribute of the `<body>` tag. But while `page_header2( )` without an argument produces a warning, `page_header3( )` without an argument runs just fine, with `$color` set to `cc3399`.

Default values for arguments must be literals, such as `12`, `cc3399`, or `Shredded Swiss Chard`. They can't be variables. The following is not OK:

```
$my_color = '#000000';

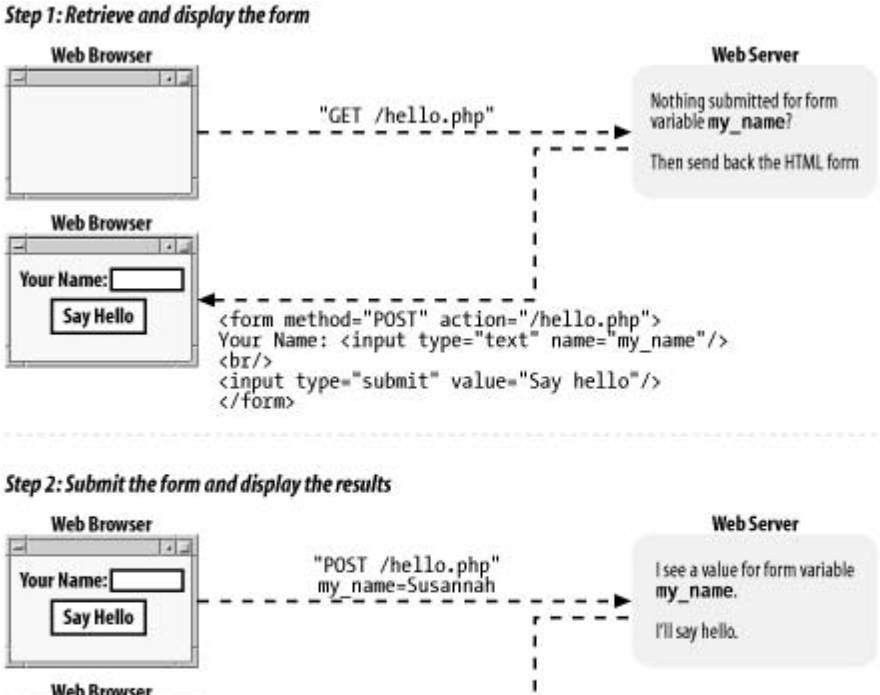
// This is incorrect: the default value can't be a variable.
function page_header_bad($color = $my_color) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
```

5.3 Returning Values from Functions

The header-printing function you've seen already in this chapter takes action by displaying some output. In addition to an action such as printing data or saving information into a database, functions can also compute a value, called the *return value*, that can be used later in a program. To capture the return value of a function, assign the function call to a variable. Example 5-10 stores the return value of the built-in function `number_format( )` in the variable `$number_to_display`.

Example 5-10. Capturing a return value

```
$number_to_display = number_format(285266237);
print "The population of the US is about: $number_to_display";
```

Just like Example 1-6, Example 5-10 prints:

```
The population of the US is about: 285,266,237
```

Assigning the return value of a function to a variable is just like assigning a string or number to a variable. The statement `$number = 57` means "store `57` in the variable `$number`." The statement `$number_to_display = number_format(285266237)` means "call the `number_format( )` function with the argument `285266237` and store the return value in `$number_to_display`." Once the return value of a function has been put into a variable, you can use that variable and the value it contains just like any other variable in your program.

To return values from functions you write, use the `return` keyword with a value to return. When a function is executing, as soon as it encounters the `return` keyword, it stops running and returns the associated value. Example 5-11 defines a function that returns the total amount of a restaurant check after adding tax and tip.

Example 5-11. Returning a value from a function

```
function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;

    return $total_amount;
}
```

The value that `restaurant_check( )` returns can be used like any other value in a program. Example 5-12 uses the return value in an `if( )` statement.

Example 5-12. Using a return value in an if( ) statement

```
// Find the total cost of a $15.22 meal with 8.25% tax and a 15% tip
$total = restaurant_check(15.22, 8.25, 15);

print 'I only have $20 in cash, so...';
if ($total > 20) {
    print "I must pay with my credit card.";
} else {
    print "I can pay with cash.";
}
```

A particular `return` statement can only return one value. You can't return multiple values with something like `return 15, 23`. If you want to return more than one value from a function, you can put the different values into one array and then return the array.

Example 5-13 shows a modified version of `restaurant_check( )` that returns a two-element array containing the total amount before the tip is added and after it is added.

Example 5-13. Returning an array from a function

```
function restaurant_check2($meal, $tax, $tip) {
```

5.4 Understanding Variable Scope

As you saw in Example 5-9, changes inside a function to variables that hold arguments don't affect those variables outside of the function. This is because activity inside a function happens in a different *scope*. Variables defined outside of a function are called *global variables*. They exist in one scope. Variables defined inside of a function are called *local variables*. Each function has its own scope.

Imagine each function is one branch office of a big company, and the code outside of any function is the company headquarters. At the Philadelphia branch office, co-workers refer to each other by their first names: "Alice did great work on this report," or "Bob never puts the right amount of sugar in my coffee." These statements talk about the folks in Philadelphia (local variables of one function), and say nothing about an Alice or a Bob who works at another branch office (local variables of another function) or at company headquarters (global variables).

Local and global variables work similarly. A variable called `$dinner` inside a function, whether or not it's an argument to that function, is completely disconnected from a variable called `$dinner` outside of the function and from a variable called `$dinner` inside another function. Example 5-20 illustrates the unconnectedness of variables in different scopes.

Example 5-20. Variable scope

```
$dinner = 'Curry Cuttlefish';

function vegetarian_dinner( ) {
    print "Dinner is $dinner, or ";
    $dinner = 'Sauteed Pea Shoots';
    print $dinner;
    print "\n";
}

function kosher_dinner( ) {
    print "Dinner is $dinner, or ";
    $dinner = 'Kung Pao Chicken';
    print $dinner;
    print "\n";
}

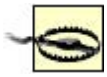
print "Vegetarian ";
vegetarian_dinner( );
print "Kosher ";
kosher_dinner( );
print "Regular dinner is $dinner";
```

Example 5-20 prints:

```
Vegetarian Dinner is , or Sauteed Pea Shoots
Kosher Dinner is , or Kung Pao Chicken
Regular dinner is Curry Cuttlefish
```

In both functions, before `$dinner` is set to a value inside the function, it has no value. The global variable `$dinner` has no effect inside the function. Once `$dinner` is set inside a function, though, it doesn't affect the global `$dinner` set outside any function or the `$dinner` variable in another function. Inside each function, `$dinner` refers to the local version of `$dinner` and is completely separate from a variable that happens to have the same name in another function.

Like all analogies, though, the analogy between variable scope and corporate organization is not perfect. In a company, you can easily refer to employees at other locations; the folks in Philadelphia can talk about "Alice at headquarters" or "Bob in Atlanta," and the overlords at headquarters can decide the futures of "Alice in Philadelphia" or "Bob in Charleston." With variables, however, you can access global variables from inside a function, but you can't access the local variables of a function from outside that function. This is equivalent to folks

5.5 Chapter Summary

Chapter 5 covers:

- Defining your own functions and calling them in your programs.
- Defining a function with mandatory arguments.
- Defining a function with optional arguments.
- Returning a value from a function.
- Understanding variable scope.
- Using global variables inside a function.

PREV < Day Day Up > NEXT

5.6 Exercises

1. Write a function to print out an HTML `<img>` tag. The function should accept a mandatory argument of the image URL and optional arguments for `alt` text, height, and width.

2. Modify the function in the previous exercise so that the filename only is passed to the function in the URL argument. Inside the function, prepend a global variable to the filename to make the full URL. For example, if you pass *photo.png* to the function, and the global variable contains `/images/`, then the `src` attribute of the printed `<img>` tag would be `/images/photo.png`. A function like this is an easy way to keep your image tags correct, even if the images move to a new path or a new server. Just change the global variable ? for example, from `/images/` to `http://images.example.com/`.

3. What does the following code print out?

```
$cash_on_hand = 31;
$meal = 25;
$tax = 10;
$tip = 10;
while(($cost = restaurant_check($meal,$tax,$tip)) < $cash_on_hand) {
    $tip++;
    print "I can afford a tip of $tip% ($cost)\n";
}
function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    return $meal + $tax_amount + $tip_amount;
}
```

16. Web colors such as `#ffffff` and `#cc3399` are made by concatenating the hexadecimal color values for red, green, and blue. Write a function that accepts decimal red, green, and blue arguments and returns a string containing the appropriate color for use in a web page. For example, if the arguments are 255, 0, and 255, then the returned string should be `#ff00ff`. You may find it helpful to use the built-in function `dechex( )`, which is documented at http://www.php.net/dechex.

Chapter 6. Making Web Forms

Form processing is an essential component of almost any web application. *Forms* are how users communicate with your server: signing up for a new account, searching a forum for all the posts about a particular subject, retrieving a lost password, finding a nearby restaurant or shoemaker, or buying a book.

Using a form in a PHP program is a two-step activity. Step one is to display the form. This involves constructing HTML that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons. If you're not familiar with the HTML required to create forms, the "Forms" chapter in *HTML & XHTML: The Definitive Guide*, by Chuck Musciano and Bill Kennedy (O'Reilly) is a good place to start.

When a user sees a page with a form in it, she inputs the information into the form and then clicks a button or hits Enter to send the form information back to your server. Processing that submitted form information is step two of the operation.

Example 6-1 is a page that says "Hello" to a user. If a name is submitted, then the page displays a greeting. If a name is not submitted, then the page displays a form with which a user can submit her name.

Example 6-1. Saying "Hello"

```
if (array_key_exists('my_name',$_POST)) {
    print "Hello, ". $_POST['my_name'];
} else {
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
 Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

Remember the client and server communication picture from Chapter 1? Figure 6-1 shows the client and server communication necessary to display and process the form in Example 6-1. The first request and response pair causes the browser to display the form. In the second request and response pair, the server processes the submitted form data and the browser displays the results.

Figure 6-1. Displaying and processing a simple form



6.1 Useful Server Variables

Aside from `PHP_SELF`, the `$_SERVER` auto-global array contains a number of useful elements that provide information on the web server and the current request. Table Table 6-1 lists some of them.

Table 6-1. Entries in $_SERVER

Element	Example	Description
`QUERY_STRING`	`category=kitchen&price=5`	The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL *http://www.example.com/catalog/store.php?category=kitchen&price=5*.
`PATH_INFO`	`/browse`	Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example `PATH_INFO` shown is for the URL *http://www.example.com/catalog/store.php/browse*.
`SERVER_NAME`	*www.example.com*	The name of the web site on which the PHP interpreter is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.
		The directory on the web server computer that holds the

6.2 Accessing Form Parameters

At the beginning of every request, the PHP interpreter sets up some auto-global arrays that contain the values of any parameters submitted in a form or passed in the URL. URL and form parameters from GET method forms are put into `$_GET`. Form parameters from POST method forms are put into `$_POST`.

The URL *http://www.example.com/catalog.php?product_id=21&category=fryingpan* puts two values into `$_GET`: `$_GET['product_id']` is set to `21` and `$_GET['category']` is set to `fryingpan`. Submitting the form in Example 6-2 causes the same values to be put into `$_POST`, assuming `21` is entered in the text box and `Frying Pan` is selected from the menu.

Example 6-2. A two-element form

```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
```

Example 6-3 incorporates the form in Example 6-2 into a complete PHP program that prints the appropriate values from `$_POST` after displaying the form. Because the `action` attribute of the `<form>` tag in Example 6-3 is `catalog.php`, you need to save the program in a file called `catalog.php` on your web server. If you save it in a file with a different name, adjust the `action` attribute accordingly.

Example 6-3. Printing submitted form parameters

```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
Here are the submitted values:

product_id: <?php print $_POST['product_id']; ?>
<br/>
category: <?php print $_POST['category']; ?>
```

A form element that can have multiple values needs to have a name that ends in `[ ]`. This tells the PHP interpreter to treat the multiple values as array elements. The `<select>` menu in Example 6-4 has its submitted values put into `$_POST['lunch']`.

Example 6-4. Multiple-valued form elements

```
<form method="POST" action="eat.php">
<select name="lunch[  ]" multiple>
<option value="pork">BBQ Pork Bun</option>
<option value="chicken">Chicken Bun</option>
<option value="lotus">Lotus Seed Bun</option>
<option value="bean">Bean Paste Bun</option>
<option value="nest">Bird-Nest Bun</option>
</select>
<input type="submit" name="submit">
</form>
```

6.3 Form Processing with Functions

The basic form in Example 6-1 can be made more flexible by putting the display code and the processing code in separate functions. Example 6-6 is a version of Example 6-1 with functions.

Example 6-6. Saying "Hello" with functions

```
// Logic to do the right thing based on
// the submitted form parameters
if (array_key_exists('my_name',$_POST) {
    process_form( );
} else {
    show_form( );
}

// Do something when the form is submitted
function process_form( ) {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form( ) {
    print<<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

To change the form or what happens when it's submitted, change the body of `process_form( )` or `show_form( )`. These functions make the code a little cleaner, but the logic at the top still depends on some form-specific information: the `my_name` parameter. We can solve that problem by using a hidden parameter in the form as the test for submission. If the hidden parameter is in `$_POST`, then we process the form. Otherwise, we display it. In Example 6-7, this strategy is shown using a hidden parameter named `_submit_check`.

Example 6-7. Using a hidden parameter to indicate form submission

```
// Logic to do the right thing based on
// the hidden _submit_check parameter
if ($_POST['_submit_check']) {
    process_form( );
} else {
    show_form( );
}

// Do something when the form is submitted
function process_form( ) {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form( ) {
    print<<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
<input type="hidden" name="_submit_check" value="1">
</form>
_HTML_;
}
```

6.4 Validating Data

Some of the validation strategies discussed in this section use *regular expressions*, which are powerful text-matching patterns, written in a language all their own. If you're not familiar with regular expressions, Appendix B provides a quick introduction.



Data validation is one of the most important parts of a web application. Weird, wrong, and damaging data shows up where you least expect it. Users are careless, users are malicious, and users are fabulously more creative (often accidentally) than you may ever imagine when you are designing your application. Without a *Clockwork Orange*-style forced viewing of a filmstrip on the dangers of unvalidated data, I can't over-emphasize how crucial it is that you stringently validate any piece of data coming into your application from an external source. Some of these external sources are obvious: most of the input to your application is probably coming from a web form. But there are lots of other ways data can flow into your programs as well: databases that you share with other people or applications, web services and remote servers, even URLs and their parameters.

As mentioned earlier, Example 6-8 doesn't indicate what's wrong with the form if the check in `validate_form( )` fails. Example 6-9 alters `validate_form( )` and `show_form( )` to manipulate and print an array of possible error messages.

Example 6-9. Displaying error messages with the form

```
// Logic to do the right thing based on
// the hidden _submit_check parameter
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        process_form( );
    }
} else {
    show_form( );
}

// Do something when the form is submitted
function process_form( ) {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form($errors = '') {
    // If some errors were passed in, print them out
    if ($errors) {
        print 'Please correct these errors: <ul><li>';
        print implode('</li><li>', $errors);
        print '</li></ul>';
    }

    print<<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
<input type="hidden" name="_submit_check" value="1">
</form>
_HTML_;
}
```

6.5 Displaying Default Values

Sometimes, you want to display a form with a value already in a text box or with selected checkboxes, radio buttons, or <select> menu items. Additionally, when you redisplay a form because of an error, it is helpful to preserve any information that a user has already entered. Example 6-23 shows the code to do this. It belongs at the beginning of show_form( ) and makes $defaults the array of values to use with the form elements.

Example 6-23. Building an array of defaults

```
if ($_POST['_submit_check']) {
    $defaults = $_POST;
} else {
    $defaults = array('delivery'  => 'yes',
                      'size'      => 'medium',
                      'main_dish' => array('taro','tripe'),
                      'sweet'     => 'cake');
}
```

If $_POST['_submit_check'] is set, that means the form has been submitted. In that case, the defaults should come from whatever the user submitted. If $_POST['_submit_check'] is not set, then you can set your own defaults. For most form parameters, the default is a string or a number. For form elements that can have more than one value, such as the multivalued <select> menu main_dish, the default value is an array.

After setting the defaults, provide the appropriate value from $defaults when printing out the HTML tag for the form element. Remember to encode the defaults with htmlentities( ) when necessary in order to prevent cross-site scripting attacks. Because of the structure of the HTML tags, you need to treat text boxes, <select> menus, text areas, and checkboxes/radio buttons differently.

For text boxes, set the value attribute of the <input> tag to the appropriate element of $defaults. Example 6-24 shows how to do this.

Example 6-24. Setting a default value in a text box

```
print '<input type="text" name="my_name" value="' .
      htmlentities($defaults['my_name']). '">';
```

For multiline text areas, put the entity-encoded value between the <textarea> and </textarea> tags, as shown in Example 6-25.

Example 6-25. Setting a default value in a multiline text area

```
print '<textarea name="comments">';
print htmlentities($defaults['comments']);
print '</textarea>';
```

For <select> menus, add a check to the loop that prints out the <option> tags that prints a selected="selected" attribute when appropriate. Example 6-26 contains the code to do this for a single-valued <select> menu.

Example 6-26. Setting a default value in a <select> menu

```
$sweets = array('puff' => 'Sesame Seed Puff',
                'square' => 'Coconut Milk Gelatin Square',
                'cake' => 'Brown Sugar Cake',
                'ricemeat' => 'Sweet Rice and Meat');

print '<select name="sweet">';
// $val is the option value, $choice is what's displayed
foreach ($sweets as $option => $label) {
    print '<option value="' .$option .'"';
    if ($option =  = $defaults['sweet']) {
```

6.6 Putting It All Together

Turning the humble web form into a feature-packed application with data validation, printing default values, and processing the submitted results might seem like an intimidating task. To ease your burden, this section contains a complete example of a program that does it all:

- Displaying a form, including default values
- Validating the submitted data
- Redisplaying the form with error messages and preserved user input if the submitted data isn't valid
- Processing the submitted data if it is valid

The do-it-all example relies on some helper functions to simplify form element display. These are listed in Example 6-29.

Example 6-29. Form element display helper functions

```
//print a text box
function input_text($element_name, $values) {
    print '<input type="text" name="' . $element_name .'" value="';
    print htmlentities($values[$element_name]) . '">';
}

//print a submit button
function input_submit($element_name, $label) {
    print '<input type="submit" name="' . $element_name .'" value="';
    print htmlentities($label) .'"/>';
}

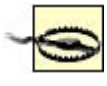
//print a textarea
function input_textarea($element_name, $values) {
    print '<textarea name="' . $element_name .'">';
    print htmlentities($values[$element_name]) . '</textarea>';
}

//print a radio button or checkbox
function input_radiocheck($type, $element_name, $values, $element_value) {
    print '<input type="' . $type . '" name="' . $element_name .'" value="' .
$element_
value . '" ';
    if ($element_value =  = $values[$element_name]) {
        print ' checked="checked"';
    }
    print '/>';
}

//print a <select> menu
function input_select($element_name, $selected, $options, $multiple = false) {
    // print out the <select> tag
    print '<select name="' . $element_name;
    // if multiple choices are permitted, add the multiple attribute
    // and add a [  ] to the end of the tag name
    if ($multiple) { print '[  ]" multiple="multiple'; }
    print '">';

    // set up the list of things to be selected
    $selected_options = array( );
    if ($multiple) {
        foreach ($selected[$element_name] as $val) {
            $selected_options[$val] = true;
        }
    } else {
        $selected_options[ $selected[$element_name] ] = true;
```

6.7 Chapter Summary

Chapter 6 covers:

- Understanding the conversation between the web browser and web server that displays a form, processes the submitted form parameters, and then displays a result.
- Making the connection between the `<form>` tag's `action` attribute and the URL to which form parameters are submitted.
- Using values from the `$_SERVER` auto-global array.
- Accessing submitted form parameters in the `$_GET` and `$_POST` auto-global arrays.
- Accessing multivalued submitted form parameters.
- Using `show_form( )`, `validate_form( )`, and `process_form( )` functions to modularize form handling.
- Using a hidden form element to check whether a form has been submitted.
- Displaying error messages with a form.
- Validating form elements: required elements, integers, floating-point numbers, strings, date ranges, email addresses, and `<select>` menus.
- Defanging or removing submitted HTML and JavaScript before displaying it.
- Displaying default values for form elements.
- Using helper functions to display form elements.

6.8 Exercises

1. What does `$_POST` look like when the following form is submitted with the third option in the `Braised Noodles` menu selected, the first and last options in the `Sweet` menu selected, and 4 entered into the text box?

```
<form method="POST" action="order.php">
Braised Noodles with: <select name="noodle">
<option>crab meat</option>
<option>mushroom</option>
<option>barbecued pork</option>
<option>shredded ginger and green onion</option>
</select>
<br/>
Sweet: <select name="sweet[  ]" multiple>
<option value="puff"> Sesame Seed Puff
<option value="square"> Coconut Milk Gelatin Square
<option value="cake"> Brown Sugar Cake
<option value="ricemeat"> Sweet Rice and Meat
</select>
<br/>
Sweet Quantity: <input type="text" name="sweet_q">
<br/>
<input type="submit" name="submit" value="Order">
</form>
```

2. Write a `process_form( )` function that prints out all submitted form parameters and their values. You can assume that form parameters have only scalar values.

3. Write a program that does basic arithmetic. Display a form with text box inputs for two operands and a `<select>` menu to choose an operation: addition, subtraction, multiplication, or division. Validate the inputs to make sure that they are numeric and appropriate for the chosen operation. The processing function should display the operands, operator, and the result. For example, if the operands are 4 and 2 and the operation is multiplication, the processing function should display something like "4 * 2 = 8".

4. Write a program that displays, validates, and processes a form for entering information about a package to be shipped. The form should contain inputs for the from and to addresses for the package, dimensions of the package, and weight of the package. The validation should check (at least) that the package weighs no more than 150 pounds and that no dimension of the package is more than 36 inches. You can assume that the addresses entered on the form are both U.S. addresses, but you should check that a valid state and a ZIP Code with valid syntax are entered. The processing function in your program should print out the information about the package in an organized, formatted report.

5. (Optional) Modify your `process_form( )` function from Exercise 6.2 so that it correctly handles submitted form parameters that have array values. Remember, those array values could themselves contain arrays.

Chapter 7. Storing Information with Databases

The HTML and CSS that give your web site its pretty face reside in individual files on your web server. So does the PHP code that processes forms and performs other dynamic wizardry. There's a third kind of information necessary to a web application, though: data. And while you can store data such as user lists and product information in individual files, most people find it easier to use databases, which are the focus of this chapter.

Lots of information falls under the broad umbrella of "data":

- Who your users are, such as their names and email addresses.
- What your users do, such as message board posts and profile information.
- The "stuff" that your site is about, such as a list of record albums, a product catalog, or what's for dinner.

There are three big reasons why this kind of data belongs in a database instead of in files: convenience, simultaneous access, and security. A database program makes it much easier to search for and manipulate individual pieces of information. With a database program, you can do things such as change the email address for user `Duck29` to `ducky@ducks.example.com` in one step. If you put usernames and email addresses in a file, changing an email address would be much more complicated: read the old file, search through each line until you find the one for `Duck29`, change the line, and write the file back out. If, at same time, one request updates `Duck29`'s email address and another updates the record for user `Piggy56`, one update could be lost, or (worse) the data file corrupted. Database software manages the intricacies of simultaneous access for you.

In addition to searchability, database programs usually provide you with a different set of access control options compared to files. It is an exacting process to set things up properly so that your PHP programs can create, edit, and delete files on your web server without opening the door to malicious attackers who could abuse that setup to alter your PHP scripts and data files. A database program makes it easier to arrange the appropriate levels of access to your information. It can be configured so that your PHP programs can read and change some information, but only read other information. However the database access control is set up, it doesn't affect how files on the web server are accessed. Just because your PHP program can change values in the database doesn't give an attacker an opportunity to change your PHP programs and HTML files themselves.

The word *database* is used in a few different ways when talking about web applications. A database can be a pile of structured information, a program (such as MySQL or Oracle) that manages that structured information, or the computer on which that program runs. In this book, I use "database" to mean the pile of structured information. The software that manages the information is a *database program*, and the computer that the database program runs on is a *database server*.

Most of this chapter uses the PEAR DB database program abstraction layer. This is an add-on to PHP that simplifies communication between your PHP program and your database program. PEAR (PHP Extension and Application Repository) is a collection of useful modules and libraries for PHP. The DB module is one of the most popular PEAR modules and is bundled with recent versions of PHP. If your PHP installation doesn't have DB installed (Section 7.2, later in this chapter, shows you how to check), see Section A.3 for instructions on how to install it.

When DB isn't available, you need to rely on other PHP functions to talk to your database program. The appropriate set of functions varies with each database program. Some of the more exotic features of your database program may only be accessible through the database-specific functions. Later in this chapter, Section 7.12 discusses shows how to work with the functions in the mysqli extension, which talks to MySQL (Versions 4.1.2 and greater).

7.1 Organizing Data in a Database

Information in your database is organized in *tables*, which have rows and columns. (Columns are also sometimes referred to as *fields*.) Each column in a table is a category of information, and each row is a set of values for each column. For example, a table holding information about dishes on a menu would have columns for each dish's ID, name, price, and spiciness. Each row in the table is the group of values for on particular dish? for example, "1," "Fried Bean Curd," "5.50," and "0" (meaning not spicy).

You can think of a table organized like a simple spreadsheet, with column names across the top, as shown in Figure 7-1.

Figure 7-1. Data organized in a grid

ID	Name	Price	Is spicy?
1	Fried Bean Curd	5.50	0
2	Braised Sea Cucumber	9.95	0
3	Walnut Bun	1.00	0
4	Eggplant with Chili Sauce	6.50	1

One important difference between a spreadsheet and a database table, however, is that the rows in a database table have no inherent order. When you want to retrieve data from a table with the rows arranged in a particular way (e.g., in alphabetic order by student name), you need to explicitly specify that order when you ask the database for the data. The *SQL Lesson:* ORDER BY and LIMIT sidebar in this chapter describes how to do this.

SQL (Structured Query Language) is a language to ask questions of and give instructions to the database program. Your PHP program sends SQL queries to a database program. If the query retrieves data in the database (for example, "Find me all spicy dishes"), then the database program responds with the set of rows that match the query. If the query changes data in the database (for example, "Add this new dish" or "Double the prices of all nonspicy dishes"), then the database program replies with whether or not the operation succeeded.

SQL is a mixed bag when it comes to case-sensitivity. SQL keywords are not case-sensitive, but in this book they are always written as uppercase to distinguish them from the other parts of the queries. Names of tables and columns in your queries generally are case-sensitive. All of the SQL examples in this book use lowercase column and table names to help you distinguish them from the SQL keywords. Any literal values that you put in queries are case-sensitive. Telling the database program that the name of a new dish is `fried bean curd` is different than telling it that the new dish is called `FRIED Bean Curd`.

Almost all of the SQL queries that you write to use in your PHP programs rely on one of four SQL commands: `INSERT`, `UPDATE`, `DELETE`, or `SELECT`. Each of these commands is described in this chapter. Section 7.3 describes the `CREATE TABLE` command, which you use to make new tables in your database.

To learn more about SQL, read *SQL in a Nutshell*, by Kevin E. Kline (O'Reilly). It provides an overview of standard SQL as well as the SQL extensions in MySQL, Oracle, PostgreSQL, and Microsoft SQL Server. For more in-depth information about working with PHP and MySQL, read *Web Database Applications with PHP & MySQL*, by Hugh E. Williams and David Lane (O'Reilly). *MySQL Cookbook*, by Paul DuBois (O'Reilly) is also an excellent source for answers to lots of SQL and MySQL questions.

7.2 Connecting to a Database Program

To use PEAR DB in a PHP program, first you have to load the DB module. Use the `require` construct, as shown in Example 7-1.

Example 7-1. Loading an external file with require

```
require 'DB.php';
```

Example 7-1 tells the PHP interpreter to execute all of the code in the file *DB.php*. *DB.php* is the main file of the PEAR DB package. It defines the functions that you use to talk to your database.

Similar to `require` is `include`. These constructs differ in how they handle errors. If you try to include or require a file that doesn't exist, `require` considers that a fatal error and your PHP program ends. The `include` construct is more forgiving and just reports a warning, allowing your program to continue running.

After the DB module is loaded, you need to establish a connection to the database with the `DB::connect( )` function. You pass `DB::connect( )` a string that describes the database you are connecting to, and it returns an *object* that you use in the rest of your program to exchange information with the database program.

An object is a new data type. It's a bundle of some data and functions that operate on that data. PEAR DB uses objects to provide you with a connection to the database. The double colons in the `DB::connect( )` function call are a way of telling the PHP interpreter that you're calling a special function based on an object.

Example 7-2 shows a call to `DB::connect( )` that connects to MySQL.

Example 7-2. Connecting with DB::connect( )

```
require 'DB.php';
$db = DB::connect('mysql://penguin:top^hat@db.example.com/restaurant');
```

The string passed to `DB::connect( )` is called a Data Source Name (DSN). Its general form is:

```
db_program://user:password@hostname/database
```

In Example 7-2, the DSN tells PEAR DB to connect to MySQL running on the database server `db.example.com` as user `penguin` with the password `top^hat`, and to access the `restaurant` database on that server.

PEAR DB supports 13 options for the `db_program` part of the DSN. These are listed in Table 7-1.

Table 7-1. PEAR DB db_program options

db_program	Database program
`dbase`	dBase
`fbsql`	FrontBase
`ibase`	InterBase
`ifx`	Informix
`msql`	Mini SQL

7.3 Creating a Table

Before you can put any data into or retrieve any data from a database table, you must create the table. This is usually a one-time operation. You tell the database program to create a new table once. Your PHP program that uses the table may read from or write to that table every time it runs. But it doesn't have to re-create the table each time. If a database table is like a spreadsheet, then creating a table is like making a new spreadsheet file. After you create the file, you open it many times to read or change it.

The SQL command to create a table is `CREATE TABLE`. You provide the name of the table and the names and types of all the columns in the table. Example 7-5 shows the SQL command to create the `dishes` table pictured in Figure 7-1.

Example 7-5. Creating the dishes table

```
CREATE TABLE dishes (
    dish_id INT,
    dish_name VARCHAR(255),
    price DECIMAL(4,2),
    is_spicy INT
)
```

Example 7-5 creates a table called `dishes` with four columns. The `dishes` table looks like the one pictured in Figure 7-1. The columns in the table are `dish_id`, `dish_name`, `price`, and `is_spicy`. The `dish_id` and `is_spicy` columns are integers. The `price` column is a decimal number. The `dish_name` column is a string.

After the literal `CREATE TABLE` comes the name of the table. Then, between the parentheses, is a comma-separated list of the columns in the table. The phrase that defines each column has two parts: the column name and the column type. In Example 7-5, the column names are `dish_id`, `dish_name`, `price`, and `is_spicy`. The column types are `INT`, `VARCHAR(255)`, `DECIMAL(4,2)`, and `INT`.

Some column types include length or formatting information in the parentheses. For example, `VARCHAR(255)` means "a variable length character column that is at most 255 characters long." The type `DECIMAL(4,2)` means "a decimal number with two digits after the decimal place and four digits total." Table 7-2 lists some common types for database table columns.

Table 7-2. Common database table column types

Column type	Description
`VARCHAR(`*`length`*`)`	A variable length string up to *`length`* characters long.
`INT`	An integer.
`BLOB`[1]	Up to 64k of string or binary data.
`DECIMAL(`*`total_digits`*`,`*`decimal_places`*`)`	A decimal number with a total of *`total_digits`* digits and *`decimal_places`* digits after the decimal point.
`DATETIME`[2]	A date and time, such as `1975-03-10 19:45:03` or `2038-01-18 22:14:07`.

[1] PostgreSQL calls this `BYTEA` instead of `BLOB`.

[2] Oracle calls this `DATE` instead of `DATETIME`.

7.4 Putting Data into the Database

Assuming the connection to the database succeeds, the object returned by `DB::connect( )` provides access to the data in your database. Calling that object's functions lets you send queries to the database program and access the results. To put some data into the database, pass an `INSERT` statement to the object's `query( )` function, as shown in Example 7-8.

Example 7-8. Inserting data with query( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die("connection error: " . $db->getMessage( )); }
$q = $db->query("INSERT INTO dishes (dish_name, price, is_spicy)
    VALUES ('Sesame Seed Puff', 2.50, 0)");
```

Just like with the `$db` object that `DB::connect( )` returns, the `$q` object that `query( )` returns can be tested with `DB::isError( )` to check whether the query was successful. Example 7-9 attempts an `INSERT` statement that has a bad column name in it. The `dishes` table doesn't contain a column called `dish_size`.

Example 7-9. Checking for errors from query( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die("connection error: " . $db->getMessage( )); }
$q = $db->query("INSERT INTO dishes (dish_size, dish_name, price, is_spicy)
    VALUES ('large', 'Sesame Seed Puff', 2.50, 0)");
if (DB::isError($q)) { die("query error: " . $q->getMessage( )); }
```

Example 7-9 prints:

```
query error: DB Error: syntax error
```

Instead of calling `DB::isError( )` after every query to see if it succeeded or failed, it's more convenient to use the `setErrorHandling( )` function to establish a default error-handling behavior. Pass the constant `PEAR_ERROR_DIE` to `setErrorHandling( )` to have your program automatically print an error message and exit if a query fails. Example 7-10 uses `setErrorHandling( )` and has the same incorrect query as Example 7-9.

Example 7-10. Automatic error handling with setErrorHandling( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die("Can't connect: " . $db->getMessage( )); }

// print a message and quit on future database errors
$db->setErrorHandling(PEAR_ERROR_DIE);

$q = $db->query("INSERT INTO dishes (dish_size, dish_name, price, is_spicy)
    VALUES ('large', 'Sesame Seed Puff', 2.50, 0)");
print "Query Succeeded!";
```

SQL Lesson: INSERT

The `INSERT` command adds a row to a database table. Example 7-11 shows the syntax of `INSERT`.

Example 7-11. Inserting data

```
INSERT INTO table (column1[, column2, column3, ...])
    VALUES (value1[, value2, value3, ...])
```

7.5 Inserting Form Data Safely

As Section 6.4.6 explained, printing unsanitized form data can leave you and your users vulnerable to a cross-site scripting attack. Using unsanitized form data in SQL queries can cause a similar problem, called an "SQL injection attack." Consider a form that lets a user suggest a new dish. The form contains a text element called `new_dish_name` into which the user can type the name of their new dish. The call to `query( )` in Example 7-26 inserts the new dish into the `dishes` table but is vulnerable to an SQL injection attack.

Example 7-26. Unsafe insertion of form data

```
$db->query("INSERT INTO dishes (dish_name)
            VALUES ('$_POST[new_dish_name]')");
```

If the submitted value for `new_dish_name` is reasonable, such as `Fried Bean Curd`, then the query succeeds. PHP's regular double-quoted string interpolation rules make the query `INSERT INTO dishes (dish_name) VALUES ('Fried Bean Curd')`, which is valid and respectable. A query with an apostrophe in it causes a problem, though. If the submitted value for `new_dish_name` is `General Tso's Chicken`, then the query becomes `INSERT INTO dishes (dish_name) VALUES ('General Tso's Chicken')`. This makes the database program confused. It thinks that the apostrophe between `Tso` and `s` ends the string, so the `s Chicken ' ` after the second single quote is an unwanted syntax error.

What's worse, a user that really wants to cause problems can type in specially constructed input to wreak havoc. Consider this unappetizing input:

```
x'); DELETE FROM dishes; INSERT INTO dishes (dish_name) VALUES ('y.
```

When that gets interpolated, the query becomes:

```
INSERT INTO DISHES (dish_name) VALUES ('x'); DELETE FROM dishes; INSERT INTO
dishes
(dish_name) VALUES ('y')
```

Some databases let you pass multiple queries separated by semicolons in one call of `query( )`. On those databases, the `dishes` table is demolished: a dish named `x` is inserted, all dishes are deleted, and a dish named `y` is inserted.

By submitting a carefully built form input value, a malicious user is able to inject arbitrary SQL statements into your database program. To prevent this, you need to escape special characters (most importantly, the apostrophe) in SQL queries. PEAR DB provides a helpful feature called *placeholders* that makes this a snap.



PHP has an unfortunate feature called "Magic Quotes." If this is turned on, submitted form data has quotes and backslashes escaped before it is put into `$_GET` or `$_POST`. If someone submits a form with `Sauteed Pig's Stomach` typed into the a text field named `entree`, then `$_POST['entree']` is not `Sauteed Pig's Stomach`, but `Sauteed Pig\'s Stomach` instead. This is conceivably handy if all you're going to do with `$_POST['entree']` is use it in a database query, but it is very inconvenient if you want to use `$_POST['entree']` in other contexts (such as simply printing it) where the extra backslash is not welcome.

The "Magic Quotes" feature is enabled when the PHP configuration directive `magic_quotes_gpc` is turned on. For increased efficiency and more straightforward handling of submitted form parameters, turn `magic_quotes_gpc` off and use placeholders or a quoting function when you need to prepare external input for use in a database query.

To use a placeholder in a query, put a `?` in the query in each place where you want a value to go. Then, pass `query( )` a second argument? an array of values to be substituted for the placeholders. The values are appropriately quoted before they are put into the query.

7.6 Generating Unique IDs

As mentioned in Section 7.1, rows in a database table don't have any inherent order. In a spreadsheet, you can refer particular records such as "the first row" or "the last row" or "rows 15 to 22." A database table is different. If you want to be able to specifically identify individual records, you need to give them each a unique identifier.

To uniquely identify individual rows in a table, make a column in the table that holds an integer ID and store a different number in that column for each row. That way, even if two rows have identical values in all the other columns, you can tell them apart by using the ID column. With a `dish_id` column in the `dishes` table, you can tell apart two dishes each called "Fried Bean Curd" because the rows have different `dish_id` values.

PEAR DB helps you generate unique integer IDs with its support for *sequences*. When you ask for the next ID in a particular sequence, you get a number that you know isn't duplicated in that sequence. Even if two simultaneously executing PHP scripts ask for the next ID in a sequence at the exact same time, they each get a different ID to use.

You can have as many independent sequences as you want. To get the next value from a sequence, call the `nextID( )` function. Example 7-29 gets an ID from the `dishes` sequence and then uses it to `INSERT` a row into the `dishes` table.

Example 7-29. Getting an ID from a sequence

```
$dish_id = $db->nextID('dishes');
$db->query("INSERT INTO orders (dish_id, dish_name, price, is_spicy)
    VALUES ($dish_id, 'Fried Bean Curd', 1.50, 0)");
```

7.7 A Complete Data Insertion Form

Example 7-30 combines the database topics covered so far in this chapter with the form-handling code from Chapter 6 to build a complete program that displays a form, validates the submitted data, and then saves the data into a database table. The form displays input elements for the name of a dish, the price of a dish, and whether the dish is spicy. The information is inserted into the `dishes` table.

The code in Example 7-30 relies on the form helper functions defined in Example 6-29. Instead of repeating them in this example, the code assumes they have been saved into a file called *formhelpers.php* and then loads them with the `require 'formhelpers.php'` line at the top of the program.

Example 7-30. Form for inserting records into dishes

```
<?php
// Load PEAR DB
require 'DB.php';
// Load the form helper functions
require 'formhelpers.php';

// Connect to the database
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die ("Can't connect: " . $db->getMessage( )); }
// Set up automatic error handling
$db->setErrorHandling(PEAR_ERROR_DIE);

// The main page logic:
// - If the form is submitted, validate and then process or redisplay
// - If it's not submitted, display
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}

function show_form($errors = '') {
    // If the form is submitted, get defaults from submitted parameters
    if ($_POST['_submit_check']) {
        $defaults = $_POST;
    } else {
        // Otherwise, set our own defaults: price is $5
        $defaults = array('price' => '5.00');
    }

    // If errors were passed in, put them in $error_text (with HTML markup)
    if ($errors) {
        $error_text = '<tr><td>You need to correct the following errors:';
        $error_text .= '</td><td><ul><li>';
        $error_text .= implode('</li><li>',$errors);
        $error_text .= '</li></ul></td></tr>';
    } else {
        // No errors? Then $error_text is blank
        $error_text = '';
    }

    // Jump out of PHP mode to make displaying all the HTML tags easier
?>
```

7.8 Retrieving Data from the Database

The `query( )` function can also be used to retrieve information from the database. The syntax of `query( )` is the same, but what you do with the object that `query( )` returns is new. When it successfully completes a `SELECT` statement, `query( )` returns an object that provides access to the retrieved rows. Each time you call the `fetchRow( )` function of this object, you get the next row returned from the query. When there are no more rows left, `fetchRow( )` returns a false value, making it perfect to use in a `while( )` loop. This is shown in Example 7-31.

Example 7-31. Retrieving rows with query( ) and fetchRow( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetchRow( )) {
    print "$row[0], $row[1] \n";
}
```

Example 7-31 prints:

```
Walnut Bun, 1.00
Cashew Nuts and White Mushrooms, 4.95
Dried Mulberries, 3.00
Eggplant with Chili Sauce, 6.50
```

The first time through the `while( )` loop, `fetchRow( )` returns an array containing `Walnut Bun` and `1.00`. This array is assigned to `$row`. Since an array with elements in it evaluates to `true`, the code inside the `while( )` loop executes, printing the data from the first row returned by the `SELECT` query. This happens three more times. On each trip through the `while( )` loop, `fetchRow( )` returns the next row in the set of rows returned by the `SELECT` query. When it has no more rows to return, `fetchRow( )` returns a value that evaluates to `false`, and the `while( )` loop is done.

To find out the number of rows returned by a `SELECT` query (without iterating through them all), use the `numrows( )` function of the object returned by `query( )`. Example 7-32 reports how many rows are in the `dishes` table.

Example 7-32. Counting rows with numrows( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
$q = $db->query('SELECT dish_name, price FROM dishes');
print 'There are ' . $q->numrows( ) . ' rows in the dishes table.';
```

With four rows in the table, Example 7-32 prints:

```
There are 5 rows in the dishes table.
```

Because sending a `SELECT` query to the database program and retrieving the results is such a common task, DB provides ways that collapse the call to `query( )` and multiple calls to `fetchRow( )` into one step. The `getAll( )` function executes a `SELECT` query and returns an array containing all the retrieved rows. Example 7-33 uses `getAll( )` to do the same thing as Example 7-31.

Example 7-33. Retrieving rows with getAll( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
$rows = $db->getAll('SELECT dish_name, price FROM dishes');
foreach ($rows as $row) {
    print "$row[0], $row[1] \n";
}
```

7.9 Changing the Format of Retrieved Rows

So far, `fetchRow( )`, `getAll( )`, and `getOne( )` have been returning rows from the database as numerically indexed arrays. This makes for concise and easy interpolation of values in double-quoted strings? but trying to remember, for example, which column from the `SELECT` query corresponds to element 6 in the result array can be difficult and error-prone. PEAR DB lets you specify that you'd prefer to have each result row delivered as either an array with string keys or as an object.

The *fetch mode* controls how result rows are formatted. The `setFetchMode( )` function changes the fetch mode. Any queries in a page after you call `setFetchMode( )` have their result rows formatted as specified by the argument to `setFetchMode( )`.

To get result rows as arrays with string keys, pass `DB_FETCHMODE_ASSOC` to `setFetchMode( )`. Note that `DB_FETCHMODE_ASSOC` is a special constant defined by PEAR DB, not a string, so you shouldn't put quotes around it. The array keys in the result row arrays correspond to column names. Example 7-46 shows how to use `fetchRow( )`, `getAll( )`, and `getRow( )` with string-keyed result rows.

Example 7-46. Retrieving rows as string-keyed arrays

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');

// Change the fetch mode to string-keyed arrays
$db->setFetchMode(DB_FETCHMODE_ASSOC);

print "With query( ) and fetchRow( ): \n";
// get each row with query( ) and fetchRow( );
$q = $db->query("SELECT dish_name, price FROM dishes");
while($row = $q->fetchRow( )) {
    print "The price of $row[dish_name] is $row[price] \n";
}

print "With getAll( ): \n";
// get all the rows with getAll( );
$dishes = $db->getAll('SELECT dish_name, price FROM dishes');
foreach ($dishes as $dish) {
    print "The price of $dish[dish_name] is $dish[price] \n";
}

print "With getRow( ): \n";
$cheap = $db->getRow('SELECT dish_name, price FROM dishes
    ORDER BY price LIMIT 1');
print "The cheapest dish is $cheap[dish_name] with price $cheap[price]";
```

Example 7-46 prints:

```
With query( ) and fetchRow( ):
The price of Walnut Bun is 1.00
The price of Cashew Nuts and White Mushrooms is 4.95
The price of Dried Mulberries is 3.00
The price of Eggplant with Chili Sauce is 6.50
With getAll( ):
The price of Walnut Bun is 1.00
The price of Cashew Nuts and White Mushrooms is 4.95
The price of Dried Mulberries is 3.00
The price of Eggplant with Chili Sauce is 6.50
With getRow( ):
The cheapest dish is Walnut Bun with price 1.00
```

In Example 7-46, `fetchRow( )`, `getAll( )`, and `getRow( )` operate almost identically as they have before: you give them an SQL query, and you get back some results. The difference is in those results. The rows that come back from these functions have string keys

7.10 Retrieving Form Data Safely

It's possible to use placeholders with SELECT statements just as you do with INSERT, UPDATE, or DELETE statements. The getAll( ), getRow( ), and getOne( ) functions each accept a second argument of an array of values that are substituted for placeholders in a query.

However, when you use submitted form data or other external input in the WHERE clause of a SELECT, UPDATE, or DELETE statement, you must take extra care to ensure that any SQL wildcards are appropriately escaped. Consider a search form with a text element called dish_search into which the user can type a name of a dish he's looking for. The call to getAll( ) in Example 7-48 uses placeholders guard against confounding single-quotes in the submitted value.

Example 7-48. Using a placeholder in a SELECT statement

```
$matches = $db->getAll('SELECT dish_name, price FROM dishes
                        WHERE dish_name LIKE ?',
                       array($_POST['dish_search']));
```

Whether dish_search is Fried Bean Curd or General Tso's Chicken, the placeholder interpolates the value into the query appropriately. However, what if dish_search is %chicken%? Then, the query becomes SELECT dish_name, price FROM dishes WHERE dish_name LIKE '%chicken%'. This matches all rows that contain the string chicken, not just rows in which dish_name is exactly %chicken%.

To prevent SQL wildcards in form data from taking effect in queries, you must forgo the comfort and ease of the placeholder and rely on two other functions:

SQL Lesson: Wildcards

Wildcards are useful for matching text inexactly, such as finding strings that end with .edu or that contain @. SQL has two wildcards. The underscore (_) matches one character and the percent sign (%) matches any number of characters (including zero characters). The wildcards are active inside strings used with the LIKE operator in a WHERE clause.

Example 7-49 shows two SELECT queries that use LIKE and wildcards.

Example 7-49. Using wildcards with SELECT

```
; Retrieve all rows in which dish name begins with D
SELECT * FROM dishes WHERE dish_name LIKE 'D%'

; Retrieve rows in which dish name is Fried Cod, Fried Bod,
; Fried Nod, and so on.
SELECT * FROM dishes WHERE dish_name LIKE 'Fried _od'
```

Wildcards are active in the WHERE clauses of UPDATE and DELETE statements, too. The query in Example 7-50 doubles the price of all dishes that have chili in their names.

Example 7-50. Using wildcards with UPDATE

```
UPDATE dishes SET price = price * 2 WHERE dish_name LIKE '%chili%'
```

The query in Example 7-51 deletes all rows whose dish_name ends with Shrimp.

Example 7-51. Using wildcards with DELETE

```
DELETE FROM dishes WHERE dish_name LIKE '%Shrimp'
```

To match against a literal % or _ when using the LIKE operator, put a backslash before the % or _. The query in Example 7-52 finds all rows whose dish_name

7.11 A Complete Data Retrieval Form

Example 7-56 is another complete database and form program. It presents a search form and then prints an HTML table of all rows in the `dishes` table that match the search criteria. Like Example 7-30, it relies on the form helper functions being defined in a separate *formhelpers.php* file.

Example 7-56. Form for searching the dishes table

```
<?php

// Load PEAR DB
require 'DB.php';
// Load the form helper functions.
require 'formhelpers.php';

// Connect to the database
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die ("Can't connect: " . $db->getMessage( )); }

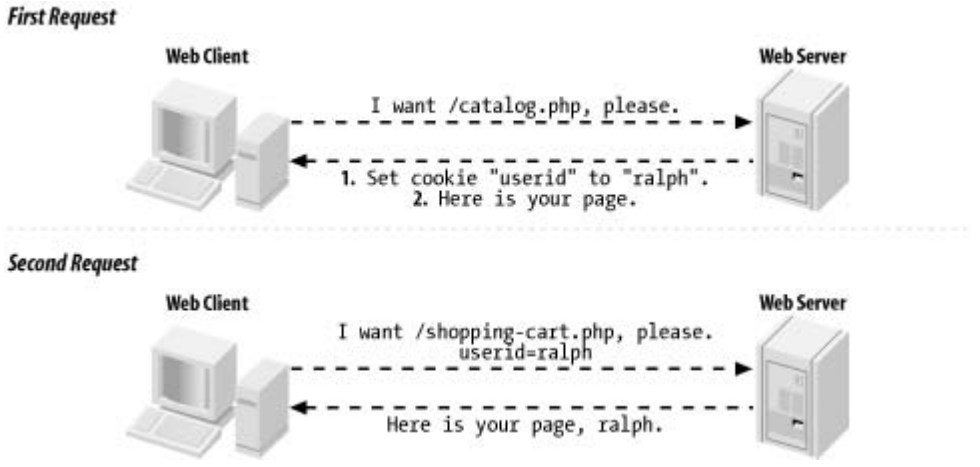
// Set up automatic error handling
$db->setErrorHandling(PEAR_ERROR_DIE);

// Set up fetch mode: rows as objects
$db->setFetchMode(DB_FETCHMODE_OBJECT);

// Choices for the "spicy" menu in the form
$spicy_choices = array('no','yes','either');

// The main page logic:
// - If the form is submitted, validate and then process or redisplay
// - If it's not submitted, display
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}

function show_form($errors = '') {
    // If the form is submitted, get defaults from submitted parameters
    if ($_POST['_submit_check']) {
        $defaults = $_POST;
    } else {
        // Otherwise, set our own defaults
        $defaults = array('min_price' => '5.00',
                          'max_price' => '25.00');
    }

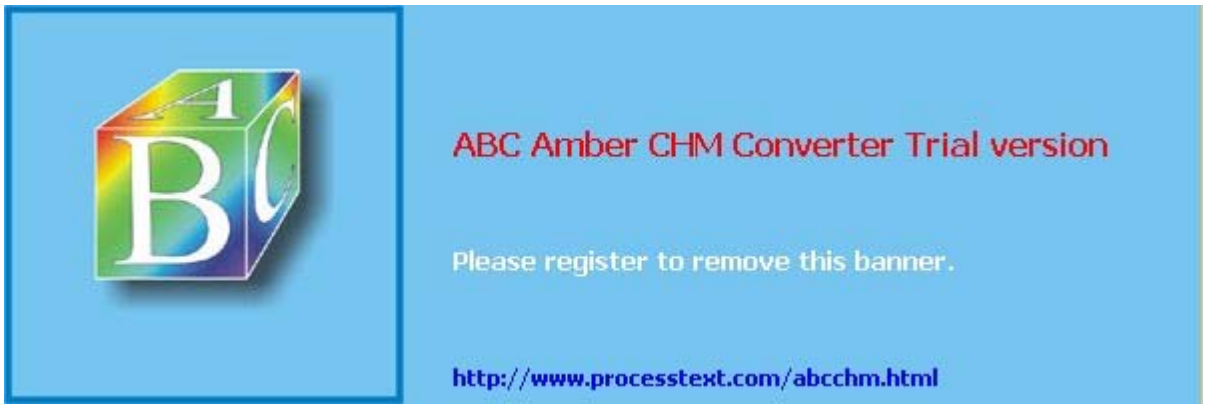
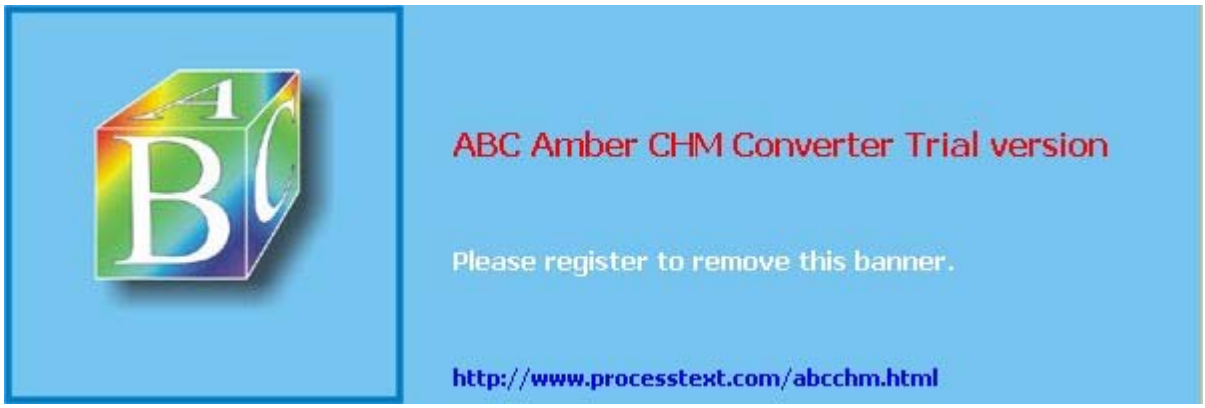
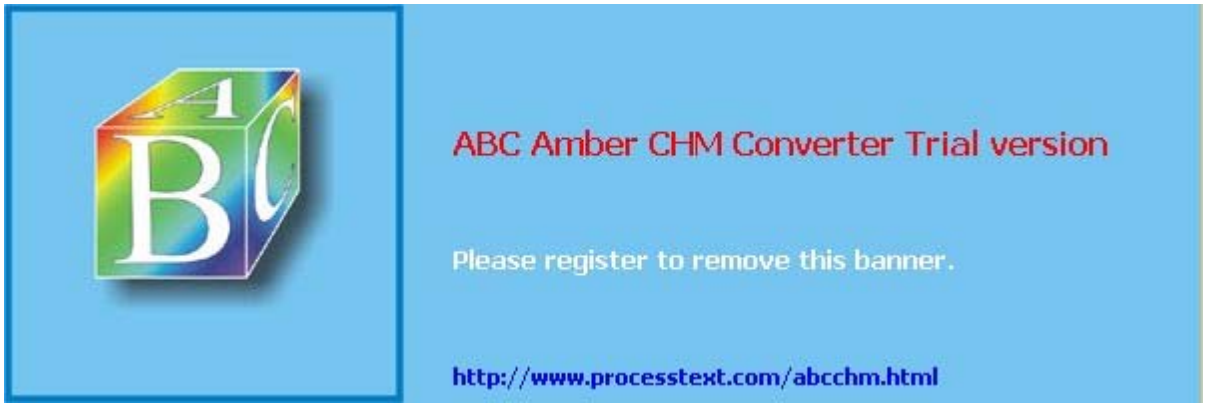
    // If errors were passed in, put them in $error_text (with HTML markup)
    if ($errors) {
        $error_text = '<tr><td>You need to correct the following errors:';
        $error_text .= '</td><td><ul><li>';
        $error_text .= implode('</li><li>',$errors);
        $error_text .= '</li></ul></td></tr>';
    } else {
        // No errors? Then $error_text is blank
        $error_text = '';
    }
```

7.12 MySQL Without PEAR DB

PEAR DB smooths over a lot of the rough edges of database access in a PHP program, but there are two reasons why it's not always the right choice: PEAR DB might not be available on some systems, and a program that uses the built-in PHP functions tailored to a particular database is faster than one that uses PEAR DB. Programmers who don't anticipate switching or using more than one database program often pick those built-in functions.

The basic model of database access with the built-in functions is the same as with PEAR DB. You call a function that connects to the database. It returns a variable that represents the connection. You use that connection variable with other functions to send queries to the database program and retrieve the results.

The differences are in the details. The applicable functions and how they work differ from database to database. In general, you have to retrieve results one row at a time instead of the convenience that `getAll( )` offers, and there is no unified error handling.

As an example for database access without PEAR DB, this section discusses the mysqli extension, which works with MySQL 4.1.2 or greater and with PHP 5. There are similar PHP extensions for other database programs. Table 7-4 lists the database programs that PHP supports and where in the PHP Manual you can read about the functions in the extension for each database. All of the extensions listed in Table 7-4 are not usually installed by default with the PHP interpreter, but the PHP Manual gives instructions on how to install them.

Table 7-4. Database extensions

Database program	PHP Manual URL
Adabas D	http://www.php.net/uodbc
DB2	http://www.php.net/uodbc
DB++	http://www.php.net/dbplus
Empress	http://www.php.net/uodbc
FrontBase	http://www.php.net/fbsql
Informix	http://www.php.net/ifx
InterBase	http://www.php.net/ibase
Ingres II	http://www.php.net/ingres
Microsoft SQL Server	http://www.php.net/mssql
mSQL	http://www.php.net/msql
MySQL (Version 4.1.1 and earlier)	http://www.php.net/mysql
MySQL (Version 4.1.2 and later)	http://www.php.net/mysqli
ODBC	http://www.php.net/uodbc
Oracle	http://www.php.net/oci8
Oracle 8	http://www.php.net/oci8

7.13 Chapter Summary

Chapter 7 covers:

- Figuring out what kinds of information belong in a database.
- Understanding how data is organized in a database.
- Loading an external file with `require`.
- Establishing a database connection.
- Creating a table in the database.
- Removing a table from the database.
- Using the SQL `INSERT` command.
- Inserting data into the database with `query( )`.
- Checking for database errors with `DB::isError( )`.
- Setting up automatic error handling with `setErrorHandling( )`.
- Using the SQL `UPDATE` and `DELETE` commands.
- Changing or deleting data with `query( )`.
- Counting the number of rows affected by a query.
- Using placeholders to insert data safely.
- Generating unique ID values with sequences.
- Using the SQL `SELECT` command.
- Retrieving data from the database with `query( )` and `fetchRow( )`.
- Counting the number of rows retrieved by `query( )`.
- Retrieving data with `getAll( )`, `getRow( )`, and `getOne( )`.
- Using the SQL `ORDER BY` and `LIMIT` keywords with `SELECT`.
- Retrieving rows as string-keyed arrays or objects.
- Using the SQL wildcards with `LIKE`: `%` and `_`.
- Escaping SQL wildcards in `SELECT` statements.
- Saving submitted form parameters in the database.
- Using data from the database in form elements.
- Using the mysqli functions instead of PEAR DB.

7.14 Exercises

The following exercises use a database table called `dishes` with the following structure:

```
CREATE TABLE dishes (
   dish_id     INT,
   dish_name   VARCHAR(255),
   price       DECIMAL(4,2),
   is_spicy    INT
)
```

Here is some sample data to put into the `dishes` table:

```
INSERT INTO dishes VALUES (1,'Walnut Bun',1.00,0)
INSERT INTO dishes VALUES (2,'Cashew Nuts and White Mushrooms',4.95,0)
INSERT INTO dishes VALUES (3,'Dried Mulberries',3.00,0)
INSERT INTO dishes VALUES (4,'Eggplant with Chili Sauce',6.50,1)
INSERT INTO dishes VALUES (5,'Red Bean Bun',1.00,0)
INSERT INTO dishes VALUES (6,'General Tso\'s Chicken',5.50,1)
```

1. Write a program that lists all of the dishes in the table, sorted by price.
2. Write a program that displays a form asking for a price. When the form is submitted, the program should print out the names and prices of the dishes whose price is at least the submitted price. Don't retrieve from the database any rows or columns that aren't printed in the table.
3. Write a program that displays a form with a `<select>` menu of dish names. Create the dish names to display by retrieving them from the database. When the form is submitted, the program should print out all of the information in the table (ID, name, price, and spiciness) for the selected dish.
4. Create a new table that holds information about restaurant customers. The table should store the following information about each customer: customer ID, name, phone number, and the ID of the customer's favorite dish. Write a program that displays a form for putting a new customer into the table. The part of the form for entering the customer's favorite dish should be a `<select>` menu of dish names. The customer's ID should be generated by your program, not entered in the form.

Chapter 8. Remembering Users with Cookies and Sessions

A web server is a lot like a clerk at a busy deli full of pushy customers. The customers at the deli shout requests: "I want a half pound of corned beef!" and "Give me a pound of pastrami, sliced thin!" The clerk scurries around slicing and wrapping to satisfy the requests. Web clients electronically shout requests ("Give me */catalog/yak.php*!" or "Here's a form submission for you!"), and the server, with the PHP interpreter's help, electronically scurries around constructing responses to satisfy the requests.

The clerk has an advantage that the web server doesn't, though: a memory. She naturally ties together all the requests that come from a particular customer. The PHP interpreter and the web server can't do that without some extra steps. That's where *cookies* come in.

A cookie identifies a particular web client to the web server and to the PHP interpreter. Each time a web client makes a request, it sends the cookie along with the request. The interpreter reads the cookie and figures out that a particular request is coming from the same web client that made previous requests, which were accompanied by the same cookie.

If deli customers were faced with a memory-deprived clerk, they'd have to adopt the same strategy. Their requests for service would look like this:

```
"I'm customer 56 and I want a half-pound of corned beef."
"I'm customer 29 and I want three knishes."
"I'm customer 56 and I want two pounds of pastrami."
"I'm customer 77 and I'm returning this rye bread -- it's stale."
"I'm customer 29 and I want a salami."
```

The "I'm customer so-and-so" part of the requests is the cookie. It gives the clerk what she needs to be able to link a particular customer's requests together.

A cookie has a name (such as "customer") and a value (such as "77" or "ronald"). Section 8.1, next, shows you how to work with individual cookies in your programs: setting them, reading them, and deleting them.

One cookie is best at keeping track of one piece of information. Often, you need to keep track of more about a user (such as the contents of their shopping cart). Using multiple cookies for this is cumbersome. PHP's *session* capabilities solve this problem.

A session uses a cookie to distinguish users from each other and makes it easy to keep a temporary pile of data for each user on the server. This data persists across requests. On one request, you can add a variable to a user's session (such as putting something into the shopping cart). On a subsequent request, you can retrieve what's in the session (such as on the order checkout page when you need to list everything in the cart). Later in this chapter, Section 8.2 describes how to get started with sessions, and Section 8.3 provides the details on working with sessions.

8.1 Working with Cookies

To set a cookie, use the `setcookie( )` function. This tells a web client to remember a cookie name and value and send them back to the server on subsequent requests. Example 8-1 sets a cookie named `userid` to value `ralph`.

Example 8-1. Setting a cookie

```
setcookie('userid','ralph');
```

To read a previously set cookie from your PHP program, use the `$_COOKIE` auto-global array. Example 8-2 prints the value of the `userid` cookie.

Example 8-2. Printing a cookie value

```
print 'Hello, ' . $_COOKIE['userid'];
```

The value for a cookie that you provide to `setcookie( )` can be a string or a number. It can't be an array or more complicated data structure.

When you call `setcookie( )`, the response that the PHP interpreter generates to send back to the web client includes a special header that tells the web client about the new cookie. On subsequent requests, the web client sends that cookie name and value back to the server. This two-step conversation is illustrated in Figure 8-1.

Figure 8-1. Client and server communication when setting a cookie



Usually, you must call `setcookie( )` before the page generates any output. This means that `setcookie( )` must come before any `print` statements. It also means that there can't be any text before the PHP `<?php` start tag in the page that comes before the `setcookie( )` function. Later in this chapter, Section 8.6 explains why this requirement exists, and how, in some cases, you can get around it.

Example 8-3 shows the correct way to put a `setcookie( )` call at the top of your page.

Example 8-3. Starting a page with setcookie( )

```
<?php
setcookie('userid','ralph');
?>
<html><head><title>Page with cookies</title><head>
<body>
This page sets a cookie properly, because the PHP block
with setcookie( ) in it comes before all of the HTML.
</body></html>
```

Cookies show up in `$_COOKIE` only when the web client sends them along with the request. This means that a name and value do not appear in `$_COOKIE` immediately after you call

8.2 Activating Sessions

Sessions use a cookie called `PHPSESSID`. When you start a session on a page, the PHP interpreter checks for the presence of this cookie and sets it if it doesn't exist. The value of the `PHPSESSID` cookie is a random alphanumeric string. Each web client gets a different session ID. The session ID in the `PHPSESSID` cookie identifies that web client uniquely to the server. That lets the interpreter maintain separate piles of data for each web client.

The conversation between the web client and the server when starting up a session is illustrated in Figure 8-2.

Figure 8-2. Client and server communication when starting a session

To use a session in a page, call `session_start( )` at the beginning of your script. Like `setcookie( )`, this function must be called before any output is sent. If you want to use sessions in all your pages, set the configuration directive `session.auto_start` to `On`. (Appendix A explains how to change configuration settings.) Once you do that, there's no need to call `session_start( )` in each page.

8.3 Storing and Retrieving Information

Session data is stored in the `$_SESSION` auto-global array. Read and change elements of that array to manipulate the session data. Example 8-9 shows a page counter that uses the `$_SESSION` array to keep track of how many times a user has looked at the page.

Example 8-9. Counting page accesses with a session

```
session_start( );

$_SESSION['count'] = $_SESSION['count'] + 1;

print "You've looked at this page " . $_SESSION['count'] . ' times.';
```

The first time a user accesses the page in Example 8-9, no `PHPSESSID` cookie is sent by the user's web client to the server. The `session_start( )` function creates a new session for the user and sends a `PHPSESSID` cookie with the new session ID in it. When the session is created, the `$_SESSION` array starts out empty. So, `$_SESSION['count'] = $_SESSION['count'] + 1` sets `$_SESSION['count']` to 1. The `print` statement outputs:

```
You've looked at this page 1 times.
```

At the end of the request, the information in `$_SESSION` is saved into a file on the web server associated with the appropriate session ID.

The next time the user accesses the page, the web client sends the `PHPSESSID` cookie. The `session_start( )` function sees the session ID in the cookie and loads the file that contains the saved session information associated with that session ID. In this case, that saved information just says that `$_SESSION['count']` is 1. Next, `$_SESSION['count']` is incremented to 2 and `You've looked at this page 2 times.` is printed. Again, at the end of the request, the contents of `$_SESSION` (now with `$_SESSION['count']` equal to 2) are saved to a file.

The PHP interpreter keeps track of the contents of `$_SESSION` separately for each session ID. When your program is running, `$_SESSION` contains the saved data for one session only? the active session corresponding to the ID that was sent in the `PHPSESSID` cookie. Each user's `PHPSESSID` cookie has a different value.

As long as you call `session_start( )` at the top of a page (or if `session.auto_start` is on), you have access to a user's session data in your page. The `$_SESSION` array is a way of sharing information between pages.

Example 8-10 is a complete program that displays a form in which a user picks a dish and a quantity. That dish and quantity are added to the session variable `order`.

Example 8-10. Saving form data in a session

```
<?php
require 'formhelpers.php';

session_start( );

$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' => 'Abalone with Marrow and Duck Feet');

if ($_POST['_submit_check']) {
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        process_form( );
```

8.4 Configuring Sessions

Sessions work great with no additional tweaking. Turn them on with the `session_start( )` function or the `session.auto_start` configuration directive, and the `$_SESSION` array is there for your enjoyment. However, if you're more particular about how you want sessions to function, there are a few helpful settings that can be changed.

Session data sticks around as long as the session is accessed at least once every 24 minutes. This is fine for most applications. Sessions aren't meant to be a permanent data store for user information? that's what the database is for. Sessions are for keeping track of recent user activity to make their browsing experience smoother.

Some situations may need a shorter session length, however. If you're developing a financial application, you may want to allow only 5 or 10 minutes of idle time to reduce the chance that an unattended computer can be used by an unauthorized person. If your application doesn't work with very critical data and you have easily distracted users, you may want to set the session length to longer than 24 minutes.

The `session.gc_maxlifetime` configuration directive controls how much idle time is allowed between requests to keep a session active. It's default value is 1,440? there are 1,440 seconds in 24 minutes. You can change `session.gc_maxlifetime` in your server configuration or by calling the `ini_set( )` function from your program. If you use `ini_set( )`, you must call it before `session_start( )`. Example 8-12 shows how to use `ini_set( )` to change the allowable session idle time to 10 minutes.

Example 8-12. Changing allowable session idle time

```
<?php
ini_set('session.gc_maxlifetime',600'); // 600 seconds =  = ten minutes
session_start( );
?>
```

Expired sessions don't actually get wiped out instantly after 24 minutes elapses. Here's how it really works: at the beginning of any request that uses sessions (because the page calls `session_start( )` or `session.auto_start` is on), there is a 1% chance that the PHP interpreter scans through all of the sessions on the server and deletes any that are expired. "A 1% chance" sounds awfully unpredictable for a computer program. It is. But that randomness makes things more efficient. On a busy site, searching for expired sessions to destroy at the beginning of every request would consume too much server power.

You're not stuck with that 1% chance if you'd like expired sessions to be removed more promptly. The `session.gc_probability` configuration directive is the percent chance that the "erase old sessions" routine runs at the start of a request. To have that happen on every request, set it to 100. Like with `session.gc_maxlifetime`, if you use `ini_set( )` to change the value of `session.gc_probability`, you need to do it before `session_start( )`. Example 8-13 shows how to change `session.gc_probability` with `ini_set( )`.

Example 8-13. Changing the expired session cleanup probability

```
<?php
ini_set('session.gc_probability',100); // 100% : clean up on every request
session_start( );
?>
```

If you are activating sessions with the `session.auto_start` configuration directive and you want to change the value of `session.gc_maxlifetime` or `session.gc_probability`, you can't use `ini_set( )` to change those values? you have to do it in your server configuration.



p

PREV < Day Day Up > NEXT



8.6 Why setcookie( ) and session_start( ) Want to Be at the Top of the Page

When a web server sends a response to a web client, most of that response is the HTML document that the browser renders into a web page on your screen: the soup of tags and text that Internet Explorer or Mozilla formats into tables or changes the color or size of. But before that HTML is a section of the response that contains *headers*. These don't get displayed on your screen but are commands or information from the server for the web client. The headers say things such as "this page was generated at such-and-such a time," "please don't cache this page," or (and the one that's relevant here) "please remember that the cookie named `userid` has the value `ralph`."

All of the headers in the response from the web server to the web client have to be at the beginning of the response, before the *response body*, which is the HTML that controls what the browser actually displays. Once some of the body is sent? even one line? no more headers can be sent.

Functions such as `setcookie( )` and `session_start( )` add headers to the response. In order for the added headers to be sent properly, they must be added before any output starts. That's why they must be called before any `print` statements or any HTML appearing outside `<?php ?>` PHP tags.

If any output has been sent before `setcookie( )` or `session_start( )` is called, the PHP interpreter prints an error message that looks like this:

```
Warning: Cannot modify header information - headers already sent by
(output started at /www/htdocs/catalog.php:2) in /www/htdocs/catalog.php on
line 4
```

This means that line 4 of *catalog.php* called a function that sends a header, but something was already printed by line 2 of *catalog.php*.

If you see the "headers already sent" error message, scrutinize your code for errant output. Make sure there are no print statements before you call `setcookie( )` or `session_start( )`. Check that there is nothing before the first `<?php` PHP start tag in the page. Also, check that there is nothing outside the `<?php ?>` tags in any included or required files? even blank lines.

An alternative to hunting down mischievous blank lines in your files is to use *output buffering*. This tells the PHP interpreter to wait to send *any* output until it's finished processing the whole request. Then, it sends any headers that have been set, followed by all the regular output. To enable output buffering, set the `output_buffering` configuration directive to `On` in your server configuration. Web clients have to wait a few additional milliseconds to get the page content from your server, but you save megaseconds fixing your code to have all output happen after calls to `setcookie( )` or `session_start( )`.

With output buffering turned on, you can mix `print` statements, cookie and session functions, HTML outside of `<?php` and `?>` tags, and regular PHP code without getting the "headers already sent" error. The program in Example 8-19 works only when output buffering is turned on. Without it, the HTML printed before the `<?php` start tag triggers the sending of headers, which prevents `setcookie( )` from working properly.

Example 8-19. A program that needs output buffering to work

```
<html>
<head>Choose Your Site Version</head>
<body>
<?php
setcookie('seen_intro', 1);
?>
<a href="/basic.php">Basic</a>
 or
<a href="/advanced.php">Advanced</a>
</body>
</html>
```

8.7 Chapter Summary

Chapter 8 covers:

- Understanding why cookies are necessary to identify a particular web browser to a web server.
- Setting a cookie in a PHP program.
- Reading a cookie value in a PHP program.
- Modifying cookie parameters such as expiration time, path, and domain.
- Deleting a cookie in a PHP program.
- Turning on sessions from a PHP program or in the PHP interpreter configuration.
- Storing information in a session.
- Reading information from a session.
- Saving form data in a session.
- Removing information from a session.
- Configuring session expiration and cleanup.
- Displaying, validating, and processing a validation form.
- Using encrypted passwords.
- Understanding why `setcookie( )` and `session_start( )` must be called before anything is printed.

PREV < Day Day Up > NEXT



8.8 Exercises

1. Make a web page that uses a cookie to keep track of how many times a user has viewed the page. The first time a particular user looks at the page, it should print something like "Number of views: 1." The second time the user looks at the page, it should print "Number of views: 2," and so on.

2. Modify the web page from the first exercise so that it prints out a special message on the 5th, 10th, and 15th time the user looks at the page. Also modify it so that on the 20th time the user looks at the page, it deletes the cookie and the page count starts over.

3. Write a PHP program that displays a form for a user to pick their favorite color from a list of colors. Make another page whose background color is set to the color that the user picks in the form. Store the color value in `$_SESSION` so that both pages can access it.

4. Write a PHP program that displays an order form. The order form should list six products. Next to each product name there should be a text box into which a user can type in how many of that product they want to order. When the form is submitted, the submitted form data should be saved into the session. Make another page that displays the contents of the saved order, a link back to the order form page, and a Check Out button. If the link back to the order form page is clicked, the order form page should be displayed with the saved order quantities from the session in the text boxes. When the Check Out button is clicked, the order should be cleared from the session.

Chapter 9. Handling Dates and Times

Dates and times are all over the place in a web application. In a shopping cart, you need to handle shipping dates of products. In a forum, you need to keep track of when messages are posted. In all sorts of applications, you need to keep track of the last time a user logged in so that you can tell them things such as "fifteen new messages were posted since you last logged in."

Handling dates and times properly in your programs is more complicated than handing strings or numbers. A date or a time is not a single value but a collection of values? month, day, and year, for example, or hour, minute, and second. Because of this, doing math with them can be tricky. Instead of just adding or subtracting entire dates and times, you have to consider their component parts and what the allowable values for each part are. Hours go up to 12 (or 24), minutes and seconds go up to 59, and not all months have the same number of days.

A programming convention that simplifies date and time calculation is to treat a particular time and date as a single value: the number of seconds that have elapsed since midnight on January 1, 1970. This value is called an *epoch timestamp*. The choice of January 1, 1970 is mostly arbitrary. But, as is the way with conventions, since lots of other people are doing it, you've got to do it, too. Fortunately, PHP provides plenty of functions for you to deal with epoch timestamps.

In this book, the phrase *time parts* (or *date parts* or *time and date parts*) means an array or group of time and date components such as day, month, year, hour, minute, and second. *Formatted time string* (or *formatted date string*, etc.) means a string that contains some particular grouping of time and date parts? for example "Wednesday, October 20, 2004" or "3:54 p.m."

9.1 Displaying the Date or Time

The simplest display of date or time is telling your users what time it is. Use the `date( )` or `strftime( )` function as shown in Example 9-1.

Example 9-1. What time is it?

```
print 'strftime( ) says: ';
print strftime('%c');
print "\n";
print 'date( ) says:';
print date('r');
```

At noon on October 20, 2004, Example 9-1 prints:

```
strftime( ) says: Wed Oct 20 12:00:00 2004
date( ) says: Wed, 20 Oct 2004 12:00:00 -0400
```

Both `strftime( )` and `date( )` take two arguments. The first controls how the time or date string is formatted, and the second controls what time or date to use. If you leave out the second argument, as in Example 9-1, each uses the current time.

With `date( )`, individual letters in the format string translate into certain time values. Example 9-2 prints out a month, day, and year with `date( )`.

Example 9-2. Printing a formatted date string with date( )

```
print date('m/d/y');
```

At noon on October 20, 2004, Example 9-2 prints:

```
10/20/04
```

In Example 9-2, the `m` becomes the month (10), the `d` becomes the day of the month (20), and the `y` becomes the two-digit year (04). Because the slash is not a format character that `date( )` understands, it is left alone in the string that `date( )` returns.

With `strftime( )`, the things in the format string that get replaced by time and date values are set off by percent signs.[1] Example 9-3 prints out a month, day, and year with `strftime( )`.

[1] This makes `strftime( )` format strings look like `printf( )` format strings, but they're different. The modifiers that work with `printf( )` don't work with `strftime( )`.

Example 9-3. Printing a formatted date string with strftime( )

```
print strftime('%m/%d/%y');
```

At noon on October 20, 2004, Example 9-3 prints:

```
10/20/04
```

In Example 9-3, the `%m` becomes the month, the `%d` becomes the day, and `%y` becomes the two-digit year.

Table 9-1 lists all of the special characters that `date( )` and `strftime( )` understand. The "Windows?" column indicates whether the character is supported by `strftime( )` on Windows.

Table 9-1. strftime( ) and date( ) format characters

Type	strftime( )	date( )	Description	Range	Windows?

9.2 Parsing a Date or Time

To work with date or time values in your program as epoch timestamps, you need to convert other time representations to epoch timestamps. If you have discrete date or time parts (for example, from different form parameters), then use `mktime( )`. It accepts an hour, minute, second, month, day, and year, and returns the corresponding epoch timestamp. Example 9-6 shows `mktime( )` at work.

Example 9-6. Making an epoch timestamp

```
// get the values from a form
$user_date = mktime($_POST['hour'], $_POST['minute'], 0, $_POST['month'],
$_POST['day'],
$_POST['year']);

// 1:30 pm (and 45 seconds) on October 20, 1982
$afternoon = mktime(13,30,45,10,20,1982);

print strftime('At %I:%M:%S on %m/%d/%y, ', $afternoon);
print "$afternoon seconds have elapsed since 1/1/1970.";
```

Example 9-6 prints:

```
At 01:30:45 on 10/20/82, 403983045 seconds have elapsed since 1/1/1970.
```

All of `mktime( )`'s arguments are optional. Whatever is left out defaults to the current date or time. For example, `mktime(15,30,0)` returns the epoch timestamp for 3:30 p.m. today, and `mktime(15,30,0,6,5)` returns the epoch timestamp for 3:30 p.m. on June 5th of this year.

When you want the epoch timestamp for something relative to a time you know, use `strtotime( )`. It understands English descriptions of relative times and returns an appropriate epoch timestamp. Example 9-7 shows how to find the epoch timestamp for some dates with `strtotime( )`.

Example 9-7. Using strtotime( )

```
$now = time( );
$later = strtotime('Thursday',$now);
$before = strtotime('last thursday',$now);
print strftime("now: %c \n", $now);
print strftime("later: %c \n", $later);
print strftime("before: %c \n", $before);
```

At noon on October 20, 2004, Example 9-7 prints:

```
now: Wed Oct 20 12:00:00 2004
later: Thu Oct 21 00:00:00 2004
before: Thu Oct 14 00:00:00 2004
```

Like `date( )` and `strftime( )`, `strtotime( )` also accepts an epoch timestamp second argument to use as the starting point for its calculations. Example 9-8 uses `mktime( )` and `strtotime( )` to find when the U.S. presidential election will be in 2008. U.S. presidential elections are held the Tuesday after the first Monday in November.

Example 9-8. Using strtotime( ) with a starting epoch timestamp

```
// Find the epoch timestamp for November 1, 2008
$november = mktime(0,0,0,11,1,2008);
// Find the First monday on or after November 1, 2008
$monday = strtotime('Monday', $november);
// Skip ahead one day to the Tuesday after the first Monday
$election_day = strtotime('+1 day', $monday);
```

9.3 Dates and Times in Forms

When you need a user to input a date in a form, the best thing to do is to use `<select>` menus. This generally restricts the possible input to whatever you display in the menus. The specific date or time information you need controls what you populate the `<select>` menus with.

9.3.1 A Single Menu with One Choice Per Day

If there are a small number of choices, you can have just one menu that lists all of them. Example 9-9 prints a `<select>` menu that lets a user pick one day in the coming week. The value for each option in the menu is an epoch timestamp corresponding to midnight on the displayed day.

Example 9-9. A day choice <select> menu

```
$midnight_today = mktime(0,0,0);
print '<select name="date">';
for ($i = 0; $i < 7; $i++) {
    $timestamp = strtotime("+$i day", $midnight_today);
    $display_date = strftime('%A, %B %d, %Y', $timestamp);
    print '<option value="' . $timestamp .'">'.$display_date."</option>\n";
}
print "\n</select>";
```

On October 20, 2004, Example 9-9 prints:

```
<select name="date"><option value="1098244800">Wednesday, October 20,
2004</option>
<option value="1098331200">Thursday, October 21, 2004</option>
<option value="1098417600">Friday, October 22, 2004</option>
<option value="1098504000">Saturday, October 23, 2004</option>
<option value="1098590400">Sunday, October 24, 2004</option>
<option value="1098676800">Monday, October 25, 2004</option>
<option value="1098763200">Tuesday, October 26, 2004</option>
```

If you're using the `input_select( )` form helper function from Chapter 6, put the timestamps and display dates in an array inside the `for( )` loop and then pass that array to `input_select( )`, as shown in Example 9-10.

Example 9-10. A day choice menu with input_select( )

```
require 'formhelpers.php';

$midnight_today = mktime(0,0,0);
$choices = array( );
for ($i = 0; $i < 7; $i++) {
    $timestamp = strtotime("+$i day", $midnight_today);
    $display_date = strftime('%A, %B %d, %Y', $timestamp);
    $choices[$timestamp] = $display_date;
}
input_select('date', $_POST, $choices);
```

Example 9-10 prints the same menu as Example 9-9.

9.3.2 Multiple Menus for Month, Day, and Year

To let a user enter an arbitrary date, provide separate menus for month, day, and year, as shown in Example 9-11.

Example 9-11. Multiple <select> menus for date picking

```
$months = array(1 => 'January', 2 => 'February', 3 => 'March', 4 => 'April',
                5 => 'May', 6 => 'June', 7 => 'July', 8 => 'August',
```

9.4 Displaying a Calendar

This section puts the date and time functions to work in displaying a calendar. The `show_form( )` function in Example 9-17 displays a form that asks for a month and year. The `process_form( )` function hands those values off to the `show_calendar( )` function, which does the real work of printing a calendar grid for a particular month.

The structure of the `if( )` statement that controls `show_form( )`, `validate_form( )`, and `process_form( )` is different in Example 9-17 than in previous form examples. That's because we want to display the form above the calendar. Usually, if the form data is valid, `show_form( )` is not called? only `process_form( )` is. But here, `show_form( )` is called before `process_form( )` so that the form is displayed above the calendar and the user can pick another month and year to view.

Similarly, the call to `show_form( )` that happens when the form has not been submitted (when there is no `$_POST['_submit_check']` parameter) is followed by a call to `show_calendar( )` to display the calendar for the current month the first time the page is loaded.

Example 9-17. Printing a calendar

```
<?php
// Use the form helper functions defined in Chapter 6
require 'formhelpers.php';

$months = array(1 => 'January', 2 => 'February', 3 => 'March', 4 => 'April',
                5 => 'May', 6 => 'June', 7 => 'July', 8 => 'August',
                9 => 'September', 10 => 'October', 11 => 'November',
                12 => 'December');

$years = array( );
for ($year = date('Y') - 1, $max_year = date('Y') + 5; $year < $max_year;
$year++) {
    $years[$year] = $year;
}

if ($_POST['_submit_check']) {
    if ($errors = validate_form( )) {
        show_form($errors);
    } else {
        show_form( );
        process_form( );
    }
} else {
    // When nothing is submitted, show the form and then
    // a calendar for the current month
    show_form( );
    show_calendar(date('n'), date('Y'));
}

function validate_form( ) {
    global $months, $years;
    $errors = array( );

    if (! array_key_exists($_POST['month'], $months)) {
        $errors[  ] = 'Select a valid month.';
    }

    if (! array_key_exists($_POST['year'], $years)) {
        $errors[  ] = 'Select a valid year.';
    }

    return $errors;
}
```

9.5 Chapter Summary

Chapter 9 covers:

- Defining some time- and date-handling vocabulary such as *epoch timestamp*, *time and date parts*, and *formatted time and date string*.
- Printing formatted time and date strings with `strftime( )` and `date( )`.
- Making an epoch timestamp with `mktime( )`.
- Making an epoch timestamp with `strtotime( )`.
- Displaying form elements to allow for date or time input.
- Doing calculations with a date or time submitted in a form.
- Displaying a calendar.

9.6 Exercises

1. Use `strftime( )` to print a formatted time and date string that looks like this:
2. ```
   Today is day 20 of October and day 294 of the year 2004. The time is
   07:45 PM
   (also known as 19:45).
   ```

   To make your output exactly match the example, use `mktime( )` to get the epoch timestamp for 7:45 p.m. on October 20, 2004.
3. Use `date( )` to print the same formatted time and date string.
4. The U.S. holiday Labor Day is the first Monday in September. Print out a table of the dates that Labor Day falls from 2004 to 2020.
5. Write a PHP program that displays a form in which users select a day, month, and year in the future. Print out a list of all the Tuesdays between the current date and the date the user submits in the form.
   ```

Chapter 10. Working with Files

The data storage destination of choice for a web application is a database. That doesn't mean that you're completely off the hook from dealing with regular old files, though. Plain text files are still a handy, universal way to exchange some kinds of information.

You can do some easy customization of your web site by storing HTML templates in text files. When it's time to generate a specialized page, load the text file, substitute real data for the template elements, and print it. Example 10-1 shows you how to do this.

Files are also good for importing or exporting tabular data between your program and a spreadsheet. In your PHP programs, you can easily read and write the CSV ("comma-separated value") files with which spreadsheet programs work.

Working with files in PHP also means working with remote web pages. A great thing about file handling in PHP is you can open a remote file on another computer as easily as you can open a file that sits on your web server. Most file-handling functions in PHP understand URLs as well as local filenames. However, for this feature to work, the `allow_url_fopen` configuration directive must be enabled. It is enabled by default, but if you're having problems loading a remote file, check this setting.

10.1 Understanding File Permissions

To read or write a file with any of the functions you'll learn about in this chapter, the PHP interpreter must have permission from the operating system to do so. Every program that runs on a computer, including the PHP interpreter, runs with the privileges of a particular user account. Most of the user accounts correspond to people. When you log in to your computer and start up your word processor, that word processor runs with the privileges that correspond to your account: it can read files that you are allowed to see and write files that you are allowed to change.

Some user accounts on a computer, however, aren't for people, but for system processes such as web servers. When the PHP intepreter runs inside of a web server, it has the privileges that the web server's "account" has. So if the web server is allowed to read a certain file or directory, then the PHP interpreter (and therefore your PHP program) can read that file or directory. If the web server is allowed to change a certain file or write new files in a particular directory, then so can the PHP interpreter and your PHP program.

Usually, the privileges extended to a web server's account are more limited than the privileges that go along with a real person's account. The web server (and the PHP interpreter) need to be able to read all of the PHP program files that make up your web site, but they shouldn't be able to change them. If a bug in the web server or an insecure PHP program lets an attacker break in, the PHP program files should be protected against being changed by that attacker.

In practice, what this means is that your PHP programs shouldn't have too much trouble reading most files that you need to read. (Of course, if you try to read another user's private files, you may run into a problem? but that's as it should be!) However, the files that your PHP program can change and the directories into which your program can write new files are limited. If you need to create lots of new files in your PHP programs, work with your system administrator to make a special directory that you can write to but that doesn't compromise system security. Section 10.5, later in this chapter, shows you how to determine what files and directories your programs are allowed to read and write.

10.2 Reading and Writing Entire Files

This section shows you how to work with an entire file at once, as opposed to manipulating just a few lines of a file. PHP provides special functions for reading or writing a whole file in a single step.

10.2.1 Reading a File

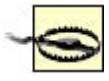
To read the contents of a file into a string, use `file_get_contents( )`. Pass it a filename, and it returns a string containing everything in the file. Example 10-1 reads the file in Example 10-2 with `file_get_contents( )`, modifies it with `str_replace( )`, and then prints the result.

Example 10-1. Using file_get_contents( ) with a page template

```
// Load the file from Example 10.2
$page = file_get_contents('page-template.html');

// Insert the title of the page
$page = str_replace('{page_title}', 'Welcome', $page);

// Make the page blue in the afternoon and
// green in the morning
if (date('H' >= 12)) {
    $page = str_replace('{color}', 'blue', $page);
} else {
    $page = str_replace('{color}', 'green', $page);
}

// Take the username from a previously saved session
// variable
$page = str_replace('{name}', $_SESSION['username'], $page);

// Print the results
print $page;
```

Example 10-2. page-template.html for Example 10-1

```
<html>
<head><title>{page_title}</title></head>
<body bgcolor="{color}">

<h1>Hello, {name}</h1>

</body>
</html>
```



Every time you use a file access function, you need to check that it didn't encounter an error because of a lack of disk space, permission problem, or other failure. Error checking is discussed in detail later in Section 10.6. The examples in the next few sections don't have error-checking code, so you can see the actual file access function at work without other new material getting in the way. Real programs that you write always need to check for errors after calling a file access function.

With `$_SESSION['username']` set to `Jacob`, Example 10-1 prints:

```
<html>
<head><title>Welcome</title></head>
<body bgcolor="green">

<h1>Hello, Jacob</h1>
```

10.3 Reading and Writing Parts of Files

The `file_get_contents( )` and `file_put_contents( )` functions are fine when you want to work with an entire file at once. But when it's time for precision work, use other functions to deal with a file a line at a time. Example 10-6 reads a file in which each line contains a name and an email address and then prints an HTML-formatted list of that information.

Example 10-6. Reading a file a line at a time

```
$fh = fopen('people.txt','rb');
for ($line = fgets($fh); ! feof($fh); $line = fgets($fh)) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto:' . $info[0] . '">' . $info[1] ."</li>\n";
}
fclose($fh);
```

If *people.txt* contains what's listed in Example 10-7, then Example 10-6 prints:

```
<li><a href="mailto:alice@example.com">Alice Liddell</li>
<li><a href="mailto:bandersnatch@example.org">Bandersnatch Gardner</li>
<li><a href="mailto:charles@milk.example.com">Charlie Tenniel</li>
<li><a href="mailto:dodgson@turtle.example.com">Lewis Humbert</li>
```

Example 10-7. people.txt for Example 10-6

```
alice@example.com|Alice Liddell
bandersnatch@example.org|Bandersnatch Gardner
charles@milk.example.com|Charlie Tenniel
dodgson@turtle.example.com|Lewis Humbert
```

The four file access functions in Example 10-6 are `fopen( )`, `fgets( )` , `feof( )`, and `fclose( )`. The `fopen( )` function opens a connection to the file and returns a variable that's used for subsequent access to the file in the program. (This is very similar to the database connection variable returned by `DB::connect( )` that you saw in Chapter 7.) The `fgets( )` function reads a line from the file and returns it as a string. The PHP interpreter keeps a bookmark of where its current position in the file is. The bookmark starts at the beginning of the file, so the first time that `fgets( )` is called, the first line of the file is read. After that line is read, the bookmark is updated to the beginning of the next line. The `feof( )` function returns `true` if the bookmark is past the end of the file. ("eof" stands for "end of file.") Last, the `fclose( )` function closes the connection to the file.

The `for( )` loop in Example 10-6 may look a little funny, but its structure ensures that `fgets( )` and `feof( )` play nice together. When the `for( )` loop starts, the initialization expression runs. This reads the first line from the file and stores it in `$line`. Then the test expression runs: `! feof($fh)`. This is `true` when `feof($fh)` returns `false`? in other words, when the bookmark is not past the end of the file. Next the loop body runs, doing some things with `$line`. After the loop body is done, the iteration expression runs and stores the next line of the file in `$line`.

Everything moves along line by line in the `for( )` loop until the last line of the file has been read by the iteration expression. The code block runs one more time, and the `<li><a href="mailto:dodgson@turtle.example.com">Lewis Humbert</li>` line of HTML is printed. Then, `fgets( )` is called in the iteration expression. At this point, though, there's nothing left in the file, so `fgets( )` returns `false` and puts the bookmark past the end of the file. Now, when `feof( )` is called in the test expression, it sees where the bookmark is and returns `true`. This ends the `for( )` loop.

Example 10-6 uses `trim( )` on `$line` because the string that `fgets( )` returns includes the trailing newline at the end of the line. The `trim( )` function removes the newline, which makes the output look better.

The first argument to `fopen( )` is the name of the file that you want to access. Use forward

10.4 Working with CSV Files

One type of text file gets special treatment in PHP: the CSV file. It can't handle graphs or charts, but excels for sharing tables of data among different programs. To read a line of a CSV file, use `fgetcsv( )` instead of `fgets( )`. It reads a line from the CSV file and returns an array containing each field in the line. Example 10-10 is a CSV file of information about restaurant dishes. Example 10-11 uses `fgetcsv( )` to read the file and insert the information in it into the `dishes` database table from Chapter 7.

Example 10-10. dishes.csv for Example 10-11

```
"Fish Ball with Vegetables",4.25,0
"Spicy Salt Baked Prawns",5.50,1
"Steamed Rock Cod",11.95,0
"Sauteed String Beans",3.15,1
"Confucius ""Chicken""",4.75,0
```

Example 10-11. Inserting CSV data into a database table

```
require 'DB.php';
// Connect to the database
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
// Open the CSV file
$fh = fopen('dishes.csv','rb');

for ($info = fgetcsv($fh, 1024); ! feof($fh); $info = fgetcsv($fh, 1024)) {
    // $info[0] is the dish name    (the  first field in a line of dishes.csv)
    // $info[1] is the price        (the second field)
    // $info[2] is the spicy status (the  third field)
    // Insert a row into the database table
    $db->query("INSERT INTO dishes (dish_name, price, is_spicy) VALUES (?, ?,
?)",
               $info);
    print "Inserted $info[0]\n";
}
// Close the file
fclose($fh);
```

Example 10-11 prints:

```
Inserted Fish Ball with Vegetables
Inserted Spicy Salt Baked Prawns
Inserted Steamed Rock Cod
Inserted Sauteed String Beans
Inserted Confucius "Chicken"
```

The second argument to `fgetcsv( )` is a line length. This value needs to be longer than the length of the longest line in the CSV file. Example 10-11 uses 1024, which is plenty longer than any of the lines in Example 10-10. If you might have lines longer than 1K in a CSV file, pick a bigger length, such as 1048576 (1 MB).

Writing a CSV-formatted line is trickier than reading one. There's no built-in function for it, so you've got to format the line yourself. Example 10-12 contains a `make_csv_line( )` function that accepts an array of values as an argument and returns a CSV-formatted string containing those values.

Example 10-12. Making a CSV-formatted string

```
function make_csv_line($values) {
    // If a value contains a comma, a quote, a space, a
    // tab (\t), a newline (\n), or a linefeed (\r),
    // then surround it with quotes and replace any quotes inside
    // it with two quotes
    foreach($values as $i => $value) {
        if ((strpos($value, ' ') != = false) ||
```

10.5 Inspecting File Permissions

As mentioned at the beginning of the chapter, your programs can only read and write files when the PHP interpreter has permission to do so. You don't have to cast about blindly and rely on error messages to figure out what those permissions are, however. PHP gives you functions with which you can determine what your program is allowed to do.

To check whether a file or directory exists, use `file_exists( )`. Example 10-16 uses this function to report whether a directory's index file has been created.

Example 10-16. Checking the existence of a file

```
if (file_exists('/usr/local/htdocs/index.html')) {
    print "Index file is there.";
} else {
    print "No index file in /usr/local/htdocs.";
}
```

To determine whether your program has permission to read or write a particular file, use `is_readable( )` or `is_writeable( )`. Example 10-17 checks that a file is readable before retrieving its contents with `file_get_contents( )`.

Example 10-17. Testing for read permission

```
$template_file = 'page-template.html';
if (is_readable($template_file)) {
    $template = file_get_contents($template_file);
} else {
    print "Can't read template file.";
}
```

Example 10-18 verifies that a file is writable before appending a line to it with `fopen( )` and `fwrite( )`.

Example 10-18. Testing for write permission

```
$log_file = '/var/log/users.log';
if (is_writeable($log_file)) {
    $fh = fopen($log_file,'ab');
    fwrite($fh, $_SESSION['username'] . ' at ' . strftime('%c') . "\n");
    fclose($fh);
} else {
    print "Cant write to log file.";
}
```

10.6 Checking for Errors

So far, the examples in this chapter have been shown without any error checking in them. This keeps them shorter, so you can focus on the file manipulation functions such as `file_get_contents( )`, `fopen( )`, and `fgetcsv( )`. It also makes them somewhat incomplete. Just like talking to a database program, working with files means interacting with resources external to your program. This means you have to worry about all sorts of things that can cause problems, such as operating system file permissions or a disk running out of free space.

In practice, to write robust file-handling code, you should check the return value of each file-related function. They each generate a warning message and return `false` if there is a problem. If the configuration directive `track_errors` is on, the text of the error message is available in the global variable `$php_errormsg`.

Example 10-19 shows how to check whether `fopen( )` or `fclose( )` encounters an error.

Example 10-19. Checking for an error from fopen( ) or fclose( )

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');

// Open dishes.txt for writing
$fh = fopen('/usr/local/dishes.txt','wb');
if (! $fh) {
    print "Error opening dishes.txt: $php_errormsg";
} else {
    $q = $db->query("SELECT dish_name, price FROM dishes");
    while($row = $q->fetchRow( )) {
        // Write each line (with a newline on the end) to
        // dishes.txt
        fwrite($fh, "The price of $row[0] is $row[1] \n");
    }
    if (! fclose($fh)) {
        print "Error closing dishes.txt: $php_errormsg";
    }
}
```

If your program doesn't have permission to write into the */usr/local* directory, then `fopen( )` returns false, and Example 10-19 prints:

```
Error opening dishes.txt: failed to open stream: Permission denied
```

It also generates a warning message that looks like this:

```
Warning: fopen(/usr/local/dishes.txt): failed to open stream: Permission
denied in
dishes.php on line 5
```

Section 12.1 talks about how to control where the warning message is shown.

The same thing happens with `fclose( )`. If it returns `false`, then the `Error closing dishes.txt` message is printed. Sometimes operating systems buffer data written with `fwrite( )` and don't actually save the data to the file until you call `fclose( )`. If there's no space on the disk for the data you're writing, the error might show up when you call `fclose( )`, not when you call `fwrite( )`.

Checking for errors from the other file-handling functions (`fgets( )`, `fwrite( )`, `fgetcsv( )`, `file_get_contents( )`, and `file_put_contents( )`) is a little trickier. This is because you have to do something special to distinguish the value they each return when an error happens from the data they each return when everything goes OK.

If something goes wrong with `fgets( )`, `file_get_contents( )`, or `fgetcsv( )`, they each return `false`. However, it's possible that these functions could succeed and still return a

10.7 Sanitizing Externally Supplied Filenames

Just like data submitted in a form or URL can cause problems when it is displayed (cross-site scripting attack) or put in an SQL query (SQL injection attack), it can also cause problems when it is used as a filename or as part of a filename. It doesn't have a fancy name like those other attacks, but it can be just as devastating.

The cause of the problem is the same: there are special characters that must be escaped so they lose their special meaning. In filenames, the special characters are `/` (which separates parts of filenames), and the two-character sequence `..` (which means "go up one directory" in a filename).

For example, the funny-looking filename */usr/local/data/../../../etc/passwd* doesn't point to a file under the */usr/local/data* directory but instead to the file */etc/passwd*, which, on most Unix systems, contains a list of user accounts. The filename */usr/local/data/../../../etc/passwd* means "from the directory */usr/local/data*, go up one level (to */usr/local*), then go up another level (to */usr*), then go up another level (to */*, the top level of the filesystem), then down into */etc*, then stop at the file *passwd*."

How could this be a problem in your PHP programs? When you use data from a form in a filename, you are vulnerable to this sort of attack unless you sanitize that submitted form data. Example 10-23 takes the approach of removing all forward slashes and `..` sequences from a submitted form parameter before incorporating the parameter into a filename.

Example 10-23. Cleaning up a form parameter that goes in a filename

```
// Remove slashes from user
$user = str_replace('/', '', $_POST['user']);
// Remove .. from user
$user = str_replace('..', '', $user);

print 'User profile for ' . htmlentities($user) .': <br/>';
print file_get_contents("/usr/local/data/$user");
```

If a malicious user supplies `../../../etc/passwd` as the `user` form parameter in Example 10-23, that is translated into `etcpasswd` before being interpolated into the filename used with `file_get_contents( )`.

Another helpful technique for getting rid of user-entered nastiness is to use `realpath( )`. It translates an obfuscated filename that contains `..` sequences into the `..`-less version of filename that more directly indicates where the file is. For example, `realpath('/usr/local/data/../../../etc/passwd')` returns the string `/etc/passwd`. You can use `realpath( )` as in Example 10-24: to see whether filenames, after incorporating form data, are acceptable.

Example 10-24. Cleaning up a file name with realpath( )

```
$filename = realpath("/usr/local/data/$_POST[user]");

// Make sure that $filename is under /usr/local/data
if ('/usr/local/data/' =  = substr($filename, 0, 16)) {
    print 'User profile for ' . htmlentities($_POST['user']) .': <br/>';
    print file_get_contents($filename);
} else {
    print "Invalid user entered.";
}
```

In Example 10-24, if `$_POST['user']` is `james`, then `$filename` is set to `/usr/local/data/james` and the `if( )` code block runs. However, if `$_POST['user']` is something suspicious such as `../secrets.txt`, then `$filename` is `/usr/local/secrets.txt`, and the `if( )` test fails, so `Invalid user entered` is printed.

10.8 Chapter Summary

Chapter 10 covers:

- Understanding where the PHP interpreter's file access permissions come from.
- Reading entire local and remote files with `file_get_contents( )`.
- Writing entire local and remote files with `file_put_contents( )`.
- Opening and closing files with `fopen( )` and `fclose( )`.
- Reading a line of a file with `fgets( )`.
- Using `feof( )` and a `for( )` loop to read each line in a file.
- Using forward slashes in filenames with all operating systems.
- Providing different file modes to `fopen( )`.
- Writing data to a file with `fwrite( )`.
- Reading a line of a CSV file with `fgetcsv( )`.
- Determining whether a file exists with `file_exists( )`.
- Inspecting file permissions with `is_readable( )` and `is_writeable( )`.
- Checking for errors returned from file access functions.
- Understanding when to check a return value with the identical operator (`= = =`).
- Removing potentially dangerous parts of externally supplied filenames.

10.9 Exercises

1. Outside of the PHP interpreter, create a new template file in the style of Example 10-2. Use `file_get_contents( )` and `file_put_contents( )` to read an HTML template file, substitute values for the template variables, and save the new page to a separate file.

2. Outside of the PHP interpreter, create a file that contains some email addresses, one per line. Make sure a few of the addresses appear more than once in the file. Call that file *addresses.txt*. Then, write a PHP program that reads each line in *addresses.txt* and counts how many times each address appears. For each distinct address in *addresses.txt*, your program should write a line to another file, *addresses-count.txt*. Each line in *addresses-count.txt* should consist of the number of times an address appears in *addresses.txt*, a comma, and the email address. Write the lines to *addresses-count.txt* in sorted order from the address that occurs the most times in *addresses.txt* to the address that occurs the fewest times in *addresses.txt*.

3. Display a CSV file as an HTML table. If you don't have a CSV file (or spreadsheet program) handy, use the data from Example 10-10.

4. Write a PHP program that displays a form that asks a user for the name of a file underneath the web server's document root directory. If that file exists on the server, is readable, and is underneath the web server's document root directory, then display the contents of the file. For example, if the user enters `article.html`, display the file *article.html* in the document root directory. If the user enters `catalog/show.php`, display the file *show.php* in the directory *catalog* under the document root directory. Table 6-1 tells you how to find the web server's document root directory.

5. Modify your solution to the previous exercise so that the program displays only files whose names end in *.html*. Letting users look at the PHP source code of any page on your site can be dangerous if those pages have sensitive information in them such as database usernames and passwords.

Chapter 11. Parsing and Generating XML

With XML, you can effortlessly exchange data between programs written in different languages, running on different operating systems, located on computers anywhere in the world. At least, that's what enthusiastic computer programmers and salespeople who work for companies that sell XML tools will tell you. They're sort of telling the truth. XML does make it easier to trade structured information between two programs. But you still have to do some work to herd your data into the right structure. This chapter shows you how to do that work with PHP.

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by `<` and `>`. There are two big differences between XML and HTML:

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

In one sense, XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the `<a></a>` tags surround a link, the `<ul></ul>` tags denote an unordered list, the `<li></li>` tags indicate a list element, and so on. An XML document, however, can use any tags you want. Put `<rating></rating>` tags around a movie rating, `<height></height>` tags around someone's height, or `<favoritecolor></favoritecolor>` tags around someone's favorite color? XML doesn't care. Of course, whomever (or whatever program) you're sharing the XML document with also needs to agree to use and understand the same set of tags.

While you get more freedom in the tag-choice department, XML clamps down much harder than HTML when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. The HTML list in Example 11-1 renders just fine in a web browser.

Example 11-1. HTML list that's not valid XML

```
<ul>
  <li>Braised Sea Cucumber
  <li>Baked Giblets with Salt
  <li>Abalone with Marrow and Duck Feet
</ul>
```

As an XML document, though, Example 11-1 has a problem. There are no closing `</li>` tags to match up with the three opening `<li>` tags. Every opened tag in an XML document must be closed. The XML-friendly way to write Example 11-1 is shown in Example 11-2.

Example 11-2. HTML list that is valid XML

```
<ul>
  <li>Braised Sea Cucumber</li>
  <li>Baked Giblets with Salt</li>
  <li>Abalone with Marrow and Duck Feet</li>
</ul>
```

There are lots of existing standard XML tag sets for describing different kinds of information. XHTML, an XML-compatible version of HTML, is described at http://www.w3.org/TR/xhtml11/. Lots of web sites distribute lists of article headlines or other syndicated data using an XML format called RSS (described at http://blogs.law.harvard.edu/tech/rss). Many of the examples in this chapter also involve RSS. You can get a PHP-themed RSS feed from the Planet PHP web site, which collects many PHP-related blogs. The Planet PHP RSS feed is available at http://www.planet-php.net/rss/.

To learn more about XML, check out *Learning XML* by Erik T. Ray (O'Reilly). To learn more about XML in PHP, read Chapter 11 of *Programming PHP* by Rasmus Lerdorf and Kevin Tatroe (O'Reilly), Chapter 12 of *PHP Cookbook* by David Sklar and Adam Trachtenberg (O'Reilly), or Chapter 5 of *Upgrading to PHP 5* by Adam Trachtenberg (O'Reilly).

11.1 Parsing an XML Document

PHP 5's new SimpleXML module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.

To create a SimpleXML object from an XML document stored in a string, pass the string to `simplexml_load_string( )`. It returns a SimpleXML object. In Example 11-3, `$channel` holds XML that represents the `<channel>` part of an RSS 0.91 feed.

Example 11-3. Parsing XML in a string

```
$channel =<<<_XML_
<channel>
 <title>What's For Dinner</title>
 <link>http://menu.example.com/</link>
 <description>These are your choices of what to eat tonight.</description>
</channel>
_XML_;

$xml = simplexml_load_string($channel);
```

The contents of XML elements are available as the data stored in the SimpleXML object. Example 11-4 prints some data inside the `$xml` object created in Example 11-3.

Example 11-4. Printing XML element contents

```
print "The $xml->title channel is available at $xml->link. ";
print "The description is \"$xml->description\"";
```

Example 11-4 prints:

```
The What's For Dinner channel is available at http://menu.example.com/. The
description is "These are your choices of what to eat tonight."
```

To descend into the hierarchy of XML elements, chain together the element names with arrows. Example 11-5 loads a full RSS feed into a SimpleXML object and prints channel information.

Example 11-5. Printing subelement contents

```
$menu=<<<_XML_
<?xml version="1.0" encoding="utf-8" ?>
<rss version="0.91">
 <channel>
  <title>What's For Dinner</title>
  <link>http://menu.example.com/</link>
  <description>These are your choices of what to eat tonight.</description>
  <item>
   <title>Braised Sea Cucumber</title>
   <link>http://menu.example.com/dishes.php?dish=cuke</link>
   <description>Gentle flavors of the sea that nourish and refresh
you.</description>
  </item>
  <item>
   <title>Baked Giblets with Salt</title>
   <link>http://menu.example.com/dishes.php?dish=giblets</link>
   <description>Rich giblet flavor infused with salt and spice.</description>
  </item>
  <item>
   <title>Abalone with Marrow and Duck Feet</title>
   <link>http://menu.example.com/dishes.php?dish=abalone</link>
   <description>There's no mistaking the special pleasure of
abalone.</description>
  </item>
 </channel>
```

11.2 Generating an XML Document

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch. For many XML documents, the easiest way to generate them is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example 11-17 generates the XML for the channel part of an RSS feed using the information in the `$channel` array.

Example 11-17. Generating XML from an array

```
$channel = array('title' => "What's For Dinner",
                 'link' => 'http://menu.example.com/',
                 'description' => 'These are your choices of what to eat
tonight.');

print "<channel>\n";
foreach ($channel as $element => $content) {
    print " <$element>";
    print htmlentities($content);
    print "</$element>\n";
}
print "</channel>";
```

Example 11-17 prints:

```
<channel>
 <title>What's For Dinner</title>
 <link>http://menu.example.com/</link>
 <description>These are your choices of what to eat tonight.</description>
</channel>
```

Any text content of XML elements must be encoded by `htmlentities( )` before it is printed. Just as characters such as < and > have special meaning in HTML, they also have special meaning in XML.

You can use a similar technique to generate XML from information that you retrieve from a database table. Example 11-18 makes an XML representation of the data about spicy dishes.

Example 11-18. Formatting information from a database table as XML

```
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');

// Change the fetch mode to string-keyed arrays
$db->setFetchMode(DB_FETCHMODE_ASSOC);

print "<dishes>\n";
$q = $db->query("SELECT dish_id, dish_name, price FROM dishes WHERE is_spicy
= 1");
while($row = $q->fetchRow( )) {
    print ' <dish id="' . htmlentities($row['dish_id']) .'">' . "\n";
    print '  <name>' . htmlentities($row['dish_name'])."</name>\n";
    print '  <price>' . htmlentities($row['price'])."</price>\n";
    print " </dish>\n";
}
print '</dishes>';
```

Example 11-18 prints:

```
<dishes>
 <dish id="4">
  <name>Eggplant with Chili Sauce</name>
  <price>6.50</price>
```

11.3 Chapter Summary

Chapter 11 covers:

- Understanding the basic differences between XML and HTML.
- Creating a SimpleXML object from a string that contains XML.
- Printing XML element contents with a SimpleXML object.
- Printing XML element attributes with a SimpleXML object.
- Accessing identically named elements with a SimpleXML object.
- Looping through a SimpleXML object with `foreach( )`.
- Changing elements and attributes in a SimpleXML object.
- Printing a SimpleXML object as an XML document.
- Sending a `Content-Type` header to indicate an XML document.
- Creating a SimpleXML object from a local or remote file that contains XML.
- Saving a SimpleXML object to a file as an XML document.
- Generating an XML document from a PHP array.
- Generating an XML document from information in a database table.

11.4 Exercises

1. Using the XML document in the `$menu` variable defined in Example 11-5, print an HTML `<ul>` list in which each list element is the `<title>` of one `<item>` in the XML document, and that `<title>` is hyperlinked to the URL listed in the `<link>` element of the item. For example, if one of the items were:

```
<item>
 <title>Steamed Rock Cod</title>
 <link>http://menu.example.com/dishes.php?dish=cod</link>
 <description>Enjoy a cod, bursting with flavor.</description>
</item>
```

   Then the corresponding list element that your code prints would be:

```
<li><a href="http://menu.example.com/dishes.php?dish=cod">Steamed Rock
Cod</a>
     </li>
```

6. Write a program that prints a form asking for a user to input an RSS item title, link, and description. Make sure the user enters something for each field. Use the submitted form data to print an XML document consisting of a one-item RSS feed. Define the `<channel>` part of the feed in your program (you don't have to gather form input for it). Make sure to use `header( )` and `htmlentities( )` to produce a valid XML response.

7. Modify your answer to Exercise 7.2 so that the output of the program is an XML document. Structure your output like Example 11-18? put the information about each dish inside `<dish></dish>` tags, and put all the `<dish></dish>` tags inside `<dishes></dishes>` tags.

8. Write a program that prints a form asking for a user to input a search term. Retrieve an RSS news feed (such as one listed at http://news.yahoo.com/rss/) and display a list of links to items in the news feed that have the search term in the item title. Format each list element the same way as Exercise 11.1. To find matching news titles, you can use a regular expression or a function such as `stristr( )`.

Chapter 12. Debugging

Programs rarely work correctly the first time. This chapter shows you some techniques for finding and fixing the problems in your programs. When you're just learning PHP, your programs are probably simpler than the programs that PHP wizards write. The errors you get, however, generally aren't much simpler, and you have to use the same tools and techniques to find and fix those errors.

PREV < Day Day Up > NEXT

12.1 Controlling Where Errors Appear

Many things can go wrong in your program that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log.

A useful way to configure an error message display is to have the errors displayed on screen when you're developing a PHP program, and then sent to the error log once you're done development and people are actually using the program. While you're working on a program, it's helpful to see immediately that there was a parse error on a particular line, for example. But once the program is (supposedly) working so that your coworkers or customers can use it, such an error message would be confusing to them.

To make error messages display in the browser, set the `display_errors` configuration directive to `On`. To send errors to the web server error log, set `log_errors` to `On`. You can set them both to `On` if you want error messages in both places.

An error message that the PHP interpreter generates falls into one of five different categories:

Parse error

> A problem with the syntax of your program, such as leaving a semicolon off of the end of a statement. The interpreter stops running your program when it encounters a parse error.

Fatal error

> A severe problem with the content of your program, such as calling a function that hasn't been defined. The interpreter stops running your program when it encounters a fatal error.

Warning

> An advisory from the interpreter that something is fishy in your program, but the interpreter can keep going. Using the wrong number of arguments when you call a function causes a warning.

Notice

> A tip from the PHP interpreter playing the role of Miss Manners. For example, printing a variable without first initializing it to some value generates a notice.

Strict notices

> An admonishment from the PHP interpreter about your coding style. Most of these have to do with esoteric features that changed between PHP 4 and PHP 5, so you're not likely to run into them too much.

You don't have to be notified about all the different error categories. The `error_reporting` configuration directive controls which kinds of errors the PHP interpreter reports. The default value for `error_reporting` is `E_ALL & ~E_NOTICE & ~E_STRICT`, which tells the interpreter to report all errors except notices and strict notices. Appendix A explains what the `&` and `~` mean in configuration directive values.

PHP defines some constants you can use to set the value of `error_reporting` such that only errors of certain types get reported: `E_ALL` (for all errors except strict notices), `E_PARSE` (parse errors), `E_ERROR` (fatal errors), `E_WARNING` (warnings), `E_NOTICE` (notices), and `E_STRICT` (strict notices).

Because strict notices are rare (and new to PHP 5), they are not included in `E_ALL`. To tell the PHP interpreter that you want to hear about everything that could possibly be an error, set `error_reporting` to `E_ALL | E_STRICT`.

12.2 Fixing Parse Errors

The PHP interpreter is really picky but not very chatty. If you leave out a necessary semicolon, or start a string with a single quote but end it with a double quote, the interpreter doesn't run your program. It throws up its (virtual) hands, complains about a "parse error," and leaves you stuck in the debugging wilderness.

This can be one of the most frustrating things about programming when you're getting started. Everything has to be phrased and punctuated *just so* in order for the PHP interpreter to accept it. One thing that helps this process along is writing your programs in an editor that is PHP-aware. This is a program that, when you tell it you are editing a PHP program, turns on some special features that make programming easier.

One of these special features is *syntax highlighting*. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as `if` and `while` are blue, comments are grey, and variables are black. Syntax highlighting makes it easier to detect things such as a string that's missing its closing quote: the pink text continues past the line that the string is on, all the way to the end of the file (or the next quote that appears later in the program).

Another feature is *quote and bracket matching*, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as `}`, the editor highlights the opening `{` that it matches. Different editors do this in different ways, but typical methods are to flash the cursor at the location of the opening {, or to bold the `{ }` pair for a short time. This behavior is helpful for pairs of punctuation that go together: single and double quotes that delimit strings, parentheses, square brackets, and curly braces.

These editors also show the line numbers of your program files. When you get an error message from the PHP interpreter complaining about a parse error in line 35 in your program, you can focus on the right place to look for your error.

Table 12-1 lists seven PHP-aware editors. Some of them go beyond the basics of syntax highlighting and bracket matching and provide more advanced features to help your coding. These features are listed in the "Comments" column of the table.

Table 12-1. PHP-aware text editors

Name	Platform(s)	URL	Cost	Comments
BBEdit	OS X	http://www.barebones.com/products/bbedit/index.shtml	$179	
Emacs and XEmacs	All	http://www.gnu.org/software/emacs/, http://www.xemacs.org	Free	
				Provides context-sensitive PHP

12.3 Inspecting Program Data

Once you clear the parse error hurdle, you still may have some work to do before you reach the finish line. A program can be syntactically correct but logically flawed. Just as the sentence "The tugboat chewed apoplectically with six subtle buffaloes" is grammatically correct but meaningless nonsense, you can write a program that the PHP interpreter doesn't find any problems with but doesn't do what you expect.

If your program is acting funny, add some checkpoints that display the values of variables. That way, you can see where the program's behavior diverges from your expectations. Example 12-3 shows a program that incorrectly attempts to calculate the total cost of a few items.

Example 12-3. A broken program without debugging output

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // 8% tax

foreach ($prices as $price) {
    $total_price = $price * $tax_rate;
}

printf('Total price (with tax): $%.2f', $total_price);
```

Example 12-3 doesn't do the right thing. It prints:

```
Total price (with tax): $13.50
```

The total price of the items should be at least $20. What's wrong with Example 12-3? One way you can try to find out is to insert a line in the `foreach( )` loop that prints the value of `$total_price` before and after it changes. That should provide some insight into why the math is wrong. Example 12-4 annotates Example 12-3 with some diagnostic `print` statements.

Example 12-4. A broken program with debugging output

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // 8% tax

foreach ($prices as $price) {
    print "[before: $total_price]";
    $total_price = $price * $tax_rate;
    print "[after: $total_price]";
}

printf('Total price (with tax): $%.2f', $total_price);
```

Example 12-4 prints:

```
[before: 0][after: 6.426][before: 6.426][after: 3.24][before: 3.24][after:
13.5]Total
price (with tax): $13.50
```

From analyzing the debugging output from Example 12-4, you can see that `$total_price` isn't increasing on each trip through the `foreach( )` loop. Scrutinizing the code further leads you to the conclusion that the line:

```
$total_price = $price * tax_rate;
```

should be:

```
$total_price += $price * tax_rate;
```

12.4 Fixing Database Errors

When your program involves talking to a database, you have to deal with an additional universe of errors. Just as the PHP interpreter expects your programs to adhere to a particular grammar, the database program expects your SQL statements to adhere to the grammar of SQL.

The `setErrorHandling( )` function introduced in Section 7.4 has an additional mode of operation that gives you increased control over how database errors are handled in your PHP programs. Instead of having a terse error message printed or your program exit when a database error happens, you can have a custom function called. That function can do whatever you want, such as print a more detailed error message or write to the web server error log.

To enable this mode, call `setErrorHandling( )` with the `PEAR_ERROR_CALLBACK` constant and the name of your error-handling function. Example 12-8 says that when there is a database error, the `database_error( )` function should be called.

Example 12-8. Setting up a custom database error handling function

```
$db->setErrorHandling(PEAR_ERROR_CALLBACK,'database_error');
```

You also have to write the custom error-handling function whose name is passed to `setErrorHandling( )`. This function must accept one argument. When DB invokes the function, it passes an object to the function that contains the error information. You can use the `getDebugInfo( )` method of that object to get more detailed error information. Example 12-9 is a sample custom error-handling function.

Example 12-9. A custom database error handling function

```
function database_error($error_object) {
    print "We're sorry, but there is a temporary problem with the database.";
    $detailed_error = $error_object->getDebugInfo( );
    error_log($detailed_error);
}
```

The `database_error( )` function defined in Example 12-9 prints a generic message when a database error happens. It sends more detailed information about the error to the web server error log. Because this detailed information includes the full text of the database queries that caused errors, you shouldn't show it to your web site visitors. The messages that `database_error( )` sends to the error log look like this:

```
SELECT dish_name, price, has_spiciness FROM dishes WHERE price >= '5.00' AND
price <=
'25.00' AND is_spicy = 0 [nativecode=1054 ** Unknown column 'has_spiciness'
in 'field
list']
```

Since the `dishes` table doesn't have a column called `has_spiciness`, a query that tries to use such a column fails.

12.5 Chapter Summary

Chapter 12 covers:

- Configuring error display for a web browser, a web server error log, or both.
- Configuring the PHP interpreter's error-reporting level.
- Getting the benefits of a PHP-aware text editor.
- Deciphering parse error messages.
- Finding and fixing parse errors.
- Printing debugging information with `print`, `var_dump( )` and `error_log( )`.
- Sending `var_dump( )` output to the error log with output buffering functions.
- Writing a custom database error-handling function.

PREV < Day Day Up > NEXT

12.6 Exercises

1. This program has a syntax error in it:

```
<?php
$name = 'Umberto';
function say_hello( ) {
    print 'Hello, ';
    print global $name;
}
say_hello( );
?>
```

   Without running the program through the PHP interpreter, try to figure out what the parse error looks like that gets printed when the interpreter tries to run the program. What change must you make to the program to get it to run properly and print `Hello, Umberto`?

2. Modify the `validate_form( )` function in your answer to Exercise 6.3 so that it prints in the web server error log the names and values of all of the submitted form parameters.

3. Modify your answer to Exercise 7.4 to use a custom database error-handling function that prints out different messages in the web browser and in the web server error log. The error-handling function should make the program exit after it prints the error messages.

4. This program is supposed to print out an alphabetical list of all the customers in the table from Exercise 7.4. Find and fix the errors in it.

```
<?php
require 'DB.php';
require 'formhelpers.php';
// Connect to the database
$db = DB:connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die ("Can't connect: " . $db->getMessage( )); }
// Set up automatic error handling
$db->setErrorHandling(PEAR_ERROR_DIE);
// Set up fetch mode: rows as objects
$db->setFetchMode(DB_FETCHMODE_OBJECT);
// get the array of dish names from the database
$dish_names = array( );
$res = $db->query('SELECT dish_id,dish_name FROM dishes');
while ($row = $res->fetchRow( )) {
    $dish_names[ $row['dish_id']] ] = $row['dish_name'];
}
$customers = $db->getAll('SELECT ** FROM customers ORDER BY phone
DESC');
if ($customers->num_rows( ) = 0) {
    print "No customers.";
} else {
    print '<table>';
   print '<tr><th>ID</th><th>Name</th><th>Phone</th><th>Favorite
Dish</th></tr>";
    while ($customer = $customers->fetchRow( )) {
        printf('<tr><td>%d</td><td>%s</td><td>%f</td><td>%s</td></tr>',
               $customer['customer_id'],
               htmlentities($customer['cutsomer_name']),
               $customer['phone'],
               $customer['favorite_dish_id']);
    }
    print '</table>';
?>
```

Chapter 13. What Else Can You Do with PHP?

This book covers the fundamental PHP topics that you need for everyday dynamic web site development, such as handling forms, working with a database, and remembering users with sessions. Beyond that core, though, PHP can do much more. Here are a few paragraphs, an example or two, and links to more info about many other capabilities of PHP.

PREV < Day Day Up > NEXT

13.1 Graphics

Your PHP programs can produce more than just HTML web pages. With the GD extension, they can also dynamically generate graphics? for example, you can create custom buttons. Example 13-1 draws a rudimentary button whose text comes from the `button` URL variable.

Example 13-1. Drawing a button image

```
<?php

// GD's built-in fonts are numbered from 1 - 5
$font = 3;

// Calculate the appropriate image size
$image_height = intval(imageFontHeight($font) * 2);
$image_width = intval(strlen($_GET['button']) * imageFontWidth($font) * 1.3);

// Create the image
$image = imageCreate($image_width, $image_height);

// Create the colors to use in the image
// gray background
$back_color = imageColorAllocate($image, 216, 216, 216);
// blue text
$text_color = imageColorAllocate($image, 0,   0,   255);
// black border
$rect_color = imageColorAllocate($image, 0,   0,   0);

// Figure out where to draw the text
// (Centered horizontally and vertically
$x = ($image_width - (imageFontWidth($font) * strlen($_GET['button']))) / 2;
$y = ($image_height - imageFontHeight($font)) / 2;

// Draw the text
imageString($image, $font, $x, $y, $_GET['button'], $text_color);
// Draw a black border
imageRectangle($image, 0, 0, imageSX($image) - 1, imageSY($image) - 1,
$rect_color);

// Send the image to the browser
header('Content-Type: image/png');
imagePNG($image);
imageDestroy($image);
?>
```

If Example 13-1 is saved as *button.php* in the document root directory of your web server, then you can call it like this:

```
<img src="/button.php?button=Click+Here">
```

It then outputs a button that looks like Figure 13-1.

Figure 13-1. Dynamic button

Read more about these functions in Chapter 9 of *Programming PHP* by Rasmus Lerdorf and Kevin Tatroe (O'Reilly), in Chapter 15 of *PHP Cookbook* by David Sklar and Adam Trachtenberg (O'Reilly), and in the Image section of the PHP Manual (http://www.php.net/image). Jeff Knight's presentation to NYPHP about PHP's image functions is also a good source of information. It's available at http://www.nyphp.org/content/presentations/GDintro.

13.2 PDF

Another kind of non-HTML document that your PHP programs can produce is a PDF file, as shown in Example 13-2. This is handy for making an invoice that incorporates information from your database or providing printable versions of pages that meet exacting layout standards.

Example 13-2. Generating a PDF document

```
// These values are in points (1/72nd of an inch)
$fontsize = 72;     // 1 inch high letters
$page_height = 612; // 8.5 inch high page
$page_width = 792;  // 11 inch wide page

// Use a default message if none is supplied
if (strlen(trim($_GET['message']))) {
    $message = trim($_GET['message']);
} else {
    $message = 'Generate a PDF!';
}

// Create a new PDF document in memory
$pdf = pdf_new( );
pdf_open_file($pdf, '');

// Add a 11"x8.5" page to the document
pdf_begin_page($pdf, $page_width, $page_height);

// Select the Helvetica font at 72 points
$font = pdf_findfont($pdf, "Helvetica", "winansi", 0);
pdf_setfont($pdf, $font, $fontsize);

// Display the message centered on the page
pdf_show_boxed($pdf, $message, 0, ($page_height-$fontsize)/2,
               $page_width, $fontsize, 'center');

// End the page and the document
pdf_end_page($pdf);
pdf_close($pdf);

// Get the contents of the document and delete it from memory
$pdf_doc = pdf_get_buffer($pdf);
pdf_delete($pdf);

// Send appropriate headers and the document contents
header('Content-Type: application/pdf');
header('Content-Length: ' . strlen($pdf_doc));
print $pdf_doc;
```

Example 13-2 uses the functions in the PDF extension. This extension depends on the PDFLib library that is available at http://www.pdflibrary.com. The CLibPDF extension also generates PDF files, but depends on the ClibPDF library that is available at http://www.fastio.com. Both PDFLib and CLibPDF require that you buy a license to use them for commercial purposes.

See Chapter 10 of O'Reilly's *Programming PHP* for detailed information about creating PDF documents, and read http://www.php.net/manual/faq.using.php#faq.using.freepdf for some free PDF creation options.

13.3 Shockwave/Flash

You can also create full-featured SWF-format Flash movies with the Ming extension. Example 13-3 produces a movie with a blue circle in it that you can drag around.

Example 13-3. Generating a Flash movie

```
// Use SWF Version 6 to enable Actionscript
ming_UseSwfVersion(6);

// Create a new movie and set some parameters
$movie = new SWFMovie( );
$movie->setRate(20.000000);
$movie->setDimension(550, 400);
$movie->setBackground(0xcc,0xcc,0xcc);

// Create the circle
$circle = new SWFShape( );
$circle->setRightFill(33,66,99);
$circle->drawCircle(40);
$sprite= new SWFSprite( );
$sprite->add($circle);
$sprite->nextFrame( );

// Add the circle to the movie
$displayitem = $movie->add($sprite);
$displayitem->setName('circle');
$displayitem->moveTo(100,100);

// Add the Actionscript that implements the dragging
$movie->add(new SWFAction("
 circle.onPress=function( ){ this.startDrag('');};
 circle.onRelease= circle.onReleaseOutside=function( ){ stopDrag( );};
"));

// Display the movie
header("Content-type: application/x-shockwave-flash");
$movie->output(1);
```

Save Example 13-3 as *ming.php* and then reference it from another page as in Example 13-4.

Example 13-4. Including the Flash movie in a web page

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.
cab#version=6,0,0,0"
 WIDTH="300" HEIGHT="300">
 <PARAM NAME=movie VALUE="ming.php">
 <PARAM NAME=bgcolor VALUE="#ffffff">
 <EMBED src="ming.php" bgcolor="#ffffff" WIDTH="300" HEIGHT="300"
 TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/
getflashplayer"></EMBED>
</OBJECT>
```

Read about the Ming functions in the PHP Manual at http://www.php.net/ming. The Ming extension depends on the external Ming library, which you can download from http://ming.sourceforge.net. The site at http://ming.sourceforge.net also contains lots of documentation and examples of how to use Ming from PHP. (Example 13-3 is adapted from one of the examples on that site.)

13.4 Browser-Specific Code

The `get_browser( )` function gives you information about the characteristics and capabilities of a user's browser. It makes it easy to dynamically determine what kind of page to output based on what a browser can do, what kind of browser it is, or on what operating system it's running. Example 13-5 prints a message that depends on the operating system of the user's browser.

Example 13-5. Using get_browser( )

```
$browser = get_browser( );

if ($browser->platform =  = 'WinXP') {
    print 'You are using Windows XP.';
} elseif ($browser->platform =  = 'MacOSX') {
    print 'You are using Mac OS X.';
} else {
    print 'You are using a different operating system.';
}
```

The `get_browser( )` function uses the `$_SERVER['HTTP_USER_AGENT']` variable described in Table 6-1. Remember, that variable can be faked, but it is still useful in producing customized pages for the majority of your users. For `get_browser( )` to work, you need to download a separate browser capabilities file and set the `browscap` configuration directive. The PHP Manual page about `get_browser( )` (http://www.php.net/get_browser) provides up-to-date information on where to get a browser capabilities file.

13.5 Sending and Receiving Mail

The `mail( )` function (which you saw briefly in Example 6-30) sends an email message. To use `mail( )`, pass it a destination address, a message subject, and a message body. Example 13-6 sends a message with `mail( )`.

Example 13-6. Sending a message with mail( )

```
$mail_body=<<<_TXT_
Your order is:
* 2 Fried Bean Curd
* 1 Eggplant with Chili Sauce
* 3 Pineapple with Yu Fungus
_TXT_;
mail('hungry@example.com','Your Order',$mail_body);
```

To handle more complicated messages, such as an HTML message or a message with an attachment, use the PEAR Mail and Mail_Mime modules. Example 13-7 shows how to use Mail_Mime to send a multipart message that has a text part and an HTML part.

Example 13-7. Sending a message with text and HTML bodies

```
require 'Mail.php';
require 'Mail/mime.php';

$headers = array('From' => 'orders@example.com',
                 'Subject' => 'Your Order');

$text_body = <<<_TXT_
Your order is:
* 2 Fried Bean Curd
* 1 Eggplant with Chili Sauce
* 3 Pineapple with Yu Fungus
_TXT_;

$html_body = <<<_HTML_
<p>Your order is:</p>
<ul>
<li><b>2</b> Fried Bean Curd</li>
<li><b>1</b> Eggplant with Chili Sauce</li>
<li><b>3</b> Pineapple with Yu Fungus</li>
</ul>
_HTML_;

$mime = new Mail_mime( );
$mime->setTXTBody($text_body);
$mime->setHTMLBody($html_body);

$msg_body = $mime->get( );
$msg_headers = $mime->headers($headers);

$mailer = Mail::factory('mail');

$mailer->send('hungry@example.com', $msg_headers, $msg_body);
```

When `hungry@example.com` reads the message sent in Example 13-7, his mail-reading program displays the HTML body or the text body, depending on its capabilities and how it is configured.

Read more about PEAR Mail and Mail_Mime in *PHP Cookbook* (O'Reilly), Recipes 17.1 and 17.2; in Chapter 9 of *Essential PHP Tools* by David Sklar (APress); and at http://pear.php.net/manual/en/package.mail.mail-mime.php.

13.6 Uploading Files in Forms

The `<input type="file">` form element lets a user upload the entire contents of a file to your server. When a form that includes a file element is submitted, the PHP interpreter provides access to the uploaded file through the `$_FILES` auto-global array. Example 13-8 shows a form-processing program whose `validate_form( )` and `process_form( )` functions use `$_FILES`.

Example 13-8. A file upload form

```
if ($_POST['_stage']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}

function show_form($errors = '') {

    if ($errors) {
        print 'You need to correct the following errors: <ul><li>';
        print implode('</li><li>',$errors);
        print '</li></ul>';
    }

    print<<<_HTML_
<form enctype="multipart/form-data" method="POST"
      action="$_SERVER[PHP_SELF]">

File to Upload: <input name="my_file" type="file"/>

<input type="hidden" name="MAX_FILE_SIZE" value="131072"/>
<input type="hidden" name="_stage" value="1">
<input type="submit" value="Upload"/>
</form>
_HTML_;
}

function validate_form( ) {
    $errors = array( );

    if (($_FILES['my_file']['error'] =  = UPLOAD_ERR_INI_SIZE)||
        ($_FILES['my_file']['error'] =  = UPLOAD_ERR_FORM_SIZE)) {
        $errors[  ] = 'Uploaded file is too big.';
    } elseif ($_FILES['my_file']['error'] =  = UPLOAD_ERR_PARTIAL) {
        $errors[  ] = 'File upload was interrupted.';
    } elseif ($_FILES['my_file']['error'] =  = UPLOAD_ERR_NO_FILE) {
        $errors[  ] = 'No file uploaded.';
    }

    return $errors;
}

function process_form( ) {
    print "You uploaded a file called {$_FILES['my_file']['name']} ";
    print "of type {$_FILES['my_file']['type']} that is ";
    print "{$_FILES['my_file']['size']} bytes long.";

    $safe_filename = str_replace('/', '', $_FILES['my_file']['name']);
```

13.7 The HTML_QuickForm Form-Handling Framework

Chapter 6 provides all the building blocks of robust form handling. A PEAR module, HTML_QuickForm, takes things a step further. It makes it easy to use common validation rules and simplifies default processing and encoding user input with `htmlentities( )`. With HTML_QuickForm, the entire form is an object. You call methods on that object to add elements and validation rules to the form. Example 13-9 uses HTML_QuickForm to build the form in Example 6-30.

Example 13-9. Building a form with QuickForm

```
<?php
// Load the QuickForm library
require 'HTML/QuickForm.php';
// Create the form object
$form = new HTML_QuickForm( );

// Define the same arrays of valid sweets and main dishes
$sweets = array('puff' => 'Sesame Seed Puff',
                'square' => 'Coconut Milk Gelatin Square',
                'cake' => 'Brown Sugar Cake',
                'ricemeat' => 'Sweet Rice and Meat');

$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' => 'Abalone with Marrow and Duck Feet');

// Set the default values for form elements
$form->setDefaults(array('delivery' => 'yes',
                         'size'     => 'medium'));

// Add each element to the form
$form->addElement('text','name','Your Name: ');
$form->addElement('radio','size','Size:','Small', 'small');
$form->addElement('radio','size','',     'Medium', 'medium');
$form->addElement('radio','size','',     'Large', 'large');

$form->addElement('select','sweet','Pick one sweet item:', $sweets);
$form->addElement('select','main_dish','Pick two main dishes:',
                  $main_dishes, 'multiple="multiple"');

$form->addElement('radio','delivery','Do you want your order delivered?',
                  'Yes','yes');

$form->addElement('textarea','comments','Enter any special instructions. <br/>
                  If you want your order delivered, put your address here:');

$form->addElement('submit','save','Order');

// Create two custom validation rules (implemented by the functions
// add the end of the script)
$form->registerRule('check_array','function','check_array');
$form->registerRule('check_array_size','function','check_array_size');

// The name field is required
$form->addRule('name','Please enter your name.','required');
// The size field is required and its value must be
// one of "small", "medium", or "large"
$form->addRule('size','Please select a size.','required');
$form->addRule('size','Please select a size.','check_array',
               array('small' => 1, 'medium' => 1, 'large' => 1));
```

13.8 Classes and Objects

PHP 5 provides comprehensive and robust support for object-oriented programming. If you've never heard of object-oriented programming, then you don't need to use any of these fancy features. But if you're coming to PHP from a language such as Java, you can structure your code in familiar ways. You can create interfaces; abstract classes; public, private, and protected properties and methods; constructors and destructors; overloaded property accessors and method dispatchers; and plenty of other OO goodies.

Chapter 2 of *Upgrading to PHP 5* by Adam Trachtenberg (O'Reilly), lays out the many object-related changes in PHP 5. The PHP Manual covers classes and objects at http://www.php.net/manual/language.oop.php.

13.8.1 Object Basics

An *object*, in the programming world, is a structure that combines data about a thing (such as the ingredients in an entree) with actions on that thing (such as preparing the entree). Using objects in a program provides an organizational structure for grouping related variables and functions together.

Some words to know when working with objects are defined in the following list:

Class

A template or recipe that describes the variables and functions for a kind of object. For example, an `Entree` class would contain variables that hold its name and ingredients. The functions in an `Entree` class would be for things such as cooking the entree, serving it, and determining whether a particular ingredient is in it.

Method

A function defined in a class is called a method.

Property

A variable defined in a class is called a property.

Instance

An individual usage of a class. If you are serving three entrees for dinner in your program, you would create three instances of the `Entree` class. While each of these instances is based on the same class, they differ internally with different properties. The methods in each instance contain the same instructions, but probably produce different results because they each rely on the particular property values in each instance. Creating a new instance of a class is called "instantiating an object."

Constructor

A special method that is automatically run when an object is instantiated. Usually, constructors set up object properties and do other housekeeping that makes the object ready for use.

Static method

A special kind of method that can be called without instantiating a class. Static methods don't depend on the property values of a particular instance. PEAR DB uses a static method to create a database connection.

13.8.2 Creating a New Object

PEAR DB uses a static method to create a new object instance for you to use:

```
$db = DB::connect($dsn);
```

This calls the `connect( )` method defined in the `DB` class. The `connect( )` method is a

13.9 Advanced XML Processing

SimpleXML is just the tip of PHP 5's new XML processing capabilities. The DOM functions give you exacting control over all aspects of an XML document, and you can also do XSL transformations, XPath queries, and XInclude processing, as well as execute an extravagant, exhaustive exaltation of other exciting and exotic XML exercises.

Example 13-10 shows an RSS feed-handling class based on the built-in `DomDocument` class. The `addItem( )` method of the `RSS` class is used to add a new item to the feed.

Example 13-10. Extending DomDocument to handle an RSS feed

```
class RSS extends DomDocument {
    function _ _construct($title, $link, $description) {
        // Set this document up as XML 1.0 with a root
        // <rss> element that has a version="0.91" attribute
        parent::_ _construct('1.0');
        $rss = $this->createElement('rss');
        $rss->setAttribute('version', '0.91');
        $this->appendChild($rss);

        // Create a <channel> element with <title>, <link>,
        // and <description> sub-elements
        $channel = $this->createElement('channel');
        $channel->appendChild($this->makeTextNode('title', $title));
        $channel->appendChild($this->makeTextNode('link', $link));
        $channel->appendChild($this->makeTextNode('description',
                                                  $description));

        // Add <channel> underneath <rss>
        $rss->appendChild($channel);

        // Set up output to print with linebreaks and spacing
        $this->formatOutput = true;
    }

    // This function adds an <item> to the <channel>
    function addItem($title, $link, $description) {
        // Create an <item> element with <title>, <link>
        // and <description> sub-elements
        $item = $this->createElement('item');
        $item->appendChild($this->makeTextNode('title', $title));
        $item->appendChild($this->makeTextNode('link', $link));
        $item->appendChild($this->makeTextNode('description',
                                               $description));

        // Add the <item> to the <channel>
        $channel = $this->getElementsByTagName('channel')->item(0);
        $channel->appendChild($item);
    }

    // A helper function to make elements that consist entirely
    // of text (no sub-elements)
    private function makeTextNode($name, $text) {
        $element = $this->createElement($name);
        $element->appendChild($this->createTextNode($text));
        return $element;
    }
}

// Create a new RSS feed with the specified title, link and description
// for the channel.
$rss = new RSS("What's For Dinner", 'http://menu.example.com/',
               'These are your choices of what to eat tonight.');
// Add three items
```

13.10 SQLite

The SQLite embedded database engine comes bundled with PHP 5. An SQLite database is a single file. Inside that file are all the tables in a database. You don't need a separate database program running on your server to access an SQLite database? when your PHP program connects to the database, it opens the file, reads from it, and writes to it. For heavily trafficked sites, SQLite isn't as fast as a regular database program such as MySQL, but it is packed with features and is capable for small projects. Example 13-13 shows the answer to Exercise 7.1 using SQLite.

Example 13-13. Using the SQLite database

```
require 'DB.php';

$db = DB::connect('sqlite://:@localhost/restaurant.db');
if (DB::isError($db)) { die("Can't connect: " . $db->getMessage( )); }

$db->setErrorHandling(PEAR_ERROR_DIE);
$db->setFetchMode(DB_FETCHMODE_ASSOC);

$dishes = $db->getAll('SELECT dish_name,price FROM dishes ORDER BY price');

if (count($dishes) > 0) {
    print '<ul>';
    foreach ($dishes as $dish) {
        print "<li> $dish[dish_name] ($dish[price])</li>";
    }
    print '</ul>';
} else {
    print 'No dishes available.';
}
```

The only thing different about Example 13-13 and the answer (Section C.6.1) to Exercise 7.1 (Section 7.14) is the DSN supplied to `DB::connect( )`. The DSN for SQLite doesn't have a username or password, and instead of a database name, the last part of the DSN is the filename of the SQLite database file.

Chapter 4 of O'Reilly's *Upgrading to PHP 5* discusses SQLite. You can also read about SQLite in the PHP Manual (http://www.php.net/sqlite).

13.11 Running Shell Commands

While you can do almost anything in PHP, you can't do everything. If you need to run an external program from inside a PHP script, you have a few options. These are described in the "Program Execution" section of the PHP Manual (http://www.php.net/exec). Example 13-14 demonstrates the `shell_exec( )` command, which runs a program and returns its output. In Example 13-14, `shell_exec( )` runs the `df` command, which (on Unix) produces information about disk usage.

Example 13-14. Running a program with shell_exec( )

```
// Run "df" and divide up its output into individual lines
$df_output = shell_exec('/bin/df -h');
$df_lines = explode("\n", $df_output);

// Loop through each line. Skip the first line, which
// is just a header
for ($i = 1, $lines = count($df_lines); $i < $lines; $i++) {
    if (trim($df_lines[$i])) {
        // Divide up the line into fields
        $fields = preg_split('/\s+/', $df_lines[$i]);
        // Print info about each filesystem
        print "Filesystem $fields[5] is $fields[4] full.\n";
    }
}
```

Example 13-14 prints something like this:

```
Filesystem / is 63% full.
Filesystem /boot is 7% full.
Filesystem /opt is 93% full.
Filesystem /dev/shm is 0% full.
```

Just like when using external input in a SQL query or filename, you need to be careful when using external input as part of an external command line. Make your programs more secure by using `escapeshellargs( )` to escape shell metacharacters in command-line arguments.

Read more about running external commands in Section 12.7 of *Programming PHP* (O'Reilly) and in *PHP Cookbook* (O'Reilly), Recipes 18.20, 18.21, 18.22 and 18.23.

13.12 Advanced Math

On most systems, the PHP interpreter can handle integers between -2147483648 and 2147483647 (that's 2 billion), and floating-point numbers between -10^308 and 10^308. If you're writing scientific or other math-intensive applications, such as figuring out each citizen's portion of the U.S. National Debt, that might not be good enough. The BCMath and GMP extensions provide more advanced mathematical capabilities. The GMP extension is more capable, but not available on Windows. Example 13-15 uses the BCMath extension to compute the hypotenuse of a really big right triangle.

Example 13-15. Doing math with the BCMath extension

```
// Figure out hypotenuse of a giant right triangle
// The sides are 3.5e406 and 2.8e406

$a = bcmul(3.5, bcpow(10, 406));
$b = bcmul(2.8, bcpow(10, 406));

$a_squared = bcpow($a, 2);
$b_squared = bcpow($b, 2);

$hypotenuse = bcsqrt(bcadd($a_squared, $b_squared));

print $hypotenuse;
```

The number that Example 13-15 prints is 407 digits long.

Example 13-16 shows the same calculation with the functions in the GMP extension.

Example 13-16. Doing math with the GMP extension

```
$a = gmp_mul(35, gmp_pow(10,405));
$b = gmp_mul(28, gmp_pow(10,405));

$a_squared = gmp_pow($a, 2);
$b_squared = gmp_pow($b, 2);

$hypotenuse = gmp_sqrt(gmp_add($a_squared, $b_squared));

print gmp_strval($hypotenuse);
```

Read about BCMath and GMP in O'Reilly's *PHP Cookbook*, Recipe 2.13; and in the PHP Manual (http://www.php.net/bc and http://www.php.net/gmp).

13.13 Encryption

With the mcrypt extension, you can encrypt and decrypt data using a variety of popular algorithms such as Blowfish, Triple DES, and Twofish. Example 13-17 encrypts and decrypts a string with Blowfish.

Example 13-17. Encrypting and decrypting with mcrypt

```
// The string to encrypt
$data = 'Account number: 213-1158238-23; PIN: 2837';
// The secret key to encrypt it with
$key  = "Perhaps Looking-glass milk isn't good to drink";

// Select an algorithm and encryption mode
$algorithm = MCRYPT_BLOWFISH;
$mode = MCRYPT_MODE_CBC;
// Create an initialization vector
$iv = mcrypt_create_iv(mcrypt_get_iv_size($algorithm,$mode),
                       MCRYPT_DEV_URANDOM);

// Encrypt the data
$encrypted_data = mcrypt_encrypt($algorithm, $key, $data, $mode, $iv);

// Decrypt the data
$decrypted_data = mcrypt_decrypt($algorithm, $key, $encrypted_data, $mode,
$iv);

print "The decoded data is $decrypted_data";
```

Example 13-17 prints:

```
The decoded data is Account number: 213-1158238-23; PIN: 2837
```

Read about mcrypt in *PHP Cookbook*, Recipes 14.7, 14.8, and 14.9, and in the PHP Manual (http://www.php.net/mcrypt). Just as a fancy lock on your front door doesn't do much if your house is made of clear plastic sheeting, the most robust encryption algorithm is just one part of a comprehensively secure program. To learn more about computer security and encryption, read *Practical Unix & Internet Security* by Simson Garfinkel, Alan Schwartz, and Gene Spafford (O'Reilly) and *Applied Cryptography* by Bruce Schneier (John Wiley and Sons).

13.14 Talking to Other Languages

With various extensions, the PHP interpreter can run programs written in other languages such as Java and Perl. On Windows, the PHP interpreter can access COM objects.

The Perl extension is for PHP 5 only. Example 13-18 demonstrates a very simple program that uses the Perl extension to print a message. Typically, you'd use the Perl extension to access some existing Perl libraries that you have.

Example 13-18. Using Perl from PHP

```
$perl = new Perl( );

$perl->eval('print "This is Perl!";');
```

Example 13-18 prints:

```
This is Perl!
```

Example 13-19 shows a simple Java example.

Example 13-19. Using Java from PHP

```
$formatter = new Java('java.text.SimpleDateFormat',
                      "EEEE, MMMM dd, yyyy 'at' h:mm:ss a zzzz");

print $formatter->format(new Java('java.util.Date'));
```

In the afternoon of October 20, 2004, Example 13-19 prints:

```
Wednesday, October 20, 2004 at 1:30:00 PM Eastern Daylight Time
```

Read about the Perl extension at http://www.zend.com/php5/articles/php5-perl.php, and the Java and COM extensions in the PHP Manual (http://www.php.net/java and http://www.php.net/com).

13.15 IMAP, POP3, and NNTP

You can write a full-featured mail or news client in PHP. (In fact, some people already have? check out http://www.horde.org/imp/ and http://www.squirrelmail.org/). The imap extension gives your PHP programs the ability to talk with IMAP, POP3, and NNTP servers. Example 13-20 uses some of the imap extension functions to connect to the `news.php.net` news server and retrieve information about 10 most recent messages from the `php.announce` newsgroup.

Example 13-20. Connecting to an NNTP server

```
$server = '{news.php.net/nntp:119}';
$group = 'php.announce';
$nntp = imap_open("$server$group", '', '', OP_ANONYMOUS);

$last_msg_id = imap_num_msg($nntp);

$msg_id = $last_msg_id - 9;

print '<table>';
print "<tr><td>Subject</td><td>From</td><td>Date</td></tr>\n";

while ($msg_id <= $last_msg_id) {
    $header = imap_header($nntp, $msg_id);

    if (! $header->Size) { print "no size!"; }

    $email = $header->from[0]->mailbox . '@' .
        $header->from[0]->host;
    if ($header->from[0]->personal) {
        $email .= ' ('.$header->from[0]->personal.')';
    }

    $date = date('m/d/Y h:i A', $header->udate);

    print "<tr><td>$header->subject</td><td>$email</td>" .
        "<td>$date</td></tr>\n";
$msg_id++;
}
print '</table>';
```

Example 13-20 prints:

```
<table><tr><td>Subject</td><td>From</td><td>Date</td></tr>
<tr><td>PHP Security Advisory: CGI vulnerability in PHP version 4.3.0</td>
<td>sniper@php.net (Jani Taskinen)</td><td>02/17/2003 01:01 PM</td></tr>
<tr><td>PHP 4.3.2 released</td><td>sniper@php.net (Jani Taskinen)</td>
<td>05/29/2003 08:05 AM</td></tr>
<tr><td>PHP 5.0.0 Beta 1</td><td>sterling@bumblebury.com (Sterling
Hughes)</td>
<td>06/29/2003 02:19 PM</td></tr>
<tr><td>PHP 4.3.3 released</td><td>ilia@prohost.org (Ilia Alshanetsky)</td>
<td>08/25/2003 09:53 AM</td></tr>
<tr><td>PHP 5.0.0 Beta 2 released!</td><td>andi@zend.com (Andi Gutmans)</td>
<td>10/30/2003 03:57 PM</td></tr>
<tr><td>PHP 4.3.4 Released</td><td>ilia@prohost.org (Ilia Alshanetsky)</td>
<td>11/03/2003 08:25 PM</td></tr>
<tr><td>PHP 5 Beta 3 Released!</td><td>andi@zend.com (Andi Gutmans)</td>
<td>12/22/2003 05:48 AM</td></tr>
<tr><td>PHP 5 Release Candidate 1</td><td>andi@zend.com (Andi Gutmans)</td>
<td>03/18/2004 12:24 PM</td></tr>
<tr><td>PHP 4.3.5 Released</td><td>ilia@prohost.org (Ilia Alshanetsky)</td>
<td>03/26/2004 08:55 AM</td></tr>
<tr><td>PHP 4.3.6 Released</td><td>ilia@prohost.org (Ilia Alshanetsky)</td>
<td>04/15/2004 05:28 PM</td></tr>
```

13.16 Command-Line PHP

PHP isn't just for web applications. Your PHP installation can include a CLI (Command-Line Interface) version of the PHP interpreter that lets you run PHP scripts as standalone programs. This can be useful for running a PHP program at certain times of day or just reusing code that you wrote for a web application in a different context.

Read about the CLI version of the PHP interpreter in Section 1.4.5 of O'Reilly's *Programming PHP*, *PHP Cookbook* (O'Reilly), Section 20.0 and Recipes 20.1-20.4; and the PHP Manual (http://www.php.net/features.commandline). The PEAR installation instructions in Appendix A use the CLI version of the PHP interpreter.

13.17 PHP-GTK

One advanced use of the CLI PHP interpreter is to use it along with the PHP-GTK functions, which let you write full-featured GUI applications. The existing version of PHP-GTK (1.0.0) works with PHP 4. A new version of PHP-GTK is in the works for PHP 5.

Example 13-21 uses PHP-GTK to display a window with a button in it.

Example 13-21. Displaying a button with PHP-GTK

```
$window =& new GtkWindow( );

$button =& new GTKButton('I am a button, please click me.');
$window->add($button);

$window->show_all( );

function shutdown( ) { gtk::main_quit( ); }
$window->connect('destroy','shutdown');

gtk::main( );
```

The window that Example 13-21 displays is shown in Figure 13-2.

Figure 13-2. Displaying a button with PHP-GTK

Read about PHP-GTK in O'Reilly's *PHP Cookbook*, Recipes 20.5-20.8 and 20.10; and at http://gtk.php.net.

13.18 Even More Things You Can Do with PHP

There are even more extensions and built-in functions available than what's discussed in this chapter. Three good places to look to learn about PHP's function library, extensions, and add-ons are:

The PHP Manual (http://www.php.net/manual/)

> Available in 24 languages, the online PHP Manual has information about all of PHP's built-in functions and lots of user-contributed comments.

The PEAR Package List (http://pear.php.net/packages.php)

> PEAR is a collection of hundreds of add-on packages to PHP. The DB package covered in Chapter 7 is probably the most popular one. This chapter highlights some others. When you need to solve a new problem with PHP, check out PEAR before you start to write your code. Someone may have already solved it for you.

The PECL Package List (http://pecl.php.net/packages.php)

> PECL is another location for finding extensions to PHP. While the packages in PEAR are themselves written in PHP, PECL packages are written in C and provide access to external libraries or other resources.

Appendix A. Installing and Configuring the PHP Interpreter

If you want to write some PHP programs, you need a PHP interpreter to turn them from punctuation-studded text files into actual interactive web pages. The easiest way to get up and running with PHP is to sign up for a cheap or free web-hosting provider that offers PHP? but you can run the PHP interpreter on your own computer, too.

PREV < Day Day Up > NEXT

A.1 Using PHP with a Web-Hosting Provider

If you already have an account with a web-hosting provider, you probably have access to a PHP-enabled server. These days, it is the odd web-hosting provider that *doesn't* have PHP support. Usually, hosting providers configure their servers so that files whose names end in *.php* are treated as PHP programs. To see whether your hosted web site supports PHP, first save the file in Example A-1 on your server as *phptest.php*.

Example A-1. PHP test program

```
<?php print "PHP enabled"; ?>
```

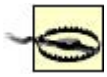
Load the file in your browser by visiting the right URL for your site (e.g., *http://www.example.com/phptest.php*). If you see just the message `PHP enabled`, then your hosted web site supports PHP. If you see the entire contents of the page (`<?php print "PHP enabled"; ?>`), then your hosting provider probably doesn't support PHP. Check with them, however, to make sure that they haven't turned on PHP for a different file extension or made some other nonstandard configuration choice.

If you can't use PHP with your web hosting provider (or you don't have one), the links at http://www.php.net/links.php#hosts are a good place to start when looking for a web-hosting provider that supports PHP.

A.2 Installing the PHP Interpreter

Installing the PHP interpreter on your own computer is a good idea if you don't have an account with a hosting provider, or you just want to experiment with PHP without exposing your programs to the entire Internet. If you're not using a hosting provider and want to install the PHP interpreter on your own computer, follow the instructions in this section. After you've installed the interpreter, you'll be able to run your own PHP programs.

Installing the PHP interpreter is a matter of downloading some files and putting them in the right places on your computer. You must also configure your web server so that it knows about PHP. This section contains instructions on how to do this for computers running Windows, Linux, Unix, and OS X. If you get stuck, check out the installation FAQ at http://www.php.net/manual/faq.installation.



As this section is being written, the final version of PHP 5 is not yet released. The instructions here are for PHP 4 but should be almost identical for PHP 5. The only difference may be in the names of some files or packages? for example, a `php5` Debian package instead of `php4` .For the latest information, see http://www.oreilly.com/catalog/0596005601.

A.2.1 Installing on Windows

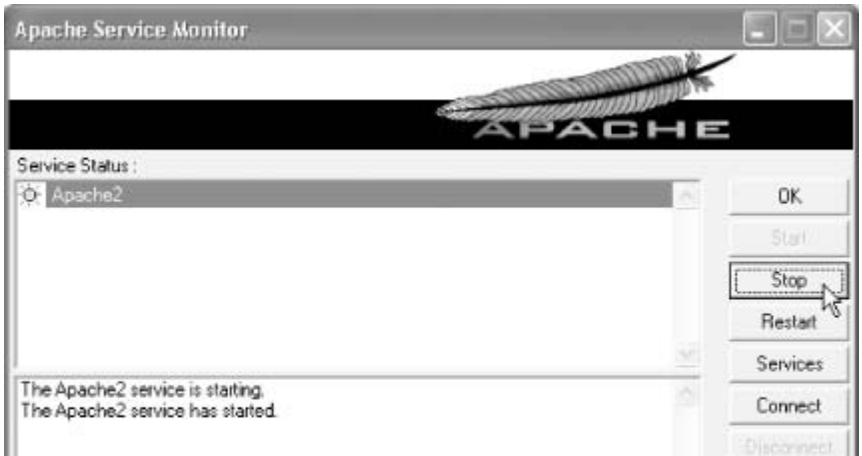
You can install PHP after downloading it from the PHP web site, or you can download a third-party package that integrates PHP, Apache, and MySQL. Installing PHP is a good idea if you already have Apache or MySQL installed, or you want more control over your setup. The integrated packages are a convenient way to get everything up and running in one step.

A.2.1.1 Installing PHP

Download the PHP installation package from http://www.php.net/downloads.php. There are two versions of the Windows download available: the *installer* download and the *zip* download. Use the installer download. It is an installation program that you run after downloading. This program copies the PHP interpreter program and supporting files to the right places and helps you configure your web server program to work with the PHP interpreter. The zip version contains the PHP interpreter and a number of PHP extensions but no installation program. If you use the zip version, then you must copy the PHP interpreter program and other files to the right places. The installer download is easier to deal with.

Your web server should be installed before you run the PHP installer. If you want to use Apache, follow the instructions in the later section Section A.4.1.1. However, Apache should not be running when you install PHP. Bring up the Apache monitor by double-clicking on the Apache Monitor icon in the System Tray, or go to to Start ⟶ All Programs ⟶ Apache HTTP Server 2.0.49 ⟶ Control Apache Server ⟶ Monitor Apache Servers. This displays the window in Figure A-1. Select Apache2 in the Service Status window and click Stop to stop Apache. If Apache is correctly stopped, the Service Monitor looks like Figure A-2.

Figure A-1. Stopping Apache with the Apache Monitor



A.3 Installing PEAR

Many PEAR modules, such as the DB module discussed in Chapter 7, make your PHP programming life easier. They are high-quality code libraries that help you do all sorts of common tasks in PHP programs such as interacting with a database or generating an HTML form. I recommend always having the PEAR libraries available.

Depending on how you have installed PHP (or how your hosting provider has installed PHP), you may need to take extra steps to also install the PEAR base libraries (including DB) and its package management tool. To see whether you have PEAR installed properly, make a short PHP program that just attempts to include *DB.php*, as shown in Example A-2.

Example A-2. Testing for PEAR installation

```
require 'DB.php';
if (class_exists('DB')) {
    print "ok";
} else {
    print "failed";
}
```

If PEAR is installed properly, Example A-2 prints `ok`. PEAR is not installed correctly if the program prints `failed`, you get a blank page, or you see an error message like this:

```
Warning: main(DB.php) [function.main]: failed to open stream:
No such file or directory in /usr/local/apache/htdocs/pearcheck.php on line 2

Fatal error: main( ) [function.require]: Failed opening required 'DB.php'
(include_path='.:/usr/local/php/lib') in
/usr/local/apache/htdocs/pearcheck.php
on line 2
```

The specific steps to take to start the PEAR installation process vary based on your operating system. On Windows, visit http://go-pear.org/ in a web browser and save the contents of that page as *C:\PHP\go-pear.org* (assuming you've installed PHP in *C:\PHP*). Then pass that file to the *php.exe* program. From the command prompt, type:

```
C:
CD \PHP
PHP go-pear.org
```

On Linux, as `root` at a shell prompt, type:

```
lynx -source go-pear.org | php
```

On OS X, at a Terminal shell prompt, type:

```
curl go-pear.org | sudo php
```

After you've started the PEAR installation process in the appropriate way, the next steps are the same on all platforms. The installation program asks a number of questions about how it should install PEAR. Use the default answers for all the questions, including when it asks you whether it should alter your *php.ini* file. The installation process must change the `include_path` setting in *php.ini* so that `require` and `include` work correctly with PEAR libraries.

Once PEAR has been installed successfully, run the PEAR package manager from a command or shell prompt to install and upgrade individual PEAR packages. The package manager is a program called `pear`. On Windows, you may need to be in the *C:\PHP* directory to run `pear`. On Linux, it should work from any directory, but you should be `root` when you run it. On OS X, you should run `sudo pear` so that the program has the appropriate permissions.

The OS X PHP package from *www.entropy.ch* installs its own complete copy of the base PEAR libraries and the PEAR package management tools. Because OS X 10.3.3 comes with a

A.4 Downloading and Installing PHP's Friends

To build a web site with PHP, you need a web server. Apache is the most popular web server in the world. It's free, powerful, stable, and secure. What more could you ask for? You probably want a database program to use with your web site. One of the most common choices for a database program to go along with PHP is MySQL. This section shows you how to install Apache and MySQL on your computer.

The instructions in this section are only for people who are installing PHP on their own computers. If you are using a web-hosting provider's PHP setup, then don't install Apache and MySQL yourself. Your hosting provider has taken care of that for you.

A.4.1 Installing Apache

How you install Apache depends on what operating system you're using. Follow the appropriate instructions for your platform.

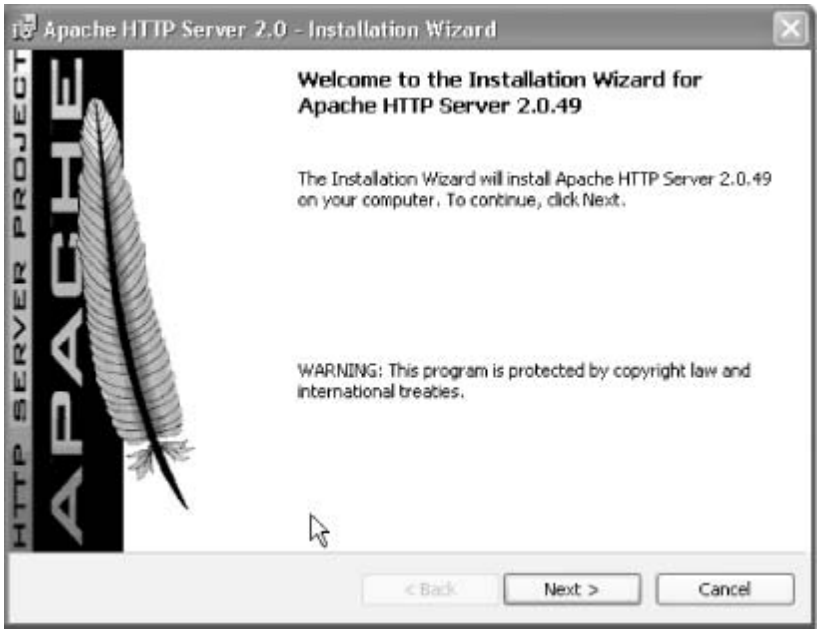
A.4.1.1 Apache on Windows

Take the following steps to install Apache on Windows:

1. Go to http://httpd.apache.org/download.cgi and download the most recent version of the "Win32 Binary (MSI Installer)" for Apache 2. This is in a section of the page titled something like "Apache 2.0.49 is the best available version," and has a filename such as *apache_2.0.49-win32-x86-no_ssl.msi*. (As new versions of Apache are released, the 2.0.49 becomes 2.0.50 or 2.1.0 and so on.)

2. After the Installer downloads, double-click on it to run it. You should see a window like the one in Figure A-15. Click the Next button to begin the installation procedure.

Figure A-15. Beginning the Windows Apache installation



3. Accept the terms of the Apache license agreement as shown in Figure A-16. Read the next screen of background information about Apache and click Next to continue.

Figure A-16. Accepting the Apache license agreement

A.5 Modifying PHP Configuration Directives

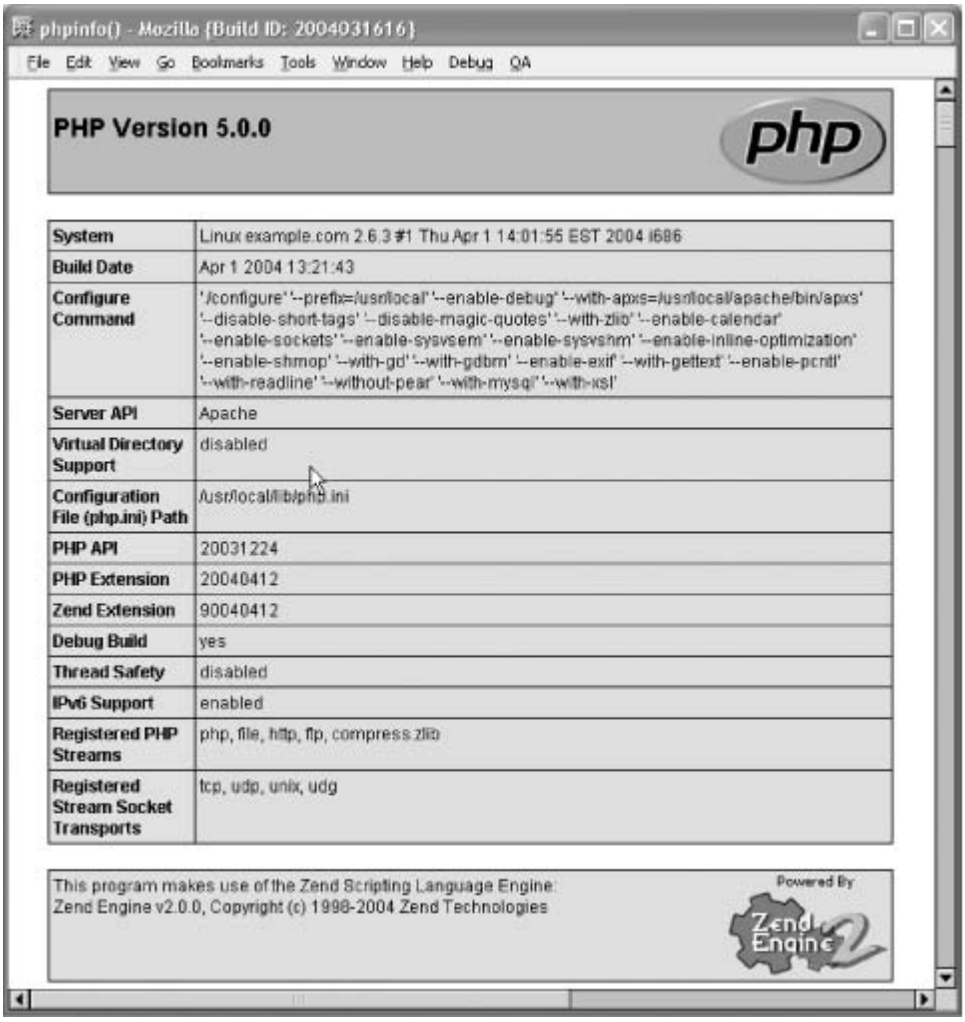
Earlier chapters in the book mention various PHP *configuration directives*. These are settings that affect the behavior of the PHP interpreter, such as how errors are reported, where the PHP interpreter looks for included files and extensions, and much more.

Read this section when you encounter a configuration directive you want to alter or are curious as to how you can tweak the PHP interpreter's settings (whether you are using PHP on your own computer or with a hosting provider). For example, changing the `output_buffering` directive (as discussed in Section 8.6) makes your life much easier if you are working with cookies and sessions.

The values of configuration directives can be changed in a few places: in the PHP interpreter's *php.ini* configuration file, in Apache's *httpd.conf* or *.htaccess* configuration files, and in your PHP programs. Not all configuration directives can be changed in all places. If you can edit your *php.ini* or *httpd.conf* file, it's easiest to set PHP configuration directives there. But if you can't change those files because of server permissions, then you can still change some settings in your PHP programs.

The *php.ini* file holds system-wide configuration for the PHP interpreter. When the web server process starts up, the PHP interpreter reads the *php.ini* file and adjusts its configuration accordingly. To find the location of your system's *php.ini* file, examine the output from the `phpinfo( )` function. This function prints a report of the PHP interpreter's configuration. The tiny program in Example A-3 produces a page that looks like the one in Figure A-21.

Figure A-21. Output of phpinfo( )



Example A-3. Getting configuration details with phpinfo( )

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

In Figure A-21, the sixth line (`Configuration File (php.ini) Path`) shows that the *php.ini*

A.6 Appendix Summary

This appendix covers:

- Using PHP with a web-hosting provider.
- Installing the PHP interpreter on Windows, Linux, or OS X.
- Installing PEAR.
- Installing Apache on Windows, Linux, or OS X.
- Installing MySQL on Windows, Linux, or OS X.
- Using `phpinfo( )` to see the PHP interpreter's configuration.
- Understanding the structure of the *php.ini* configuration file.
- Configuring the PHP interpreter in the *httpd.conf* configuration file.
- Reading and writing configuration directive values with `ini_get( )` and `ini_set( )`.
- Using common configuration directives.

PREV < Day Day Up > NEXT

Appendix B. Regular Expression Basics

Behind the innocuous and generic phrase *regular expression* lives an intricate and powerful world of text pattern matching. With regular expressions, you can make sure that a user really entered a ZIP Code or an email address in a form field, or find all the HTML `<a>` tags in a page. If your web site relies on data feeds that come in text files, such as sports scores, news articles, or frequently updated headlines, regular expressions can help you make sense of these.

This appendix provides an overview of the most useful and commonly encountered parts of the regular expression menagerie. By learning the special meanings of 5 or 10 symbols and 2 or 3 PHP functions, you can use regular expressions to solve most of the text-processing problems you run into when building a web site with PHP. There are some dark corners and steep ravines of the regular expression landscape that are not covered here, however, such as locale support, lookahead and assertions, and conditional subpatterns. To learn more about regular expressions, see the PCRE section of the PHP Manual, at http://www.php.net/pcre, or read the comprehensive *Mastering Regular Expressions* by Jeffrey E.F. Friedl (O'Reilly).

To work with regular expressions in PHP, use the functions in the PCRE (Perl-compatible regular expressions) extension.[1] These functions are included with PHP by default and are described in the online manual at http://www.php.net/pcre. Section B.6, later in this appendix, gives an overview of the PCRE functions. If you're already familiar with regular expression basics, read that section to learn the language-specific details of using regular expressions in PHP.

[1] Generally, it's best to avoid the POSIX regular expression functions: `ereg( )` and friends. They are not as capable as the PCRE functions.

A regular expression is a string. That string defines a pattern that matches other strings. For example, the regular expression `\d{5}(-\d{4})?` matches U.S. ZIP or ZIP+4 Codes:

`\d`
: A digit (0-9)

`{5}`
: A total of five of the previous item (a digit)

`-`
: A literal - character

`\d`
: A digit

`{4}`
: A total of four of the previous item (a digit)

`( )?`
: Makes what's inside the parentheses optional

So, the regular expression `\d{5}(-\d{4})?` matches "five digits, optionally followed by a hyphen and four digits."

Here's another regular expression: `</?[bBiI]>`. This one matches opening or closing HTML `<b>` or `<i>` tags:

`<`
: A literal < character

B.1 Characters and Metacharacters

In a regular expression, some characters match themselves, such as the hyphen in the ZIP Code regex or the `<` in the HTML tag regex. Some characters have special meanings, such as the `?` that makes something optional or the square brackets that mean "one character from the list inside the square brackets." The characters that match themselves are called *literals*. The characters that have special meanings are called *metacharacters*.

A pattern containing only literals matches strings that contain the sequence of literals in the pattern. For example, the pattern `href=` matches the strings `<a href="/">Home</a>`, `schref=`, and `set href=12`.

The metacharacter . (dot) matches any character.[2] So, the pattern `d.g` matches `dog`, `d7g`, `adagio`, `digdug`, and `*d*g*`, among other possibilities. It also matches `d.g`, since dot (the metacharacter) matches a literal . character. Without a quantifier (introduced in Section B.2), dot matches exactly one character. This means that `d.g` doesn't match `ridge` (it has no characters between the `d` and the `g`) or `doug` (it has more than one character between the `d` and the `g`).

[2] This isn't entirely true. By default, dot doesn't match a newline character. Turning on the `s` pattern modifier makes dot match newline, however. This and other pattern modifiers are explained later in this appendix in Section B.6.

The metacharacter `|` (bar) is for alternation. Use alternation to construct a pattern that matches more than one set of characters. For example, `dog|cat` matches strings that contain `dog` or `cat`, such as `dog`, `cathode`, `redogame`, and `hotdog stand`. The pattern `dog|cat` does not mean "match `do`, then either `g` or `c`, then `at`." The alternation text generally includes everything back to the beginning of the pattern or forward to the end of the pattern. However, you can restrict the reach of alternation by enclosing the choices in parentheses. For example, `s(cr|in)ew` means "match `s`, then either `cr` or `in`, then `ew`"? it matches `screw`, `sinew`, and `my screwdriver`, but not `screen` or `deminews`. Without the parentheses, the pattern `scr|inew` means "match `scr` or `inew`." This still matches `screw` and `sinew`, but it also matches `screen` and `deminews`. Alternation can also be used with more than just two choices. For example, `s(cr|in|tr|ch)ew` matches `screw`, `sinew`, `strew`, and `eschew`.

Using parentheses to group together characters for alternation is called *grouping*. (Some things about regular expressions are straightforward.) Grouping also applies to quantifiers, as discussed in the next section. Parentheses also *capture* the text inside them for subsequent use. The characters that match the part of the pattern inside a set of parentheses are stored in a special variable so you can retrieve them later. Capturing is explained later in this appendix in more detail in Section B.6.1 and Section B.6.2.

B.2 Quantifiers

A *quantifier* is a metacharacter that tells "how many." You put a quantifier after an item to indicate you want to match that item a certain number of times. Quantifiers are listed in Table B-1.

Table B-1. Quantifiers

Quantifier	How many times
*	Zero or more
+	One or more
?	Optional (zero or one)
{*x*}	Exactly *x*
{*x*,}	At least *x*
{*x*,*y*}	At least *x*, but no more than *y*

To use a quantifier, put it immediately after the item you want to quantify. Table B-2 shows some regular expressions with quantifiers.

Table B-2. Quantifier examples

Regular expression	Meaning	Matches	Doesn't match
ba+	b, then at least one a	ba, baa, baaa, rhumba, babar	b, abs, taaa-daaa, celeste
ba+na*s	b, at least one a, n, zero or more a, s	turbans, baanas, rhumbanas!	banana, bananas
ba(na){2}	ba, then na twice	banana, bananas, semi-banana, bananarama	cabana, banarama
ba{2,}ba{3,}	b, then at least two a, then b, then at least three a	baabaaa, baaaaabaaaaa, rhumbaabaaas	baabaa, babaaar, banana
(baa-){2,4}baa	baa- at least two, but not more than four times, then baa	baa-baa-baa, baa-baa-baa-baa-baa, oomp-pa-pa-baa-baa-baa-oomp-pa-pa	baa-baa, baa-baad-news
dogs? and cats?(and	dog, then an optional s, then and cat, then an optional s, then	dog and cat and chicken, dog and cat and chickens, hotdogs and cats, dogs and	doggies and cats, dogs and cats or chickens, dogss and

B.3 Anchors

Anchors align a pattern for more specific matching. A pattern such as `ba(na)+` matches `banana` but also `cabana` or `bananarama`. As long as text matching `ba(na)+` is somewhere in a string, the pattern matches. An anchor, however, matches a pattern at the beginning or end of a string. The `^` anchor matches the beginning of a string and the `$` anchor matches the end of a string. For example, this pattern matches strings that begin with `Gre`:

```
^Gre
```

The pattern matches `Green`, `Grey Lantern`, and `Grep is my favorite`, but not `GGreen VVegetables`, `gre`, or `InGres`.

This pattern matches strings that end with an exclamation point:

```
!$
```

It matches "`Zip!`," "`Zoom!`," and "`Pow! Kablam!`," but not "`Kerfloofie.`," "`! is the negation operator`," "`Pow! Oh.`," or "`!!!!!!!!?`."

You can use both anchors in a single pattern to match an entire string. The pattern `^ba(na)+` matches `banana` and `bananarama` but not `cabana`. Similarly, `ba(na)+$` matches `banana` and `cabana` but not `bananarama`. Anchored on both ends, however, `^ba(na)+$` matches only `banana` (and `bananana`, `banananana`, and so on.) This pattern matches various nicknames for the name William:

```
^(w|W|b|B)illy?$
```

It matches `Will`, `will`, `Bill`, `bill`, `Willy`, `willy`, `Billy`, and `billy`, but not `Willa`, `billo`, `twill`, `handbill`, or `William`.

In addition to the `^` and `$` anchors, there are anchor metacharacters that deal with word boundaries. The `\b` anchor matches at a word boundary and `\B` matches everywhere that isn't a word boundary. A word boundary is between one character that is a letter, digit, or underscore and another character that is none of those.[3] So, in the phrase `It's not a_tumor.`, the word boundaries are before the I, before and after the apostrophe, before and after each space, and before and after the period.

[3] More specifically, a word boundary is between a place where something matches `\w` and something does not match `\w`. This includes the beginning of strings that start with word characters and the end of strings that end with word characters. The `\w` metacharacter is discussed in Section B.4.

The word boundary anchors are useful for matching a string that could occur as part of another word. For example, this pattern matches `fish` only when it's not part of a compound word:

```
\b[fF]ish
```

The pattern matches `fish`, `Go fish!`, and `Hamilton Fish High School`, but not `bluefish`, `sportfishing`, or `swordfish`. However, it also matches `sport-fishing`, since a word boundary is between `-` and `f`.

B.4 Character Classes

A *character class* lets you represent a bunch of characters (a "class") as a single item in a regular expression. Put characters in square brackets to make a character class. A character class matches any one of the characters in the class. This pattern matches a person's name or a bird's name:

```
^D[ao]ve$
```

The pattern matches `Dave` or `Dove`. The character class `[ao]` matches either `a` or `o`.

To put a whole range of characters in a character class, just put the first and last characters in, separated by a hyphen. For instance, to match all English alphabetic characters:

```
[a-zA-Z]
```

When you use a hyphen in a character class to represent a range, the character class includes all the characters whose ASCII values are between the first and last character (and the first and last character). If you want a literal hyphen inside a character class, you must backslash-escape it. The character class `[a-z]` is the same as `[abcdefghijklmnopqrstuvwxyz]`, but the character class `[a\-z]` matches only three characters: `a`, `-`, and `z`.

You can also create a *negated character class*, which matches any character that is not in the class. To create a negated character class, begin the character class with `^`:

```
// Match everything but letters
[^a-zA-Z]
```

The character class `[^a-zA-Z]` matches every character that isn't an English letter: digits, punctuation, whitespace, and control characters. Even though `^` is used as an anchor outside of character classes, its only special meaning inside a character class is negation. If you want to use a literal `^` inside a character class, either don't put it first in the character class or backslash-escape it. Each of these patterns match the same strings:

```
[0-9][%^][0-9]
[0-9][\^%][0-9]
```

Each pattern matches a digit, then either `%` or `^`, then another digit. This matches strings such as `5^5`, `3%2`, or `1^9`.

Character classes are more efficient than alternation when choosing among single characters. Instead of `s(a|o|i)p`, which matches `sap`, `sop`, and `sip`, use `s[aoi]p`.

Some commonly used character classes are also represented by dedicated metacharacters, which are more concise than specifying every character in the class. These metacharacters are shown in Table B-3.

Table B-3. Character class metacharacters

Metacharacter	Description	Equivalent class
`\d`	Digits	`[0-9]`
`\D`	Non-digits	`[^0-9]`
`\w`	Word characters	`[a-zA-Z0-9_]`
`\W`	Non-word characters	`[^a-zA-Z0-9_]`
`\s`	Whitespace	`[ \t\n\r\f]`

B.5 Greed

Quantifiers in the PHP interpreter's regular expression engine are *greedy*. This means they match as much as they can. The pattern `<b>.*</b>` means "the string `<b>`, then zero or more characters, then the string `</b>`." The "more" in "zero or more" matches as many characters as possible. When the pattern is applied to the string `<b>Look Out!</b> <i>Caution!</i> <b>Uh-Oh!</b>`, the `.*` matches `Look Out!</b> <i>Caution!</i> <b>Uh-Oh!`. The greediness of the quantifier causes it to skip over the first `</b>` it sees and gobble up characters to the last `</b>` in the string.

To turn a quantifier from greedy to nongreedy, put a question mark after it. The pattern `<b>.*?</b>` still matches "the string `<b>`, then zero or more characters, then the string `</b>`", but now the "more" in "zero or more" matches as few characters as possible. Example B-1 shows the difference between greedy and nongreedy matching with `preg_match_all( )`. (Example B-5 details how `preg_match_all( )` works, including the meaning of the `@` characters at the start and end of the pattern.)

Example B-1. Greedy and nongreedy matching

```
$meats = "<b>Chicken</b>, <b>Beef</b>, <b>Duck</b>";

// With a non-greedy quantifier, each meat is matched separately
preg_match_all('@<b>.*?</b>@',$meats,$matches);
foreach ($matches[0] as $meat) {
    print "Meat A: $meat\n";
}

// With a greedy quantifier, the whole string is matched just once
preg_match_all('@<b>.*</b>@',$meats,$matches);
foreach ($matches[0] as $meat) {
    print "Meat B: $meat\n";
}
```

Example B-1 prints:

```
Meat A: <b>Chicken</b>
Meat A: <b>Beef</b>
Meat A: <b>Duck</b>
Meat B: <b>Chicken</b>, <b>Beef</b>, <b>Duck</b>
```

The nongreedy quantifier in the first pattern makes the first match by `preg_match_all( )` stop short at the first `</b>` it sees. This leaves part of `$meats` to be matched by subsequent applications of the pattern by `preg_match_all( )`.

But with the greedy quantifier in the second example, the first match by `preg_match_all( )` scoops up all of the text, leaving nothing matchable for subsequent applications of the pattern.

B.6 PHP's PCRE Functions

Use the functions in PHP's PCRE extension to work with regular expressions in your programs. These functions allow you to match a string against a pattern and to alter a string based on how it matches a pattern. When you pass a pattern to one of the PCRE functions, it must be enclosed in delimiters. Traditionally, the delimiters are slashes, but you can use any character that's not a letter, number, or backslash as a delimiter. If the character you choose as a delimiter appears in the pattern, it must be backslash-escaped in the pattern, so you should only use a nonslash delimiter when a slash is in your pattern.

After the closing delimiter, you can add one or more pattern modifiers to change how the pattern is interpreted. These modifiers are listed at http://www.php.net/pcre.pattern.modifiers. One handy modifier is `i`, which makes the pattern matching case-insensitive. For example, the patterns (with delimiters) `/[a-zA-Z]+/` and `/[a-z]+/i` produce the same results.

Another useful modifier is `s`, which makes the dot metacharacter match newlines. The pattern (with delimiters) `@<b>.*?</b>@` matches a set of `<b></b>` tags and the text between them, but only if that text is all on one line. To match text that may include newlines, use the `s` modifier:

```
@<b>.*?</b>@s
```

B.6.1 Matching

The `preg_match( )` function tests whether a string matches a pattern. Pass it the pattern and the string to test as arguments. It returns `1` if the string matches the pattern and `0` if it doesn't. Example B-2 demonstrates `preg_match( )`.

Example B-2. Matching with preg_match( )

```
// Test the value of $_POST['zip'] against the
// pattern ^\d{5}(-\d{4})?$
if (preg_match('/^\d{5}(-\d{4})?$/',$_POST['zip'])) {
    print $_POST['zip'] . ' is a valid US ZIP Code';
}

// Test the value of $html against the pattern <b>[^<]+</b>
// The delimiter is @ since / occurs in the pattern
$is_bold = preg_match('@<b>[^<]+</b>@',$html);
```

A set of parentheses in a pattern capture what matches the part of the pattern inside the parentheses. To access these captured strings, pass an array to `preg_match( )` as a third argument. The captured strings are put into the array. The first element of the array (element 0) contains the string that matches the entire pattern, and subsequent array elements contain the strings that match the parts of the pattern in each set of parentheses. Example B-3 shows how to use `preg_match( )` with capturing.

Example B-3. Capturing with preg_match( )

```
// Test the value of $_POST['zip'] against the
// pattern ^\d{5}(-\d{4})?$
if (preg_match('/^(\d{5})(-\d{4})?$/',$_POST['zip'],$matches)) {
    // $matches[0] contains the entire zip
    print "$matches[0] is a valid US ZIP Code\n";
    // $matches[1] contains the five digit part inside the first
    // set of parentheses
    print "$matches[1] is the five-digit part of the ZIP Code\n";
    // If they were present in the string, the hyphen and ZIP+4 digits
    // are in $matches[2]
    if (isset($matches[2])) {
        print "The ZIP+4 is $matches[2];";
    } else {
        print "There is no ZIP+4";
```

B.7 Appendix Summary

Appendix B covers:

- Thinking about what you can use a regular expression for.
- Understanding the difference between literals and metacharacters.
- Using the metacharacters . (dot) and | (bar).
- Using the quantifiers *, +, ?, {*x*}, {*x*,}, and {*x*,*y*}.
- Anchoring a regular expression with ^ or $.
- Anchoring a regular expression with \b or \B.
- Using a character class.
- Using a negated character class.
- Using character class metacharacters such as \d, \D, \w, \W, \s, and \S.
- Understanding greed (in a regular expression context, at least).
- Making quantifiers greedy or nongreedy.
- Matching with preg_match( ).
- Capturing with preg_match( ).
- Matching and capturing with preg_match_all( ).
- Using backreferences in a regular expression.
- Replacing with preg_replace( ).
- Using backreferences when replacing.
- Making an array from a string with preg_split( ).
- Selecting array elements with preg_grep( ).

B.8 Exercises

1. Write a regular expression that flexibly matches a U.S. phone number whether or not it has parentheses around the area code and has its parts separated by spaces, hyphens, or periods. The regular expression should match phone numbers written like this:
   - (718) 498-1043
   - (718) 498 1043
   - 718 498 1043
   - 718 498-1043
   - 718-498-1043
   - 718.498.1043
2. What would you add to a `validate_form( )` function to check that a submitted form field named `username` contains only letters and numbers? Use `if( )`, `preg_match( )`, and a regular expression.
3. Starting with the code from Example 10-3, write a program that retrieves the weather page for your ZIP Code and parses that page with a regular expression to get the current temperature.
4. Write a program that retrieves a remote web page and prints a list of the hyperlinks in that page. Just look for links that look like this: `<a href="http://www.example.com">The Example Page</a>`. Don't worry about links with other attributes in the `<a>` tag.

Appendix C. Answers To Exercises

C.1 Chapter 2

C.1.1 Exercise 1:

1. The opening PHP tag should be `<?php`. There should not be a space between `<?` and `php`.
2. The string '`I'm fine`' should either be enclosed in double quotes ("`I'm fine`") or the apostrophe should be escaped ('`I\'m fine`').
3. The closing PHP tag should be `?>`, not `??>`.

C.1.2 Exercise 2:

```
$hamburger = 4.95;
$milkshake = 1.95;
$cola = .85;
$food = 2 * $hamburger + $milkshake + $cola;
$tax = $food * .075;
$tip = $food * .16;
$total = $food + $tax + $tip;
print "Total cost of the meal is \$$total";
```

C.1.3 Exercise 3:

```
$hamburger = 4.95;
$milkshake = 1.95;
$cola = .85;
$food = 2 * $hamburger + $milkshake + $cola;
$tax = $food * .075;
$tip = $food * .16;
printf("%1d %9s at \$%.2f each: \$%.2f\n", 2, 'Hamburger', $hamburger, 2 *
$hamburger);
printf("%1d %9s at \$%.2f each: \$%.2f\n", 1, 'Milkshake', $milkshake,
$milkshake);
printf("%1d %9s at \$%.2f each: \$%.2f\n", 1, 'Cola', $cola, $cola);
printf("%25s: \$%.2f\n", 'Food and Drink Total', $food);
printf("%25s: \$%.2f\n", 'Total with Tax', $food + $tax);
printf("%25s: \$%.2f\n", 'Total with Tax and Tip', $food + $tax + $tip);
```

C.1.4 Exercise 4:

```
$first_name = 'James';
$last_name = 'McCawley';
$full_name = "$first_name $last_name";
print $full_name;
print strlen($full_name);
```

C.1.5 Exercise 5:

```
$i = 1; $j = 2;
print "$i $j";
$i++; $j *= 2;
print "$i $j";
$i++; $j *= 2;
print "$i $j";
$i++; $j *= 2;
print "$i $j";
$i++; $j *= 2;
print "$i $j";
```

C.2 Chapter 3

C.2.1 Exercise 1:

a. `false`

b. `true`

c. `true`

d. `false`

e. `false`

f. `true`

g. `true`

C.2.2 Exercise 2:

```
Message 3.Age: 12. Shoe Size: 14
```

C.2.3 Exercise 3:

```
$fahr = -50;
$stop_fahr = 50;
print '<table>';
print '<tr><th>Fahrenheit</th><th>Celsius</th></tr>';
while ($fahr <= $stop_fahr) {
    $celsius = ($fahr - 32) * 5 / 9;
    print "<tr><td>$fahr</td><td>$celsius</td></tr>";
    $fahr += 5;
}
print '</table>';
```

C.2.4 Exercise 4:

```
print '<table>';
print '<tr><th>Fahrenheit</th><th>Celsius</th></tr>';
for ($fahr = -50; $fahr <= 50; $fahr += 5) {
    $celsius = ($fahr - 32) * 5 / 9;
    print "<tr><td>$fahr</td><td>$celsius</td></tr>";
}
print '</table>';
```

C.3 Chapter 4

C.3.1 Exercise 1:

```
$population = array('New York, NY' => 8008278,
                    'Los Angeles, CA' => 3694820,
                    'Chicago, IL' => 2896016,
                    'Houston, TX' => 1953631,
                    'Philadelphia, PA' => 1517550,
                    'Phoenix, AZ' => 1321045,
                    'San Diego, CA' => 1223400,
                    'Dallas, TX' => 1188580,
                    'San Antonio, TX' => 1144646,
                    'Detroit, MI' => 951270);
$total_population = 0;
print "<table><tr><th>City</th><th>Population</th></tr>\n";
foreach ($population as $city => $people) {
    $total_population += $people;
    print "<tr><td>$city</td><td>$people</td></tr>\n";

}
print "<tr><td>Total</td><td>$total_population</td></tr>\n";
print "</table>\n";
```

C.3.2 Exercise 2:

1. Use `asort( )` to sort by population.

```
$population = array('New York, NY' => 8008278,
                    'Los Angeles, CA' => 3694820,
                    'Chicago, IL' => 2896016,
                    'Houston, TX' => 1953631,
                    'Philadelphia, PA' => 1517550,
                    'Phoenix, AZ' => 1321045,
                    'San Diego, CA' => 1223400,
                    'Dallas, TX' => 1188580,
                    'San Antonio, TX' => 1144646,
                    'Detroit, MI' => 951270);
$total_population = 0;
asort($population);
print "<table><tr><th>City</th><th>Population</th></tr>\n";
foreach ($population as $city => $people) {
    $total_population += $people;
    print "<tr><td>$city</td><td>$people</td></tr>\n";

}
print "<tr><td>Total</td><td>$total_population</td></tr>\n";
print "</table>\n";
```

2. Use `ksort( )` to sort by city name.

```
$population = array('New York, NY' => 8008278,
                    'Los Angeles, CA' => 3694820,
                    'Chicago, IL' => 2896016,
                    'Houston, TX' => 1953631,
                    'Philadelphia, PA' => 1517550,
                    'Phoenix, AZ' => 1321045,
                    'San Diego, CA' => 1223400,
                    'Dallas, TX' => 1188580,
                    'San Antonio, TX' => 1144646,
                    'Detroit, MI' => 951270);
$total_population = 0;
ksort($population);
print "<table><tr><th>City</th><th>Population</th></tr>\n";
```

C.4 Chapter 5

C.4.1 Exercise 1:

```
function html_img($url, $alt = '', $height = 0, $width = 0) {
    print '<img src="' . $url . '"';
    if (strlen($alt)) {
        print ' alt="' . $alt . '"';
    }
    if ($height) {
        print ' height="' . $height . '"';
    }
    if ($width) {
        print ' width="' . $width . '"';
    }
    print '>';
}
```

C.4.2 Exercise 2:

```
function html_img2($file, $alt = '', $height = 0, $width = 0) {
    print '<img src="' . $GLOBALS['image_path'] . $file . '"';
    if (strlen($alt)) {
        print ' alt="' . $alt . '"';
    }
    if ($height) {
        print ' height="' . $height . '"';
    }
    if ($width) {
        print ' width="' . $width . '"';
    }
    print '>';
}
```

C.4.3 Exercise 3:

```
I can afford a tip of 11% (30)
I can afford a tip of 12% (30.25)
I can afford a tip of 13% (30.5)
I can afford a tip of 14% (30.75)
```

C.4.4 Exercise 4:

Using `sprintf( )` is necessary to ensure that one-digit hex numbers (like 0) get padded with a leading 0.

```
function build_color($red, $green, $blue) {
    $redhex   = dechex($red);
    $greenhex = dechex($green);
    $bluehex  = dechex($blue);
    return sprintf('#%02s%02s%02s', $redhex, $greenhex, $bluehex);
}
```

You can also rely on `sprintf( )`'s built-in hex-to-decimal conversion with the `%x` format character:

```
function build_color($red, $green, $blue) {
    return sprintf('#%02x%02x%02x', $red, $green, $blue);
}
```

C.5 Chapter 6

C.5.1 Exercise 1:

`var_dump($_POST)` prints:

```
array(4) {
  ["noodle"]=>
  string(14) "barbecued pork"
  ["sweet"]=>
  array(2) {
    [0]=>
    string(4) "puff"
    [1]=>
    string(8) "ricemeat"
  }
  ["sweet_q"]=>
  string(1) "4"
  ["submit"]=>
  string(5) "Order"
}
```

C.5.2 Exercise 2:

```
function process_form( ) {
    print "<ul>";
    foreach ($_POST as $element => $value) {
        print "<li> \$_POST[$element] = $value</li>";
    }
    print "</ul>";
}
```

C.5.3 Exercise 3:

```
<?php
$ops = array('+','-','*','/');
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}
function show_form($errors = '') {
    if ($errors) {
        print 'You need to correct the following errors: <ul><li>';
        print implode('</li><li>',$errors);
        print '</li></ul>';
    }
    // the beginning of the form
    print '<form method="POST" action="'.$_SERVER['PHP_SELF'].'">';
    // the first operand
    print '<input type="text" name="operand_1" size="5" value="';
    print htmlspecialchars($_POST['operand_1']) .'"/>';
    // the operator
    print '<select name="operator">';
    foreach ($GLOBALS['ops'] as $op) {
        print '<option';
        if ($_POST['operator'] =  = $op) { print ' selected="selected"'; }
```

C.6 Chapter 7

C.6.1 Exercise 1:

```
<?php
require 'DB.php';
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die("Can't connect: " . $db->getMessage( )); }
$db->setErrorHandling(PEAR_ERROR_DIE);
$db->setFetchMode(DB_FETCHMODE_ASSOC);
$dishes = $db->getAll('SELECT dish_name,price FROM dishes ORDER BY price');
if (count($dishes) > 0) {
    print '<ul>';
    foreach ($dishes as $dish) {
        print "<li> $dish[dish_name] ($dish[price])</li>";
    }
    print '</ul>';
} else {
    print 'No dishes available.';
}
?>
```

C.6.2 Exercise 2:

```
<?php
require 'DB.php';
require 'formhelpers.php'; // load the form element printing functions
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die("Can't connect: " . $db->getMessage( )); }
$db->setErrorHandling(PEAR_ERROR_DIE);
$db->setFetchMode(DB_FETCHMODE_ASSOC);
if ($_POST['_submit_check']) {
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        process_form( );
    }
} else {
    show_form( );
}
function show_form($errors = '') {
    if ($errors) {
        print 'You need to correct the following errors: <ul><li>';
        print implode('</li><li>',$errors);
        print '</li></ul>';
    }
    // the beginning of the form
    print '<form method="POST" action="'.$_SERVER['PHP_SELF'].'">';
    print '<table>';
    // the price
    print '<tr><td>Price:</td><td>';
    input_text('price', $_POST);
    print '</td></tr>';

    // form end
    print '<tr><td colspan="2"><input type="submit" value="Search Dishes">";
    print '</td></tr>';
    print '</table>';
    print '<input type="hidden" name="_submit_check" value="1"/>';
    print '</form>';
}
function validate_form( ) {
    $errors = array( );
    if (! strval(floatval($_POST['price'])) == $_POST['price']) {
```

C.7 Chapter 8

C.7.1 Exercise 1:

```
<?php
$page_count = $_COOKIE['page_count'] + 1;
setcookie('page_count',$page_count);
print "Number of views: $page_count";
?>
```

C.7.2 Exercise 2:

```
<?php
$page_count = $_COOKIE['page_count'] + 1;
if ($page_count =  = 20) {
    // an empty value deletes the cookie
    setcookie('page_count','');
    print "Time to start over.";
} else {
    setcookie('page_count', $page_count);
    print "Number of views: $page_count";
    if ($page_count =  = 5) {
        print "<br/> This is your fifth visit.";
    } elseif ($page_count =  = 10) {
        print "<br/> This is your tenth visit. Aren't you sick of this page
yet?";
    } elseif ($page_count =  = 15) {
        print "<br/> This is your fifteenth visit. Don't you have anything
better to
do?";
    }
}
?>
```

C.7.3 Exercise 3:

Here is the color selection form page:

```
<?php
require 'formhelpers.php';
session_start( );
$colors = array('#ff0000' => 'red',
                '#ff6600' => 'orange',
                '#ffff00' => 'yellow',
                '#0000ff' => 'green',
                '#00ff00' => 'blue',
                '#ff00ff' => 'purple');
if ($_POST['_submit_check']) {
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        process_form( );
    }
} else {
    show_form( );
}
function show_form($errors = '') {
    print '<form method="POST" action="'.$_SERVER['PHP_SELF'].'">';
    if ($errors) {
        print '<ul><li>';
        print implode('</li><li>',$errors);
        print '</li></ul>';
    }
    // Since we're not supplying any defaults of our own, it's OK
```

C.8 Chapter 9

C.8.1 Exercise 1:

```
$stamp = mktime(19,45,0,10,20,2004);
print strftime('Today is day %d of %B and day %j of the year %Y. The time is
%I:%M %p
(also known as %H:%M).', $stamp);
```

C.8.2 Exercise 2:

```
$stamp = mktime(19,45,0,10,20,2004);
print 'Today is day '.date('d',$stamp).' of '.date('F',$stamp).' and day '.
(date('z',$stamp)+1);
print ' of the year '.date('Y',$stamp).'. The time is '.date('h:i A',$stamp);
print ' (also known as '.date('H:i',$stamp).').';
```

C.8.3 Exercise 3:

```
<?php
print '<table>';
print '<tr><th>Year</th><th>Labor Day</th></tr>';
for ($year = 2004; $year <= 2020; $year++) {
    // Get the timestamp for September 1 of $year
    $stamp = mktime(12,0,0,9,1,$year);
    // Advance to the first monday
    $stamp = strtotime('monday', $stamp);
    print "<tr><td>$year</td><td>";
    print date('F j', $stamp);
    print "</td></tr>\n";
}
print '</table>';
?>
```

C.8.4 Exercise 4:

```
<?php
require 'formhelpers.php';
// Set up arrays of months, days, years, hours, and minutes
$months = array(1 => 'January', 2 => 'February', 3 => 'March', 4 => 'April',
                5 => 'May', 6 => 'June', 7 => 'July', 8 => 'August',
                9 => 'September', 10 => 'October', 11 => 'November',
                12 => 'December');
$days = array( );
for ($i = 1; $i <= 31; $i++) { $days[$i] = $i; }
$years = array( );
for ($year = date('Y') -1, $max_year = date('Y') + 5; $year < $max_year;
$year++) {
    $years[$year] = $year;
}
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}
function show_form($errors = '') {
    global $months, $days, $years;
```

C.9 Chapter 10

C.9.1 Exercise 1:

Here's a sample template file, *article.html*:

```
<html>
<head><title>{title}</title></head>
<body>
<h1>{headline}</h1>
<h2>By {byline}</h2>
{article}
<hr/>
<h4>Page generated: {date}</h4>
</body>
</html>
```

Here's the program that replaces the template fields with actual values. It stores the field names and values in an array and then uses `foreach( )` to iterate through that array and do the replacement:

```
<?php
$page = file_get_contents('article.html');
if ($page =  =  = false) {
    die("Can't read article.html: $php_errormsg");
}
$vars = array('{title}' => 'Man Bites Dog',
              '{headline}' => 'Man and Dog Trapped in Biting Fiasco',
              '{byline}' => 'Ireneo Funes',
              '{article}' => "<p>While walking in the park today,
Bioy Casares took a big juicy bite out of his dog, Santa's Little
Helper. When asked why he did it, he said, \"I was hungry.\"</p>",
              '{date}' => date('l, F j, Y'));
foreach ($vars as $field => $new_value) {
    $page = str_replace($field, $new_value, $page);
}
$result = file_put_contents('dog-article.html', $page);
if (($result =  =  = false) || ($result =  = -1)) {
    die("Couldn't write dog-article.html: $php_errormsg");
}
?>
```

C.9.2 Exercise 2:

Here's a sample *addresses.txt*:

```
brilling@tweedledee.example.com
slithy@unicorn.example.com
uffish@knight.example.net
slithy@unicorn.example.com
jubjub@sheep.example.com
tumtum@queen.example.org
slithy@unicorn.example.com
uffish@knight.example.net
manxome@king.example.net
beamish@lion.example.org
uffish@knight.example.net
frumious@tweedledum.example.com
tulgey@carpenter.example.com
vorpal@crow.example.org
beamish@lion.example.org
mimsy@walrus.example.com
frumious@tweedledum.example.com
raths@owl.example.net
```

C.10 Chapter 11

C.10.1 Exercise 1:

```
$menu=<<<_XML_
<?xml version="1.0" encoding="utf-8" ?>
<rss version="0.91">
 <channel>
  <title>What's For Dinner</title>
  <link>http://menu.example.com/</link>
  <description>These are your choices of what to eat tonight.</description>
  <item>
   <title>Braised Sea Cucumber</title>
   <link>http://menu.example.com/dishes.php?dish=cuke</link>
   <description>Gentle flavors of the sea that nourish and refresh
you.</description>
  </item>
  <item>
   <title>Baked Giblets with Salt</title>
   <link>http://menu.example.com/dishes.php?dish=giblets</link>
   <description>Rich giblet flavor infused with salt and spice.</description>
  </item>
  <item>
   <title>Abalone with Marrow and Duck Feet</title>
   <link>http://menu.example.com/dishes.php?dish=abalone</link>
   <description>There's no mistaking the special pleasure of
abalone.</description>
  </item>
 </channel>
</rss>
_XML_;
$xml = simplexml_load_string($menu);
print "<ul>\n";
foreach ($xml->channel->item as $item) {
    print '<li><a href="' . $item->link .'">' . $item->title ."</a></li>\n";
}
print '</ul>';
```

C.10.2 Exercise 2:

```
<?php
  // Load form helper functions
require 'formhelpers.php';
if ($_POST['_submit_check']) {
    // If validate_form( ) returns errors, pass them to show_form( )
    if ($form_errors = validate_form( )) {
        show_form($form_errors);
    } else {
        // The submitted data is valid, so process it
        process_form( );
    }
} else {
    // The form wasn't submitted, so display
    show_form( );
}
function show_form($errors = '') {
    if ($errors)) {
        print 'You need to correct the following errors: <ul><li>';
        print implode('</li><li>',$errors);
        print '</li></ul>';
    }
    // the beginning of the form
    print '<form method="POST" action="'.$_SERVER['PHP_SELF'].'">';
    // title
```

C.11 Chapter 12

C.11.1 Exercise 1:

The error message looks like:

```
Parse error: parse error, unexpected T_GLOBAL in exercise-12-1.php on line 6
```

The `global` declaration has to be on a line by itself, not inside the `print` statement. To fix the program, separate the two:

```
<?php
$name = 'Umberto';
function say_hello( ) {
    global $name;
    print 'Hello, ';
    print $name;
}
say_hello( );
?>
```

C.11.2 Exercise 2:

```
function validate_form( ) {
    $errors = array( );
    // Capture the output of var_dump( ) with output buffering
    ob_start( );
    var_dump($_POST);
    $vars = ob_get_contents( );
    ob_end_clean( );
    // Send the output to the error log
    error_log($vars);
    // operand 1 must be numeric
    if (! strlen($_POST['operand_1'])) {
        $errors[  ] = 'Enter a number for the first operand.';
    } elseif (! floatval($_POST['operand_1']) =  = $_POST['operand_1']) {
        $errors[  ] = "The first operand must be numeric.";
    }
    // operand 2 must be numeric
    if (! strlen($_POST['operand_2'])) {
        $errors[  ] = 'Enter a number for the second operand.';
    } elseif (! floatval($_POST['operand_2']) =  = $_POST['operand_2']) {
        $errors[  ] = "The second operand must be numeric.";
    }
    // the operator must be valid
    if (! in_array($_POST['operator'], $GLOBALS['ops'])) {
        $errors[  ] = "Please select a valid operator.";
    }
    return $errors;
}
```

C.11.3 Exercise 3:

Change the beginning of the program to:

```
<?php
require 'DB.php';
require 'formhelpers.php';
// Connect to the database
$db = DB::connect('mysql://hunter:w)mp3s@db.example.com/restaurant');
if (DB::isError($db)) { die ("Can't connect: " . $db->getMessage( )); }
function db_error_handler($error) {
    error_log('DATABASE ERROR: ' . $error->getDebugInfo( ));
    die('There is a ' . $error->getMessage( ));
```

C.12 Appendix B

C.12.1 Exercise 1:

The regular expression `^\(?\d{3}\)?[- \.]\d{3}[- \.]\d{4}$` matches "an optional literal `(`, then three digits, then an optional literal `)`, then either a hyphen, space, or period, then three digits, then either a hyphen, space, or period, then four digits." The `^` and `$` anchors make the expression match only phone numbers, not larger strings that contain phone numbers.

C.12.2 Exercise 2:

```
if (! preg_match('/^[a-z0-9]$/i', $_POST['username'])) {
    $errors[  ] = "Usernames must contain only letters or numbers.";
}
```

C.12.3 Exercise 3:

```
$zip = 98052;
$url = 'http://www.srh.noaa.gov/zipcity.php?inputstring=' . $zip;
$weather_page = file_get_contents($url);
if (preg_match('@<br><br>(-?\d+)&deg;F<br>\((-?\d+)&deg;C\)</td>@',
$weather_page,$matches)) {
    // $matches[1] is the Fahrenheit temp
    // $matches[2] is the Celsius temp
    print "The current temperature is $matches[1] degrees.";
} else {
    print "Can't get current temperature.";
}
```

C.12.4 Exercise 4:

```
$url = 'http://www.sklar.com/';
$page = file_get_contents($url);
if (preg_match_all('@<a href="[^"]+">.+?</a>@', $page, $matches)) {
    foreach ($matches[0] as $link) {
        print "$link <br/>\n";
    }
}
```

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *Learning PHP 5* is an eagle. Eagles fall into the category of bird known as "raptors," a category that also includes falcons and hawks. There are two types of raptor: grasping killers, with beaks shaped for tearing and cutting and short toes with curved claws designed for killing; and grasping holders, with beaks shaped for tearing and biting, and longer toes designed for holding. Eagles are grasping killers. Sea eagles have special adaptations to their toes that enable them to grasp smooth prey such as fish. Their excellent vision enables all eagles to spot prey from the air or a high perch. The eagle then swoops down, grabs its prey, and takes off in flight again, in one graceful movement. Eagles often eat their victims while still flying, breaking them apart and discarding the nonedible parts to lighten their load. Eagles, like most raptors, often dine on sick or wounded animals.

There are more than 50 species of eagle spread throughout the world, with the exception of New Zealand and Antarctica. All species of eagles build nests, or aeries, high above the ground, in trees or on rocky ledges. A pair of eagles will use the same nest year after year, lining it with green leaves and grass, fur, turf, or soft materials. The eagle will add to its nest each year. The largest eagle nest ever found was 20 feet deep and 10 feet across.

Hunting, increased use of pesticides, and the diminishment of their natural environment, with the attendant reduction in food sources, have endangered many species of eagle.

Mary Brady was the production editor and the copyeditor for *Learning PHP 5*. Leanne Soylemez was the proofreader. Mary Anne Weeks Mayo and Claire Cloutier provided quality control. Judy Hoer wrote the index.

Hanna Dyer designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Mary Brady.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Ellie Cutler) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, Ellie Cutler, and Jeff Liggett.

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

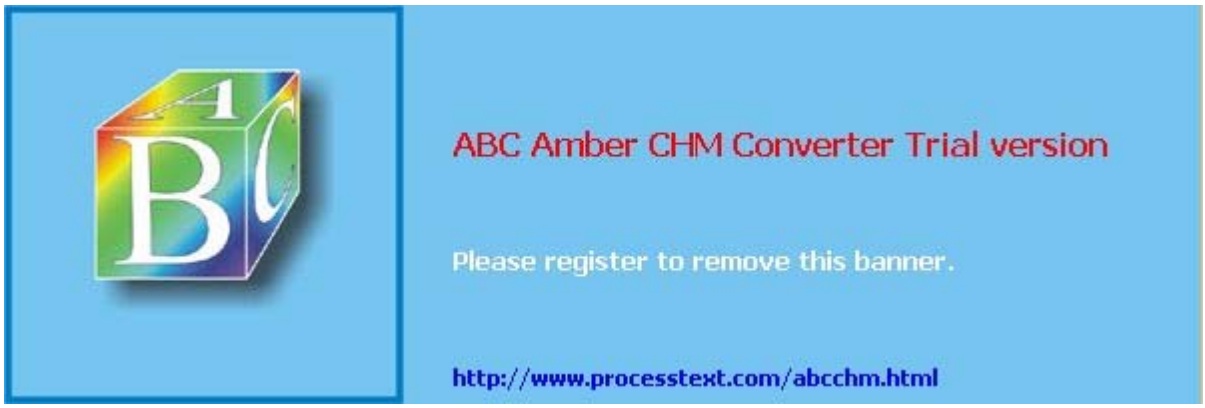
PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

Java extension for PHP
JavaScript in submitted form data, validating

PREV < Day Day Up >



PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [**N**] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

Zend IDE text editor

PREV < Day Day Up >