

THOMSON
COURSE TECHNOLOGY

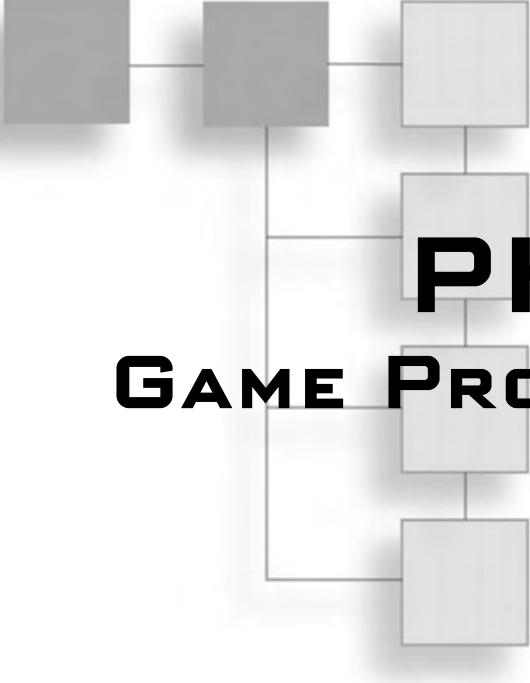
PHP

GAME PROGRAMMING

MATT RUTLEDGE

Author Matt
Rutledge, CEO of Game
Code LLC





PHP

GAME PROGRAMMING



MATT RUTLEDGE

THOMSON
*
COURSE TECHNOLOGY
Professional ■ Trade ■ Reference

© 2004 by Premier Press, a division of Course Technology. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system without written permission from Course PTR, except for the inclusion of brief quotations in a review.

The Premier Press logo and related trade dress are trademarks of Premier Press and may not be used without written permission.

Paint Shop Pro 8 is a registered trademark of Jasc Software.

PHP Coder is a trademark of phpIDE.

All other trademarks are the property of their respective owners.

Important: Course PTR cannot provide software support. Please contact the appropriate software manufacturer's technical support line or Web site for assistance.

Course PTR and the author have attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

Information contained in this book has been obtained by Course PTR from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Course PTR, or others, the Publisher does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from use of such information. Readers should be particularly aware of the fact that the Internet is an ever-changing entity. Some facts may have changed since this book went to press.

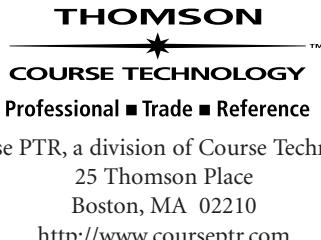
Educational facilities, companies, and organizations interested in multiple copies or licensing of this book should contact the publisher for quantity discount information. Training manuals, CD-ROMs, and portions of this book are also available individually or can be tailored for specific needs.

ISBN: 1-59200-153-X

Library of Congress Catalog Card Number: 2004090731

Printed in the United States of America

04 05 06 07 08 BH 10 9 8 7 6 5 4 3 2 1



Course PTR, a division of Course Technology
25 Thomson Place
Boston, MA 02210
<http://www.courseptr.com>

**SVP, Course Professional, Trade,
Reference Group:**

Andy Shafran

Publisher:

Stacy L. Hiquet

Senior Marketing Manager:

Sarah O'Donnell

Marketing Manager:

Heather Hurley

Series Editor:

André LaMothe

Manager of Editorial Services:

Heather Talbot

Senior Acquisitions Editor:

Emi Smith

Associate Marketing Manager:

Kristin Eisenzopf

Project Editor and Copy Editor:

Dan Foster, Scribe Tribe

Technical Reviewer:

John Freitas

Retail Market Coordinator:

Sarah Dubois

Interior Layout:

Marian Hartsough

Cover Designer:

Steve Deschene

CD-ROM Producer:

Brandon Penticuff

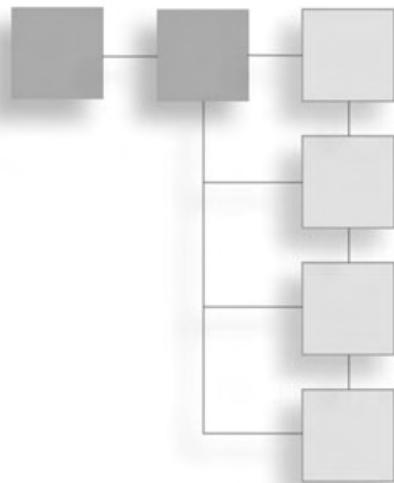
Indexer:

Kelly Talbot

Proofreader:

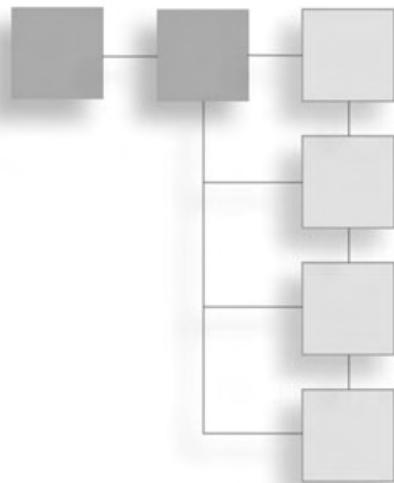
Estelle Manticas

To Jenai. Thank you.



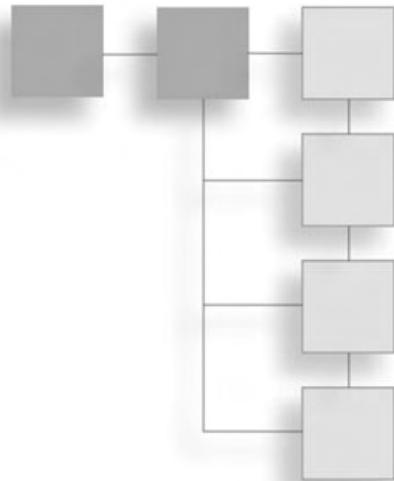
ACKNOWLEDGMENTS

There are so many people who made this book possible for me. First I would like to thank André LaMothe for giving me this opportunity. I would also like to thank my editor, Dan Foster, for all of his hard work. I also need to thank my company, DATA Inc. (www.datausa.com) for understanding my rigorous schedule. Thank you to Mike Gesner (President and CEO), Michael Melson (Technical Director), Randy Chong (Game Design Director), Wesley Potter (Game Designer), and Jonathan Shusta (Art Director) of Dragonfly Game Design (www.dragonflygamedesign.com) for making the MMO possible. Without you guys this book wouldn't have been as cool as it is. I would also like to thank Jenai for being my constant support and putting up with my odd hours, odd moods, and general weirdness throughout the creation of this whole book.



ABOUT THE AUTHOR

MATT RUTLEDGE (ruts@datausa.com) is lead developer at DATA, Inc. (www.datausa.com), a Denver, Colorado-based digital visualization company specializing in 3D computer animation and Web development. Matt has 7 years of experience in the multimedia world. He developed several telecommunication applications before moving into the realm of Web development. Matt specializes in database-driven and interactive online applications. Matt also actively writes articles for [asp101](http://www.asp101.com) (www.asp101.com) and the MSDNAA (www.msdnnaa.net).



ABOUT THE SERIES I

ANDRÉ LAMOTHE, CEO, Xtreme Games LLC, has been involved in the computing industry for more than 25 years. He wrote his first game for the TRS-80 and has been hooked ever since! His experience includes 2D/3D graphics, AI research at NASA, compiler design, robotics, virtual reality, and telecommunications. His books are top sellers in the game programming genre, and his experience is echoed in the Premier Press *Game Development* books.

Letter from the Series Editor

Over the last half decade or so, a little piece of software called PHP has been generating a lot of buzz and has quickly risen from a single programmer's side-project to the Web programming language. Now in version 5, PHP (a recursive abbreviation for PHP Hypertext Preprocessor) is practically synonymous with the development of dynamic Web sites and flexible content. But what few realize is that it's capable of a lot more than just making Web sites—in fact, it's showing serious potential for making games! With the surge in popularity of Web-based games and applications like Habbo Hotel, Yahoo's board, card and puzzle games, and even The Sims, there's never been a better time to get involved in online game development. And with PHP, a completely free tool, it has never been cheaper or easier!

That's what's so exciting about *PHP Game Programming*. Matt Rutledge has taken the formidable tasks of teaching both game development and the PHP language and condensed them into one easy-to-read text that makes it all seem effortless. But what makes this book so special is that it's not just meant for game developers; anyone who wants to learn PHP will find that this book teaches it in a fun way that's far more engaging than the countless other PHP books on the shelves. After all, making games is a lot more fun than sorting employee records! But don't think this book is all about goofing off. From Chapter 1 on, you get a hardcore PHP education that starts in the trenches, with everything you need to know about server configuration and setup. Once you're locked and loaded, you'll complete a tour of the language itself, covering all the major constructs and features. PHP's syntax is remarkably similar to languages like C, C++, and Java, and if you're already familiar with one of those then this material will be very easy to absorb. Finally, the entire last half of the book is devoted to serious game programming topics, from graphics to chess algorithms to dynamic battlefield terrains.

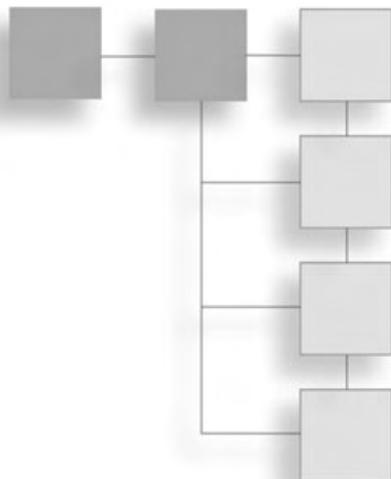
In conclusion, it won't be long before everything from your shoes to your refrigerator has a connection to the Internet, and online games are certainly leading the way toward the ultra-connected world of the future. Many programmers find themselves leaving languages like C or C++ behind in favor of languages with built-in networking capabilities like Java, Perl, and PHP. With a development tool as powerful and accessible as PHP, this is your chance to turn your online game ideas into a reality!

Sincerely,

A handwritten signature in black ink that reads "André LaMothe". The signature is fluid and cursive, with "André" on top and "LaMothe" below it, though the two words are somewhat interconnected.

André LaMothe
Series Editor, Premier Game Development Series

CONTENTS



Introduction	xiii
--------------------	------

PART I INTRODUCTION TO THE WORLD OF PHP..... 1

Chapter 1	So What Is All This Server Stuff?	3
	Understanding the Client/Server Relationship	3
	The Web Server.....	5
	Sessions and Session Variables	5
	TCP/IP.....	13
	Installing the IIS Web Server	14
	Installation on Windows 2000/XP Professional.....	15
	Installation on Windows 98	17
	Installing the Apache Web Server on Windows ME/XP	18
	Installing the Apache Web Server on UNIX	19
Chapter 2	Waging the Configuration War.....	23
	The Platforms	23
	Building and Installing PHP on UNIX	24
	Installation on Windows for IIS/Apache	26
	Installing the Windows Extensions	29
	Testing Your Installation	29

Chapter 3	I Have Conquered the Server, Let Me at the Code!	31
	The Basics of the HTML Tag	31
	The Almighty HTML Document.	32
	The HTML Body.	33
	Graphics and HTML.	36
	The File Formats	37
	Using the File Formats in Your Document	39
	Tables	41
	Layouts, Tables, and Graphics.	44
	Creating Forms for Input.	48
	Conclusion.	51

PART II ENTER THE LANGUAGE **53**

Chapter 4	Say Hello to PHP	55
	Creating a PHP Page	56
	Data Types.	57
	Type Casting	58
	Variable Variables	58
	Constants.	58
	Naming Conventions.	59
	Functions for Variables	60
	Functions for Strings	61
	printf() and sprintf()	64
	Regular Expressions and Pattern Matching.	66
	Using the Regular Expression Functions	68
	Processing Forms with PHP	71
	Conclusion.	76
Chapter 5	Operators, Statements, and Functions	77
	Operators	77
	Arithmetic Operators	78
	Comparison Operators	78
	Logical Operators	79
	Ternary Operator.	80
	Bitwise Operators	82
	Variable Assignment Shortcuts	85
	Operation Precedence.	86

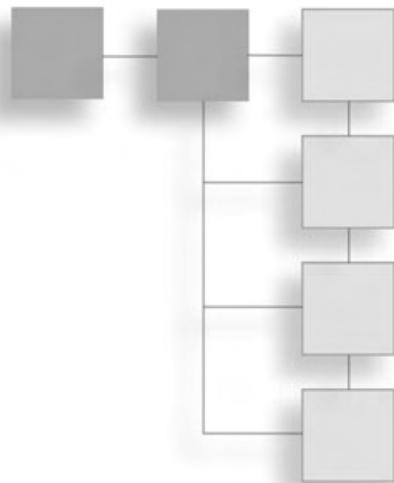
Statements	86
if Statements	88
The switch Statement	90
while and do...while Loops	93
The for Loop	94
Functions	94
Passing Parameters to a Function	95
Recursion	98
The Magic of Including Files	99
Conclusion	100

PART III ARRAYS, GAMES, AND GRAPHICS 101

Chapter 6 Arrays!	103
Initializing Arrays	104
Using Strings for Indexes	105
Looping through Sequential Arrays	106
Looping through Non-Sequential Arrays	108
Multi-Dimensional Arrays	110
Sorting Arrays	112
Your First PHP Game	117
Conclusion	132
Chapter 7 Playing with Chess and Databases	133
Non-Relational Databases	134
Creating and Opening a Database	134
Looping through the Database	136
Inserting an Entry into Your Database	137
Updating an Entry in Your Database	139
Deleting an Entry from Your Database	140
Chess Programming: A Quick Overview	140
Starting the Chess Game	141
Working with the Pieces	143
Getting the User Input and Modifying the Database	151
Conclusion	156

Chapter 8	GD Graphics Overview	157
What Is GD?	157	
Installing GD	158	
Creating and Using a New Image	160	
How to Use Colors	161	
Allocating Colors to an Image	161	
Filling the Image	162	
Setting Your Transparent Color	163	
Converting a True-Color Image to a Palette Image	165	
Counting Colors in an Image	165	
Retrieving a Color at a Point	166	
Drawing Basic Shapes on Your Empty Canvas	167	
Pixels and Lines	167	
From Lines to Rectangles	169	
From Rectangles to Polygons	172	
From Polygons to Arcs and Ellipses	174	
Creating Images with Text	179	
Saving Your Images	184	
Using Existing Images	185	
Conclusion	192	
Chapter 9	Creating Battle Tank and Using Dynamic Terrain	193
Planning Battle Tank	193	
Creating the Graphics	195	
Creating the Game Logic	196	
Creating Dynamic Terrain	205	
Conclusion	209	
PART IV	EXTRAS AND FINAL PROJECTS	211
Chapter 10	PHP and Sockets	213
Socket Basics	213	
Creating a Server	220	
Creating the Client	223	
Integrating Sockets with Battle Tank	224	
Conclusion	228	

Chapter 11	Kiddy Cartel—Creating Your Own MMO	229
Installing mySQL	230	
Relational Databases: A Quick Rundown	231	
Kiddy Cartel: The Rules and Specifications	233	
Creating Your Base Actions	235	
Creating a Command with Sub-Commands	240	
Creating a Command without Sub-Commands	245	
Look at All the Commands...Now What?	248	
Conclusion	256	
Chapter 12	Building Your PHP Skills	257
PHP and Ming	257	
How to Create a Flash Movie	259	
Drawing to Your Flash Movie	262	
Filling Objects with Ming	265	
Adding Animation to Your Flash Movie	271	
Adding ActionScript to Your Flash Piece	275	
Conclusion	279	
PART V	APPENDICES	281
Appendix A	HTML Language Reference	283
Appendix B	PHP Language Reference	309
Appendix C	Support—Debugging Applications	329
Syntax Errors	329	
Semantic Errors	331	
Logic Errors	332	
PHP and Error Reporting	333	
Handling Errors	334	
Application and Installation Problems	335	
Appendix D	GD SDK Language Reference	341
Index	344	



INTRODUCTION

Over the past few years the World Wide Web has grown tremendously. From its infant stages when a Web page was nothing more than text with HTML to dynamic, robust, extensible, rich multimedia content. Five years ago you would never have thought of playing a game on the Web, but today, with current scripting languages, you can do just that. PHP has transformed the Web as we know it. PHP provides quick, dynamic, real-time tools to bring life to Web sites.

PHP (otherwise known as the hypertext preprocessor) burst onto the scene in 1994 when Rasmus Lendorf released a package of “Personal Home Page” tools to the public. As more interest for these tools grew, Rasmus decided to create his own scripting engine to parse input from an HTML form. The first version of PHP was born; it was called PHP/FI.

The programming community quickly grew out of PHP/FI, and PHP soon became the API as you know it today. If you know C/C++ or Java then learning the basic PHP constructs will be extremely easy.

PHP is a wonderful tool with quick, on-the-fly, compilation. It also offers you a ton of libraries to work with to create graphics, Flash pieces, connections to databases, and connections to other computers.

The main focus of this book is to take all of these tools and give you the knowledge and power to create turn-based games on the Web.

Why PHP?

You might ask, why PHP? What is the difference between PHP and another embedded scripting language such as ASP? The main difference between these languages is obviously syntax. But beyond that PHP offers you tons of libraries, all for free. The interpretation of code is faster than ASP, and connecting to a database requires only two to three lines of code. The best part of PHP is that it is an object-oriented language, which inherently gives you a great deal of flexibility in your games.

Who Should Read This Book?

This book is for anyone who wants to implement any Web-based games. An understanding of HTML would be helpful. I have included, however, a section dedicated to teaching you some basics of HTML. Knowledge of your operating system will be helpful too. Since PHP is a cross-platform language I can cover only so much in the way of installing PHP. If something goes wrong during installation it will help if you know your operating system. This book is not for someone with advanced PHP skills.

What's On the CD?

- All the source code in the book
- The PHP installer for Windows and UNIX
- PHP libraries
- Evaluation edition of Jasc Paint Shop Pro 8
- ST Software's PHP Integrated Development Environment
- MySQL 4.0

How This Book Is Organized

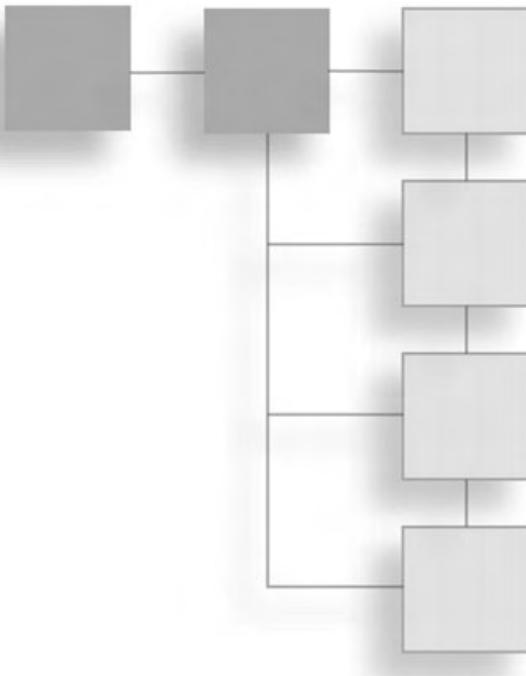
I intended this book to be read in sequential order. If you are a beginner and this is your first exposure to PHP I suggest that you read the book in this manner. However, if you already have a knowledge of PHP you can feel free to skip around from chapter to chapter.

This book is broken into four main parts. The first part of this book walks you through setting up PHP and your Web server. The second part introduces you to PHP itself. In the third part of this book you start creating your very own PHP Web-based games. In the fourth part of this book you take a look at all of the cool stuff PHP can do.

Remember that all of the code that is shown in this book is provided for you on the included CD. There are also several applications included on the CD to help you develop your PHP games.

- **Chapter 1—So What Is All This Server Stuff?** This chapter gives you an overview of server architecture and how the Web works generally. You will also learn how to install an IIS server and an Apache Web server.
- **Chapter 2—Waging the Configuration War.** After you have installed your Web server, this chapter will walk you through the steps of installing the PHP CGI Interpreter. This chapter will also show you some of the basics for configuring PHP.
- **Chapter 3—I Have Conquered the Server, Let Me at the Code!** This chapter will teach you the basics of HTML. It also includes a brief explanation of how images work in the World Wide Web.
- **Chapter 4—Say Hello to PHP.** This chapter introduces you to the PHP language itself. You will learn the basic points of PHP, such as how to create a PHP page, how to declare variables in PHP, and how to use regular expressions.
- **Chapter 5—Operators, Statements, and Functions.** In this chapter you will learn about arithmetic, logic, and bitwise operators. You will also learn how to create logic statements and functions.
- **Chapter 6—Arrays!** This chapter will teach you how to allocate and use arrays in PHP. After you have learned all the finer points of arrays you will create your first game.
- **Chapter 7—Playing with Chess and Databases.** In this chapter you will learn about non-relational databases. You will also create a simple chess game that uses a non-relational database to keep track of moves.
- **Chapter 8—GD Graphics Overview.** This chapter introduces Boutell's GD graphics library. You will learn how to create graphics, manipulate graphics, and add dynamic text to your graphics.
- **Chapter 9—Creating Battle Tank and Using Dynamic Terrain.** In this chapter you will first create a Scorched Earth game remake called Battle Tank. After you have created the initial game you will add dynamic terrain.
- **Chapter 10—PHP and Sockets.** This chapter introduces you to socket programming with PHP. It describes how to send and receive data through sockets.
- **Chapter 11—Kiddy Cartel—Creating Your Own MMO.** Here you will create your very own MMO game. In this game you will control your own neighborhood. It's basically like kids meet mafia.

- **Chapter 12—Building Your PHP Skills.** In this final chapter you will take a look at additional applications for PHP, such as using Ming to create dynamic Flash movies.
- **Appendix A—HTML Language Reference**
- **Appendix B—PHP Language Reference**
- **Appendix C—Support—Debugging Applications**
- **Appendix D—GD SDK Language Reference**



PART I

INTRODUCTION TO THE WORLD OF PHP

CHAPTER 1

So What Is All This Server Stuff? 3

CHAPTER 2

Waging the Configuration War 23

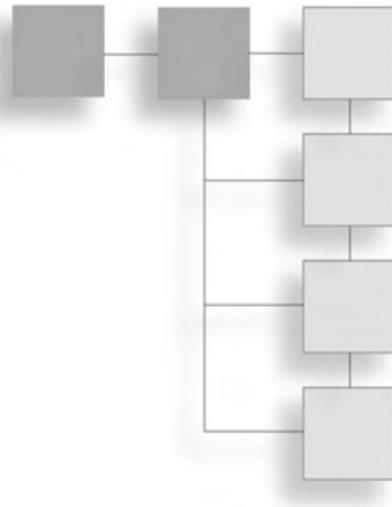
CHAPTER 3

I Have Conquered the Server, Let Me at the Code! 31

This page intentionally left blank

CHAPTER 1

So What Is All This Server Stuff'



- How the Client/Server Relationship Works
- The Web Server
- Sessions and Session Variables
- TCP/IP
- Installing the IIS Web Server
- Installing the Apache Web Server on Windows
- Installing the Apache Web Server on UNIX

Before getting to the configuration and installation of PHP, it is important to know how a Web server and Web browser work together. Accordingly, this chapter will cover some basic server architecture and discuss some points that will change the way you design your applications. If you feel like you already know these concepts, you may wish to skip this section and move on to the section on installation.

Understanding the Client/Server Relationship

When you talk about viewing a PHP page, you are referring to a Web page. When you go to a Web page a series of events occur. These events start at the client with the request, go to a server to get the page, and end back at the client for viewing. Take a look at these events as listed below, and depicted in Figure 1.1.

1. The client computer connects to the Internet.
2. The client opens a Web browser.

3. The client requests a page from a Web site. When you do this, a message is sent over the Internet to a name server, and the name server then directs you to the server that hosts the Web page.
4. The server that hosts the Web page receives your request and retrieves the requested page.
5. If the page is a scripted page, such as a PHP page, the server compiles the page through a just-in-time compiler that generates HTML.
6. The server then transmits the completed HTML back to the client (the browser).
7. The Web browser receives the HTML and displays an interpretation of the page.

All of that is fairly straightforward; it's almost like magic. There is only one catch—from browser to browser, your Web pages are not displayed identically. That is why we say the browser *interprets* the page. For instance, when Internet Explorer receives a page that includes nested tables, both with heights of 100%, Internet Explorer interprets the height of the inner table relative to the height of the outer table. However, in Netscape Navigator, the browser will make both the inner and the outer table the height of the browser window. This poses a problem if you want a page to look the same in multiple browsers. It gets worse when you switch between OS platforms. Let's say you have a page of text, and for some reason the text determines the layout of the page, and you get it looking perfect on a Windows platform. When you then view the page on a Macintosh, the font is

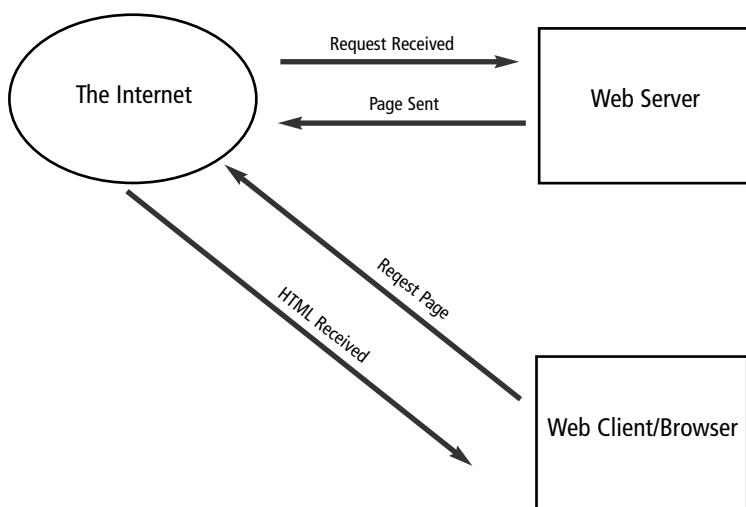


Figure 1.1 The client/server process.

rendered very differently. The kerning (the space between characters) and the leading (the space between lines) is totally different. In fact, the font itself might be changed to a different font altogether. Style sheets can mitigate some of these problems, but you'll still run into layout issues. The point is that what the browser displays is not a photograph, so if you want your games to work on several different browsers you have to be careful about the types of elements (text and graphics) you use.

The Web Server

Now that you have a general understanding of how a client/server relationship works you can take a closer look at the heart of the beast: the Web server itself.

A Web server runs what is called an HTTP daemon. This daemon handles all of the requests received on a particular port. The HTTP daemon will listen to two ports—port 80 and port 443. Port 80 is the general Web port (<http://>). Port 443 is the standard for secure socket (<https://>). A Web server is also said to be *stateless*. That is to say, no permanent connection is maintained between the client and the server. This is extremely important to understand. It will make debugging or solving certain problems much easier, and, more important, this concept will completely change the way you design your games. It won't be as easy as making a library of functions or a simple engine and calling functions. You will need to develop a way to keep state.

Think about that for a minute. What does that really mean? Well, it means that every time a particular event happens in your PHP game—e.g., the user enters coordinates and clicks the fire button—the Web browser will reopen the connection to the server, resend the request to the server, and the server will then process the page and send it back. Now you need to reload all of your variables/states and update the page appropriately. Otherwise the whole game would start over, and that would be no fun. Don't get me wrong—there are some client-side scripting languages, such as JavaScript, that you can use for client-side event processing. However, that is beyond the scope of this book, which focuses on how to make games with PHP. So how exactly do you do this?

Sessions and Session Variables

To keep state you need to utilize session variables. Every time you hit a Web site you start a session, and this session is identified by a unique GUID (Globally Unique Identifier). A session is defined as the period of time during which a unique user interacts with a Web application. As the programmer, you can store variables in this session. This is an extremely useful tool. You don't have to keep track of each individual user and his state; you just have to reference the session. I'll discuss how to reference sessions later on.

When working with sessions there are some important things to remember. If you have a server farm running, a user's session does not follow him as he moves from server to server. In other words, a session is only valid on a single server. The session is process dependent. This means that if your Web server needs to be restarted and there are current sessions active, then the sessions will be lost. One unique feature of sessions in PHP is that they are not cookie dependent. So if a client does not accept HTTP cookies, the client can still take advantage of sessions.

There are currently two methods supported for passing sessions in PHP4:

- Cookies
- URL Parameter

Cookies are of course the preferred method, but since they are not always available you can pass the session id along the query string as a URL parameter. Here is an example of how you might pass the session id along the query string in PHP.

```
<?php
if(!session_is_registered('nCount'))
{
    session_register('nCount');
    $nCount = 1;
}
else
{
    $nCount++;
}
?>

<p>Hello, you have seen this page <?php echo $nCount; ?> times.</p>

<p>To continue, <a href="somepage.php?<?php echo strip_tags(SID)?>">click here (somepage.php?<?php
echo strip_tags(SID)?>)</A></p>
```

In this example we check to see if the session variable `nCount` is registered. If it is not registered we register it and initialize it. Otherwise we add 1 to the count and display our message to the user. The output of this page might look something like you see in Figure 1.2.

Note

Non-relative URLs are assumed to point to external sites and don't append the sid as it would be a security risk.

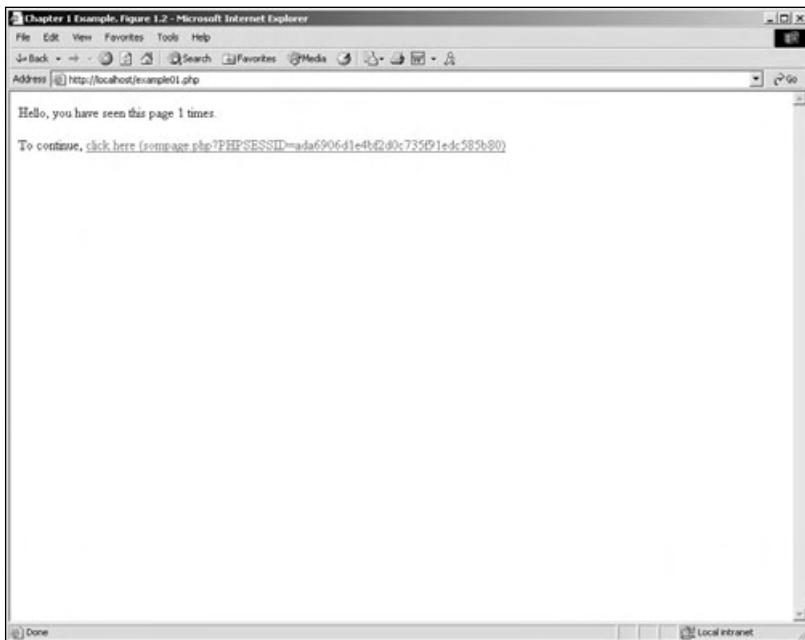


Figure 1.2 An example output of passing a session id.

That long number at the end of the link, `ada6906d1e4bf2d0c753f91edc585b80`, is the session id. PHP stores these sessions in the `sessiondata` folder, usually `C:\PHP\sessiondata`. If you look in this folder you will see files that start with `sess_` and end with a long GUID like the one on the end of the URL. This is how PHP itself keeps track of the session data. When you start debugging you will probably reference this folder quite often because you can open up these files and see if the session variables you set are being stored. If they are not being stored then you will need to check your `php.ini` file to make sure sessions are enabled.

Now that you have a general knowledge of how sessions in PHP work, take a look at the configuration options available to you and what each one of them means. There are 21 configurable options for sessions in PHP (see Table 1.1) and each one of these behaviors is configurable in the `php.ini` file.

All of these options, except `user_trans_sid`, can be set anywhere using the `ini_set()` function.

Table 1.1 Session Configuration Options

Option Name	Default Value
save_path	"/tmp"
name	"PHPSESSID"
save_handler	"files"
auto_start	0
serialize_handler	"php"
gc_probability	1
gc_dividend	100
gc_maxlifetime	1440
cookie_path	"/"
cookie_domain	" "
cookie_secure	" "
use_cookies	1
use_only_cookies	0
cookie_lifetime	0
referer_check	" "
entropy_file	" "
entropy_length	0
cache_limiter	"nocache"
cache_expire	180
user_trans_sid	0
url_rewriter.tags	"a=href,area=href,frame=src,input=src,form=fakeentry"

Here is an overview of each of these configurable session options.

session.save_handler

`session.save_handler` defines the name of the handler that is used for the sessions data management.

session.save_path

`session.save_path` This is where the sessions are stored if `session.save_handler` is set to `files`. There is an optional `N` argument to this option that determines the number of directory levels that your sessions will be stored across. For example, setting the `save_path` option to `5;./temp`, where `5` is the optional argument, will result in a directory structure something like this:

```
./tmp/5/c/4/e/1/sess_ ada6906d1e4bf2d0c753f91edc585b80.
```

If the `N` argument is used you must create the directory structure yourself. Also, if you do use the `N` argument you must surround the `session.save_path` in quotes because the semicolon is also used for comments in the `php.ini` file. For our purposes, I do not recommend using this option.

session.name

`session.name` specifies the name of the session that is used to reference the `sessionid` in cookies. This should only take alphanumeric characters.

session.auto_start

`session.auto_start` specifies if the session module should start automatically. If you do use `session.auto_start` then you cannot put objects into your sessions because the class definition must be loaded before starting the session.

session.serialize_handler

`session.serialize_handler` is the name of the handler which is used to serialize or deserialize data in the session. Currently only the internal formats “`php`” and “`WDDX`” are valid. `WDDX` is also available only if PHP is compiled with `WDDX` support.

session.gc_probability and **session.gc_dividend**

`session.gc_probability` specifies the probability when the garbage collection process should start. This is calculated by using `gc_probability/gc_dividend`.

session.gc_maxlifetime

`session.gc_maxlifetime` is the number of seconds at which the session data is seen as garbage and cleaned up.

session.cookie_path

`session.cookie_path` is the path for which the cookie is valid.

session.cookie_domain

`session.cookie_domain` is the domain for which the cookie is valid.

session.cookie_secure

`session.cookie_secure` tells the cookies whether they should be sent over secure connections only.

session.use_cookies

`session.use_cookies` specifies if the session module should store session ids on the client side.

session.use_only_cookies

session.use_only_cookies tells the session module that it should *only* use cookies to store the session ids. This prevents attacks from session variables being passed along the URL.

session.cookie_lifetime

session.cookie_lifetime specifies the lifetime of the cookie in seconds. The default value 0 means wait until the browser is closed to expire the cookie.

session.referer_check

session.referer_check contains a string for which you want to check each HTTP referer. If PHP does not find this substring, the embedded session id will be marked as invalid.

session.entropy_file

session.entropy_file gives a path to an external file that will be used as an addition in creating the session id. The external file will contribute to the randomness in the session creation.

session.entropy_length

session.entropy_length specifies the number of bytes which will be read from the session.entropy_file variable. When set to its default value of 0, it disables the entropy options.

session.cache_limiter

session.cache_limiter specifies the cache control method to use. Valid options for this are none, nocache, private, private_no_expire, and public.

session.cache_expire

session.cache_expire is the time to live for cached session pages in minutes. This option has no effect if session.cache_limiter is set to nocache.

session.user_trans_sid

session.user_trans_sid specifies whether or not transparent sid support is enabled.

url_rewriter.tags

url_rewriter.tags specifies what HTML tags should be rewritten to include the session id if session.user_trans_sid is set to 1.

Wow, there are a lot of configuration options for sessions. PHP is very unique and customizable. You do not have to specify all of these options in the php.ini file; as a matter of fact, PHP gives you several functions that expose these options to your code. So, if you need to change any of these in particular applications you can call on the functions listed in Table 1.2.

Table 1.2 Session Functions

Function Name	Arguments	Description
session_cache_expire	[int new_cache_expire]	returns the current setting for session.cache_expire. If new_cache_expire is specified, the current session.cache_expire is replaced with new_cache_expire.
session_cache_limiter	[string cache_limiter]	returns the name of the current setting for session.cache_limiter. If cache_limiter is specified, the name of session.cache_limiter is set to cache_limiter.
session_decode	string val	decodes the session data in the variable val.
session_encode	void	returns an encoded string with the contents of the current session.
session_destroy	void	destroys the current session and any data associated with the current session.
session_get_cookie_params	void	returns an array with the current sessions cookie information. The array contains the following items: lifetime, path, domain, and secure.
session_id	[string val]	returns the current session id. If val is specified it will replace the current session id. If val is specified you must call this before session_start().
session_is_registered	string val	returns true if there is a global variable with a name specified in val.
session_module_name	[string val]	returns the name of the current session module. If val is specified, that module will be used instead.
session_name	[string name]	returns the name of the current session. If name is specified it sets the name of the current session.
session_regenerate_id	void	will regenerate the session id and replace the current session id with the newly generated session id.
session_register	mixed name [, mixed]..	accepts a variable number of arguments; any of these arguments can be a string holding the name of a variable or an array consisting of variable names or other arrays. For each of the names specified, session_register() will register a global variable with that name in the current session.
session_save_path	[string path]	returns the path of the current directory where sessions are being stored. If path is specified, the current path is replaced with path.
session_set_cookie_params	int lifetime [,string path] [,string domain] [, bool secure]	sets the cookie parameters, lifetime, path, domain, and secure.

Table 1.2 Session Functions (*continued*)

Function Name	Arguments	Description
session_set_save_handler	string open, string close, string read, string write, string destroy, string gc	sets the session storage functions used for storing and retrieving data. You would use this when you wanted to create your own session handling functions.
session_start	void	creates a session or resumes the current session. This is all based on the current session id.
session_unregister	string var	unregisters global variables with the name specified in var.
session_unset	void	this frees all session variables currently registered.
session_write_close	void	this will end the current session and store the session data. You do not need to explicitly call this function.

I know that you are saying to yourself, “What the hell am I supposed to do with all of this?” To tell you the truth, you won’t use half of these functions for what you will be doing. Just because you won’t use them in the examples does not mean that you won’t use them in some of your games. As for the functions you will use, here is an example PHP script that sets some session variables. Do not worry about the syntax of all the code right now; I’ll cover that in Chapter 4. For now, just look at how I am using the functions listed in Table 1.2.

```
<?php
// use $_SESSION instead of session_register due to security issues
session_start();
$turn = $_SESSION['turn'];
if(!isset($turn))
{
    $turn = 1;
    $_SESSION['turn'] = $turn;
}

printf("Current turn value: " + $_SESSION['turn']);

unset($turn);
session_destroy();
?>
```

Now take a look and see what this small example is doing. First you start the session using `session_start()`. Then you attempt to get a variable named `turn` back from the session. If the variable `$turn` is not set then you have not yet actually started the session and you need

to initialize the variable. After you initialize the variable, it sets the session variable `$turn` to whatever is in the variable `$turn`. Then, the current value is printed, the variable is unset, and the session is destroyed.

Do not worry if you aren't comfortable with all of this yet. You will be using sessions in all the projects in this book, so it will make more sense when you see sessions being used in an actual context.

If you're feeling a bit overwhelmed this would probably be a good time to take a break. If you're ready for more, continue with the discussion of how the Web server talks to the client with TCP/IP. After that, you will install and configure an IIS (Internet Information Server) Web server and an Apache Web server.

TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is simply a communications protocol that computers use to talk to each other. What do the different components of TCP/IP do?

- **TCP.** This is responsible for verifying the correct delivery of data from a client to a server, or a server to a client. TCP has built-in error checking and will trigger transmissions until the data has successfully reached its destination.
- **IP.** This is responsible for moving packets of data from node to node. IP forwards these packets based on a four-byte destination address (IP Address).

Just like every other communications protocol, TCP/IP is constructed of layers; however, each of the layers in TCP/IP is logical instead of physical. All network traffic moves through these layers as they travel through the hardware. Here are the layers and a little bit of information about each.

- **Application Layer.** This layer handles everything that is not TCP/IP. The Web browser, in our case, is the application layer.
- **Transport Layer.** This layer handles all the routing and delivery of data. This layer also includes the error control and sequence checking.
- **Internet Layer.** This is the layer responsible for data addressing, transmission, packet fragmentation, and packet reassembly.
- **Network Access Layer.** This layer handles the transmission of the data across the network, including determining how to access the hardware.

How does all this work? Before sending data across the network, TCP/IP establishes a connection with the destination machine by sending what we call *management packets*. After TCP/IP establishes this connection, data can be sent freely. Once the application that was using TCP/IP to send data is finished transmitting, it sends more management packets to the client machine to tell it to close the connection. TCP/IP also controls the flow rate of

the data to achieve the maximum data exchange rate, while avoiding congestion and packet loss. TCP/IP also attempts to pack as much data into a packet as possible.

To sustain an acceptable data transfer rate while avoiding network overload, TCP/IP contains four algorithms: Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery.

The Slow Start algorithm avoids injecting multiple segments into the network during the transmission. The reason for this is if there are routers and slower links between the sender and the receiver you start to lose packets and you slow down your connection. So the Slow Start algorithm enables TCP/IP to only send data at the rate which the acknowledgments are returned.

The Congestion Avoidance algorithm does what the name says—it avoids congestion. Network congestion usually occurs when a large amount of data is sent out of a larger pipe and received on a smaller pipe. Congestion can also occur when multiple streams of data arrive at a router whose output capacity is less than the sum of the inputs.

The Fast Retransmit algorithm is triggered when the receiving end sends a duplicate acknowledgment. This occurs when an out-of-order segment is received, a duplicate ACK is received, or a packet is lost. When this occurs, TCP/IP will retransmit the packet that was excepted.

The Fast Recovery algorithm is triggered after the Fast Retransmit sends the missing packet segment. This allows TCP/IP to recover from the missing segment while still avoiding network congestion. If fast recovery was not available, the user might see a significant decrease in data transmission.

Now that you know how TCP/IP works, you are set with the knowledge you need to continue the march. The next step is to install the Web server. Following are installation walkthroughs for IIS and Apache.

Installing the IIS Web Server

IIS (Internet Information Server) is included with the Windows 2000 installation disk. PWS (Personal Web Server), a trimmed down version of IIS, is available with the Windows 98 installation disk. The differences between IIS and PWS are relatively small; the biggest differences are:

- PWS cannot host multiple Web sites on a single machine.
- PWS cannot log to an ODBC compliant database.
- PWS cannot restrict access by IP address.
- PWS cannot perform indexing.
- PWS cannot isolate processes.

Fortunately for us, none of this affects our development. Basically, PWS is not intended to be a production server. So if you want to run your own Web sites beyond just development you will need to install IIS.

Caution

IIS/PWS is not supported on Windows ME or Windows XP Home Edition, so you will need to install Apache Web Server for Windows.

Before you install IIS, if there are any other Web servers running on your machine you will need to uninstall them. If IIS or PWS is already installed on your machine, please skip to Chapter 2 to set up PHP.

Installation on Windows 2000/XP Professional

IIS/PWS, which I will just refer to as IIS from here on, is a Windows component, so you will need to have your Windows installation disk ready. To start the installation, click on Start>Settings>Control Panel, and double-click on the Add/Remove Programs icon as shown in Figure 1.3.

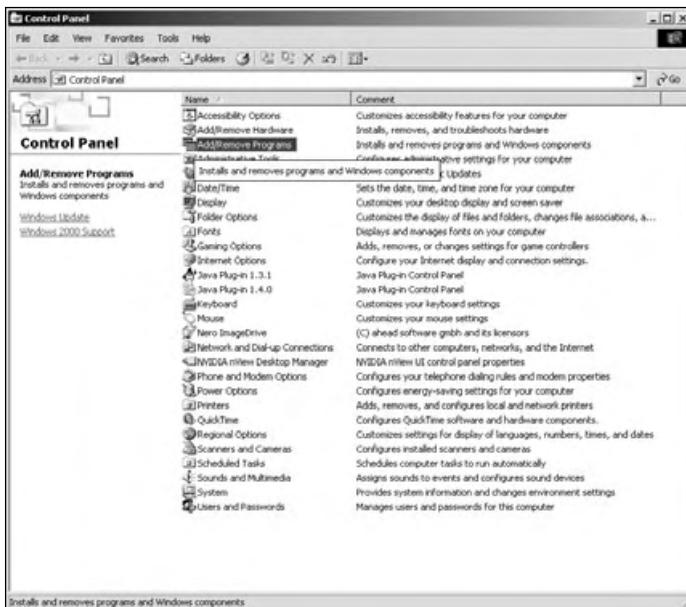


Figure 1.3 The Windows Control Panel.

Caution

Your Control Panel may differ from that in Figure 1.3.

Now click on the Windows Component Tab (if you are using Windows XP click on the Add/Remove Windows Components button) on the left-hand side and scroll down until you see Internet Information Services (IIS). Click on Internet Information Services (IIS), then click the Details button in the lower right corner of the window. You should see a screen that looks similar to Figure 1.4.

Now you can select the components you want to install with IIS. By default, Common Files, Internet Information Services Snap-In, Personal Web Manager, and World Wide Web Server should be selected. On Windows XP you will not see the Personal Web Manager option. Also, the World Wide Web Server is actually World Wide Web Services on Windows XP. If these are not selected then you should select them and click OK.

To install, click the Next button. After Windows is finished installing IIS to C:\Inetpub, there will be several new items on your computer. In C:\Inetpub there are five folders: AdminScripts, iissamples, Scripts, webpub, and wwwroot. The wwwroot folder is where the default Web site for IIS lives. You will be working mostly with this folder and virtual directories.

Tip

A virtual directory can live anywhere on your computer, but it shows up under the Default Web site. This is extremely useful if you don't want to keep your Web files under the C:\Inetpub directory.

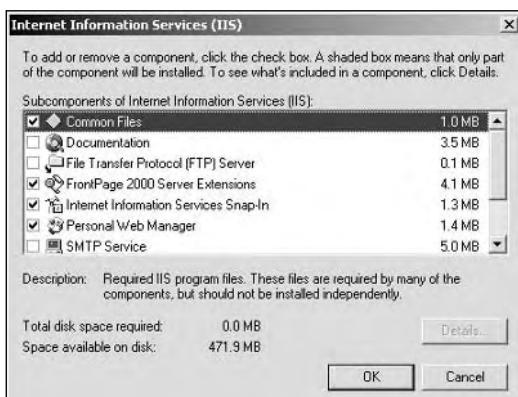


Figure 1.4 The Windows Component tab.

That's it! IIS is now installed on your machine. With this install you can run HTML and ASP pages. Now it is time to add the PHP CGI Interpreter to your IIS installation. You can either read on to see how this process is completed for Windows 98/ME and the Apache Web Server, or you may just skip straight to Chapter 2 to learn how to install PHP.

Installation on Windows 98

To install PWS on Windows 98 you will need to insert your installation disk and navigate to the add-ons\pws folder. Once you reach the add-ons\pws folder, double-click on the setup.exe icon. This will launch the PWS installer. You will be presented with a screen with three buttons: Minimum, Typical, and Custom. Click on Custom. You should see a screen that looks like Figure 1.5.

Once you get to this screen make sure the following are checked:

- Common Program Files
- FrontPage 98 Server Extensions
- Microsoft Data Access Components
- Personal Web Server (PWS)
- Transaction Server

Once you have selected these options, click the Next button and make sure the installation directory for the WWW service is set to C:\Inetpub\wwwroot, as shown in Figure 1.6.

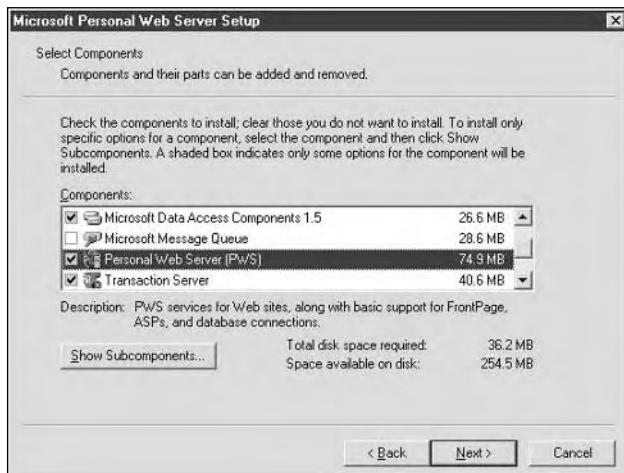


Figure 1.5 The PWS installation—Custom Installation screen.



Figure 1.6 The PWS installation—WWW Service screen.

Click Next to finish the installation. Once the installation is complete you can skip to Chapter 2 to install the PHP CGI Interpreter.

Installing the Apache Web Server on Windows ME/XP

Installing Apache on Windows ME or Windows XP Home Edition is very easy. I have included the Apache 2.0.46 msi installer on the CD in the Apache directory. If a newer version is available at the time you read this, you can download it at www.apache.org.

To install Apache, double-click on the apache_2.0.46-win32-x86-no_src.msi in the Apache directory on the CD included with this book. Once the installation starts it will ask you for a few things.

- **Network Domain.** This is where you enter the DNS domain for which your server will be registered. Just type in `mydomain.com`.
- **Server Name.** This is your server's fully qualified domain name. Type in `server.mydomain.com`.
- **Administrator's E-Mail Address.** Just enter your e-mail address here.
- **For Whom to Install Apache.** Select for All Users, on Port 80, and as a Service.

Your screen should now look like Figure 1.7.

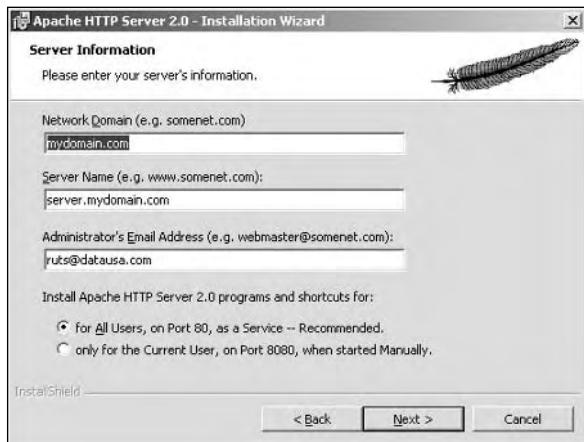


Figure 1.7 The Apache MSI installer.

Once you have all that information entered, click the Next button. Select Typical Install, and then click Next. Click Next one more time to accept the default directory, C:\Program Files\Apache Group. The installation will create a subdirectory in the default directory named Apache2—this is where all the files for Apache will live.

That's it! Wasn't that easy? Now you can move on to Chapter 2 and install the PHP CGI Interpreter.

Installing the Apache Web Server on UNIX

The first thing you need to do is get the source code for the Apache server. I have included version 2.0.46 on the CD in the Apache directory. There might be a newer version out when you read this book—if you would like to check, you can download the newer version from www.apache.org. There are a two basic requirements that you will need to install Apache.

- **Disk Space.** You need at least 50 megabytes of temporary free disk space available. After Apache is installed, it occupies 10 megabytes.
- **Compiler.** You must have the ANSI-C compiler installed. You can get the GNU C Compiler (GCC) for free from www.gnu.org/software/gcc/gcc.html.

Now that you know the requirements and have the Apache source files, you need to extract them from the tarball. To do this, type the following at the command prompt:

```
$ gzip -d httpd-2.0.46.tar.gz  
$ tar xvf httpd-2.0.46.tar
```

This will create a new directory under your present working directory containing the source code of the distribution. You will need to cd into that directory before compiling the server.

```
$ cd httpd-2.0.46
```

Now you need to configure the Apache source tree. This is done by using the configure script that is in the root directory of the distribution that you just extracted. Before you run this script there are several environment variables and flags that the configure script can take. The environment variables are listed in Table 1.3 and always appear before the configure command.

Depending on the setup of your computer, you will need to define some of these environment variables. An example of the configure command might look something like this:

```
$ CC="gcc" ./configure --prefix=/sw/pkg/apache
```

For a complete list of the configuration options, run `./configure—help`. After configure has run, it will take several minutes to test for the availability of features on your machine and to finish building the Makefiles.

After the configure script has finished running you can build the distribution by typing in `make` at the command prompt. This will create the installation files. Be patient with this process; it usually takes approximately 3 minutes to finish. It could be even longer depending on the number of modules you selected.

Table 1.3 Environment Variables

Variable Name Description

CC=	The name of the C compiler.
CPPFLAGS=	Miscellaneous C preprocessor compiler options.
CFLAGS=	Debugging and optimizations for the C Compiler.
LDFLAGS=	Options for the linker.
LIBS=	Library location information.
INCLUDES=	Header file search directories.
TARGET=	Name of the executable which will be built. The default is <code>httpd</code> .
SHLIB_PATH=	Options which specify shared library paths for the compiler and the linker.

As soon as the make script is finished, you can install the Apache Web Server package. To do this, simply type `make install` at the command prompt. After the install script has finished, Apache is installed and ready to go. There are some customizations that you can do by editing the `httpd.conf` file. There are tons of options in this file—far too many to go through here in this book. So, I direct you to the online Apache documentation at <http://httpd.apache.org/docs-2.0/>.

After you have finished customizing your Apache installation you should have to start the server. During the installation, Apache put a start script in your bin directory called `apachectl`. To start the server you simply have to type the following:

```
$ /apache/bin/apachectl start
```

You should now be able to request a Web page via a URL (`http://localhost/`). If you ever need to stop the server, you type:

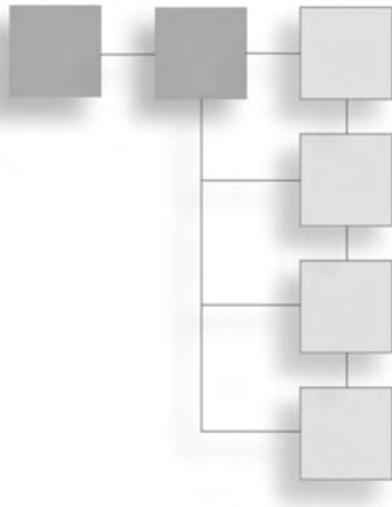
```
$ /apache/bin/apachectl stop
```

That's it! You should now have the Apache Web Server installed and running. You can now move on and install the PHP CGI Interpreter and get moving on PHP.

This page intentionally left blank

CHAPTER 2

WAGING THE CONFIGURATION WAR



- The Platforms
- Building and Installing PHP on UNIX
- Installation on Windows for IIS/Apache
- Testing Your Installation

PHP is a very versatile language and comes with a myriad of options. PHP supports several hundred APIs and interfaces to other programming tools. So it would be worthwhile to consider what you want PHP to do and install only the modules you need.

You will be configuring the PHP interpreter with a few extra modules, one of which is Boutell's GD library (GD SDK). The GD SDK allows you to generate and manipulate images on the fly, which I consider to be very important in the development of games. If you would like to install other modules, feel free. The majority of the installs are very similar.

Let's get to it!

The Platforms

Since PHP is not OS dependent we will be installing and configuring PHP for UNIX and Windows.

When you talk about UNIX, the Web server of choice is Apache. You can configure PHP in two different ways when you install on Apache.

- As a CGI Interpreter.
- As a Apache module.

Note

You can only configure PHP as an Apache module with the Apache server.

When PHP is compiled as an Apache module it runs in the same memory space as the Apache server. This means two things to you: 1) you get a performance boost when interpreting pages, but 2) you get screwed if a page goes haywire. If one of your PHP pages goes haywire when running in the same memory space as the Apache server, the Apache server will halt. This means you are dead in the water—no pages will be served up until it is restarted. However, if you configure PHP as a interpreter you avoid the whole issue altogether. The disadvantage is obviously a minor performance hit, but it is negligible.

On the other side of the coin is Microsoft Windows with IIS (Internet Information Server) or PWS (Personal Web Server). Distributions of PHP are available from IIS 3 up to IIS 5; I have included the IIS and Apache installs on the CD provided with the book.

Building and Installing PHP on UNIX

If you're going to install PHP on a Windows platform you can continue on to the next section, unless this topic is really of interest to you.

The installation on UNIX-like systems is fairly straightforward. You need to generate the make files by running the autoconf scripts and then carry out a make and install. I will only be covering how to install PHP as a CGI Interpreter for simplicity's sake.

Let's get started. The first step is to uncompress the distribution and extract the files.

```
$ gzip -cd php-4.2.3.tar.gz | tar xvf -
```

Now you must change to the directory where the distribution was unpacked and run the configure script to generate the necessary make files. You can pass other options to tell the configure script what modules you would like to install. For example:

```
$ ./configure --with-gd=/usr/src/gd-2.0.4 --enable-gd-native-ttf --enable-gd-imgstrttf --with-jpeg-dir=/usr --with-png-dir=/usr
```

The `--with-gd` option is an example of how you can specify at build-time what modules you would like to install. Some of these extras or add-on modules are detected automatically; other add-ons you may need to specify specifically in the configure script. If you have any doubt that the module won't be installed automatically you should specify it in the options. Refer to Table 2.1 for a list of the modules and their commands.

Table 2.1 Configuration Add-Ons

Feature or Add-On Module	Argument to Configure	Description
Checking for internal redirects	—enable-force-cgi-redirect	This checks to see if a given request was an internal redirect with respect to the server.
Configuration file location	—with-config-file-path=DIR	Specifies the location of the configuration file php4.ini.
Directory of executables	—with-exec-dir=DIR	When PHP runs in safe mode the executables are chosen only from the directory specified by DIR.
Escaping quotes from data	—enable-magic-quotes	Data put into a database may have quotes in it. Enabling this option escapes these quotes with a backslash.
GPC variables	—enable-track-vars	GPC—GET, POST variables from forms are sent to the server. If this is enabled the server will track these variables.
LDAP support	—with-ldap=DIR	Enables support for Light-Weight Directory Access Protocol.
Mcrypt support	—with-mcrypt=DIR	mcrypt encryption support.
Regular expression library	—with-system-regex	Allows PHP to use the underlying OS's regular expression library.
Remote include()	—enable-url-includes	This allows includes from HTTP and FTP locations.
Short tags	—disable-short-tags	PHP scripts are enclosed by <?php ?> tags or the short form <? ?>. This option will disable the latter.
Syntax highlighting	—disable-syntax-hl	Disables the default behavior of highlighting PHP syntax.
Warnings	—enable-maintainer-mode	Turns on compiler and dependency warnings.
XML support	—with-xml	Enables support for XML.
BC math functionality	—enable-bcmath	Adds support for Bench Calculator math functions.
Debugger	—enable-debugger	Enables support for PHP's internal debugger.
GD	—with-gd=DIR	Enables support for GD. DIR points to the directory for GD.

* This is a small list of the add-on options. For more, run configure—help.

Now if the earlier configure script ran successfully there should be a file called make in the directory. This is the file that will build the distribution. To run it, type the following:

```
$ make
```

If you see a link time failure, you are most likely missing a reference to a library. If a library is in a non-standard location you will need to specify the path to the library and re-run the make file again.

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/otherlibdir  
$ make
```

After all of the source files have been successfully compiled and linked you can install the distribution. To do this you must have root privileges. I am assuming that you are installing as root; if you are not you will need to assume the super user.

Tip

To assume the super user, or root user from a command line, simply type su at the command prompt and enter the password.

To install the distribution, simply enter the following command:

```
$ make install
```

That's it. You should now have PHP installed on your system. The only thing left to do is tell Apache what to do with PHP pages. Navigate to the folder where your Apache configuration file lives (usually httpd.conf), open it, and type the following lines:

```
AddType application/x-httpd-php4 .php4 .php  
AddType application/x-httpd-php4-source .phps  
Action application/x-httpd-php4 /cgi-bin/php  
DirectoryIndex index.html index.shtml index.cgi index.php4 index.php  
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl .php4 .php .phps
```

Finally, you will need to restart the Apache Web server. That's all that you need to do—you're done configuring PHP and Apache. Now would probably be a good time to take a break and get another Mountain Dew.... Next up: testing!

Installation on Windows for IIS/Apache

The folks at www.php.net have made the installation for Windows extremely simple. You can download the already compiled binaries, which I have included for you on the CD, or you can download the source and compile it yourself. I will not go over how to compile

the source on a Windows machine, but if you really would like to learn that, there is wonderful documentation at www.php.net.

To start the installation of PHP, double-click on the php-4.3.2-installer.exe file on the CD. This will bring up the installation wizard. Go ahead and click the Next button and agree to the license agreement. Make sure that the installation type is set to Standard and click the Next button. If the directory C:\PHP is where you want the installation, click Next; otherwise change the directory. Now you will be presented with a screen to enter your SMTP server and the from address for the mail function. Just leave these as they are (SMTP Server = localhost, and from address = me@localhost.com), and click Next.

You should see a screen that looks like Figure 2.1. This is where you will select the Web server that you installed in Chapter 1. If you installed IIS for Windows 2000/XP you will want to select Microsoft IIS 4 or higher. If you installed PWS for Windows 98 you will want to select PWS Microsoft Windows 9x. If you installed Apache for Windows ME you will want to select Apache.

After you have selected your Web server click Next to start the installation. If you are asked to select scriptmap nodes to add PHP mappings to, click on Select All and then click on OK.

After the installation is finished you can add the modules that you need.

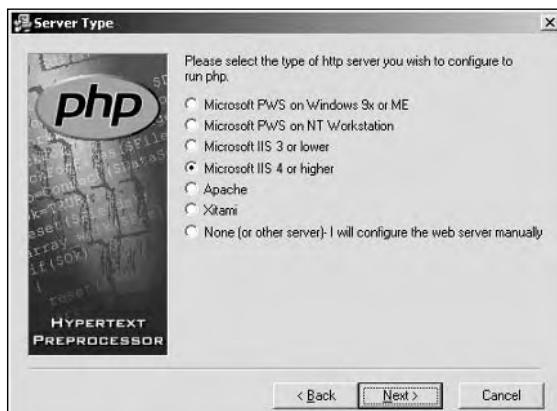


Figure 2.1 Select your Web server.

Table 2.2 Available PHP Extensions

Extension Name	Extension DLL	Extension Description
Bzip2	php_bz2.dll	Compression functions
Calendar	php_calendar.dll	Calendar functions
ClibPDF	php_cpdf.dll	Lets you create PDF files.
Crack	php_crack.dll	Allows you to test the strength of a password.
CType	php_ctype.dll	Character and string functions.
CURL	php_curl.dll	Client URL Library.
DataBase (dbm-style)	php_dba.dll	Functions for accessing Berkley DB style databases.
DBase	php_dbase.dll	Functions for accessing dbase style databases.
DBX	php_dbx.dll	Allows you to access all supported databases.
DOM XML	php_domxml.dll	Allows you to operate on an XML document.
Read EXIF	php_exif.dll	Reads EXIF headers from a JPEG file.
FDF	php_fdf.dll	Allows you to handle forms in a PDF document.
GD	php_gd2.dll	Boutell's GD SDK. Allows image manipulation.
Get Text	php_gettext.dll	Allows you to internationalize your PHP applications.
Hyperwave	php_hyperwave.dll	Access to Hyperwave functions.
ICONV	php_iconv.dll	Allows you to convert strings between various character sets.
IIS Management	php_iisfunc.dll	Allows management of IIS through code.
IMAP	php_imap.dll	IMAP, POP3, and NNTP functions.
InterBase	php_interbase.dll	Allows access to the InterBase database.
Java	php_java.dll	Integrates Java support into PHP.
LDAP	php_ldap.dll	Lightweight Directory Access Protocol.
Multi-Byte String	php_mbstring.dll	Handles Japanese character sets.
Mhash	php_mhash.dll	Exposes some functions used for hashing.
Ming	php_ming.dll	Allows you to create Flash movies on the fly.
mSQL	php_mSQL.dll	Allows access to the mSQL database.
MSSQL	php_mssql.dll	Allows access to the MS SQL database.
Oracle	php_oracle.dll	Allows access to the Oracle database.
Open SSL	php_openssl.dll	Exposes OpenSSL functions for encrypting/decrypting.
PDF	php_pdf.dll	More PDF functions.
Printer	php_printer.dll	Exposes functions for a printer.
Sockets	php_sockets.dll	Access to sockets.
W32API	php_w32api.dll	Access to the Win32 API from PHP.
XML-RPC	php_xmlrpc.dll	Allows you to write XML-RPC servers and clients.
XSLT	php_xslt.dll	Exposes functions for working with XSLT.
YAZ	php_yaz.dll	Allows PHP to interface with the YAZ toolkit.
Zip	php_zip.dll	Read-only access to zip files.
ZLib	php_zlib.dll	Allows you to read and write gzip files.

* More information on these extensions can be found at www.php.net/manual/en/install.windows.php

Installing the Windows Extensions

Now that you have PHP installed you can add extensions to the installation. Table 2.2 provides a list of the extensions available to Windows.

Adding an extension is very easy—all you need to do is copy the dlls you want and edit the php.ini file. First create a directory called extensions in the folder where you installed PHP (usually C:\PHP). Now extract the php-4.3.2-Win32.zip file that is included on the CD. Go to the directory that the zip file extracted to and copy all of the dlls out of the dlls directory and paste them into C:\WINNT\system32 if you are on a Windows 2000 machine; C:\WINDOWS\system32 if on a Windows XP machine; or if you are on a Windows 98 machine, C:\WINDOWS\system. Now go back to the folder where the zip file extracted to and copy all the dlls out of the extensions folder and paste them into the extensions folder you created.

Now that you have the dlls in their proper places, open up your php.ini file. It should be located in the C:\<WINDOWS DIRECTORY>\php.ini. Once you have that file open, go to the “Paths and Directories” section and look for extension_dir = “./”. We want to set this to C:\PHP\extensions\. Now go to the “Dynamic Extensions” section and uncomment the following lines:

1. extension=php_dbase.dll
2. extension=php_gd2.dll
3. extension=php_sockets.dll

Note

It is important that you put the trailing backslash on C:\PHP\extensions\

Go ahead and save your php.ini file and exit. You now have installed the extensions and the PHP interpreter. All that's left to do is to test the installation to make sure it is parsing PHP pages.

Testing Your Installation

PHP should now be installed and working on your machine. Go ahead and test it out. Open up your favorite text editor and type the following:

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

Then save this file as test.php and move it to a place where you will be able to hit it. If you are on Windows you can put it in C:\Inetpub\wwwroot\test.php; if you are in UNIX you can put it in /usr/web/test/php.

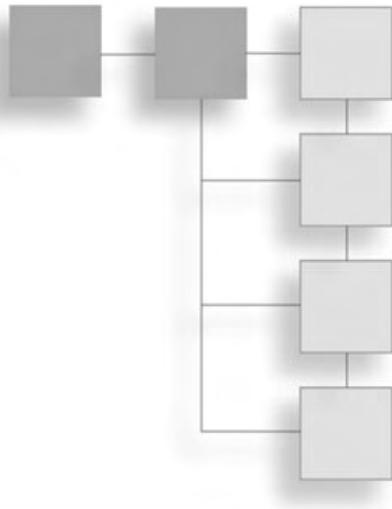
Open up a Web browser and type in <http://localhost/test.php> and you should see something similar to Figure 2.2.



Figure 2.2 Test results of your new PHP installation.

CHAPTER 3

I HAVE CONQUERED THE SERVER, LET ME AT THE CODI



- The Basics of the HTML Tag
- The Almighty HTML Document
- The HTML Body
- Graphics and HTML
- Tables
- Creating Forms for Input

Congratulations; if you are here then you have successfully configured and set up the Web server and the PHP CGI interpreter. In this chapter I will cover HTML (Hyper Text Markup Language). HTML is very important to us, because without it we would not be able to display anything to our user. That would not be a very fun game would it?

If you feel comfortable with HTML you can skip ahead to Chapter 4 and start in on PHP. However, if you feel like you need a refresher on HTML, grab some snacks, crack open a Mountain Dew, then open up your favorite text editor and we will get started.

The Basics of the HTML Tag

HTML tags are straightforward to understand. You have a beginning tag and an ending tag. In other words, every tag has two elements that are paired together. The beginning tag looks like `<tag_name>` and the ending tag looks like `</tag_name>`. The only difference between these two tags is the forward slash in the end tag. Almost every tag in HTML

has attributes. To use these attributes your tag would look like this: `<tag_name attribute1="someValue"></tag_name>`. The only tags in HTML that don't have attributes you can set in the tag are `<HTML></HTML>`, `<HEAD></HEAD>`, and `<TITLE></TITLE>`. All other tags have one or more attributes that you can apply to them.

Now that you know the basics of an HTML tag, take a look at the different elements of an HTML document and how it is structured.

The Almighty HTML Document

An HTML document usually has an extension of .htm or .html; however, since you will be coding in PHP in the near future you should just get in the habit of saving your HTML documents with the extension .php. To begin, please refer to Table 3.1, which contains the four basic tags that make up the beginnings of a HTML document.

As you may guess, everything that goes into your HTML document will be surrounded by the `<HTML>` tags and anything you want to display will go in between the `<BODY>` tags. The `<HTML>` tags are important; even though the majority of browsers out there can handle a document without the HTML tags, there is still a chance that your page will be misinterpreted. It is just safer to include them.

Note

Every tag starts out with `<name_of_tag>` and ends with `</name_of_tag>`.

The head section of your HTML document (`<HEAD></HEAD>`) is sort of like a quick reference for browsers. It supplies the document title, contains META data, and establishes relationships between HTML documents and file directories. For the most part, you will not be dealing with all of these in your application. The only tag you will really need to worry about is the `<TITLE></TITLE>` tag. All this tag does is give your page a title and display it in the title bar of the window, like in Figure 3.1.

Table 3.1 Tags for the Beginning

Tag	Description
<code><HTML></HTML></code>	Creates a HTML document.
<code><HEAD></HEAD></code>	Contains the document title and other information that is not displayed on the Web.
<code><TITLE></TITLE></code>	Sets the title of the document.
<code><BODY></BODY></code>	Contains the visible area of the document.

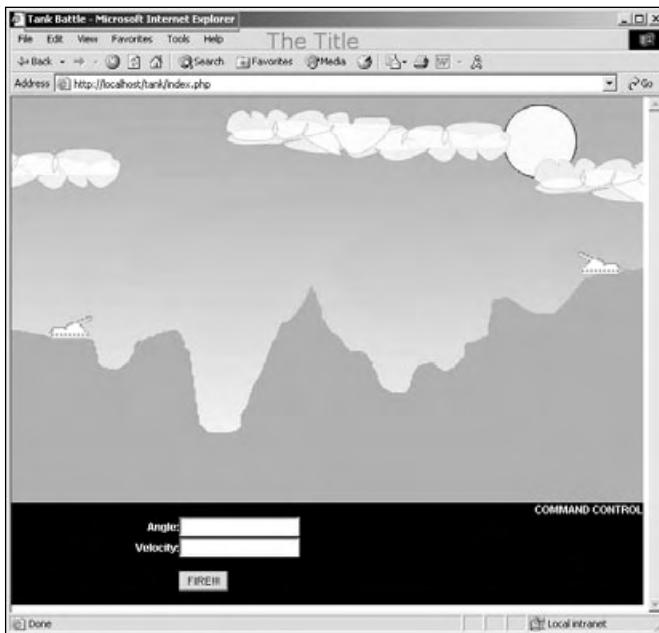


Figure 3.1 An example title tag.

I strongly suggest you give your pages logical names to relieve confusion when you edit them later; don't use page names like "Page 1". Similarly, do not make your titles extremely long. A good rule of thumb is, if your title is longer than three to five words, then it is too long.

Let's go ahead and move on to the nitty gritty of HTML. If you are not quite clear on how all of these tags fit together, don't worry—I will show an example soon.

The HTML Body

Table 3.2 shows the BODY tag attributes.

Everything that displays in your HTML page happens inside the `<BODY></BODY>` tags. When I say "everything" I mean anything that is visible to the user. That includes images, text, tables, links, background colors, text colors, link colors, forms, and buttons.

The majority of what we will be using in our games deal with table layouts, forms, images, and some text formatting. There will not be that much text formatting because the majority of our games will be images and forms, but you might want to create a help section for your game that looks professional, so take a look at the available text formatting tags.

Table 3.2 BODY Tag Attributes

Attribute	Values	Description
Bgcolor	#RRGGBB	Sets the color to the specified HEX value.
Text	#RRGGBB	Sets the text to the specified color.
Link	#RRGGBB	Sets the link to the specified color.
Vlink	#RRGGBB	Sets the visited states of links to the specified color.
Alink	#RRGGBB	Sets the color of the link when a user is clicking on it.
Background	URL to image	Sets the background of the document to a specified image.
Topmargin	# of pixels	Sets the value of the top margin of the document in Internet Explorer.
Leftmargin	# of pixels	Sets the value of the left margin of the document in Internet Explorer.
Marginheight	# of pixels	Sets the value of the top margin of the document in Netscape.
Marginwidth	# of pixels	Sets the value of the left margin of the document in Netscape.

* A complete list of tags is included in Appendix A.

Most of the time you will just want to make the text stand out a little more, maybe for a header or something. For instance, you might want to label an input field or just make paragraphs to give instructions. So, we will be using the `<H3></H3>`, ``, `<P></P>`, and `
` tags the majority of the time. Take a look at an example of an HTML document with these elements in it.

```

<HTML>
<HEAD>
    <TITLE>Code Example</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">
    <H3>Welcome!</H3>
    <p><b>Instructions</b><br>
    To master the art of PHP game programming you need to be fluent in HTML. Otherwise no
    one will be able to see your game.</p>
</BODY>
</HTML>

```

Table 3.3 Text Formatting Tags

Tag	Description
<pre></pre>	Creates a preformatted text block.
<h1></h1>	Creates the largest headline that HTML provides. There are six heading sizes available: <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5>, <h6></h6>. <h6></h6> creates the smallest heading size.
	Bolds text.
<i></i>	Italicizes text.
<u></u>	Underlines text.
<tt></tt>	Creates Teletype or typewriter-style text.
	Bolds text.
<p></p>	Creates a paragraph.
 	Inserts a line break.
<blockquote></blockquote>	Indents text from both sides.
<dl></dl>	Creates a definition list.
<dt></dt>	Creates a definition term.
<dd></dd>	Creates a definition.
	Creates a numbered list.
	Creates a bulleted list.
	Precedes each list item and adds a number or a bullet depending on the type of list.
<div></div>	This is used to format large blocks of HTML. It is usually used in conjunction with a style sheet.

* A complete list of tags is included in Appendix A.

Take a look at the code. Notice that the whole document lives between the <HTML></HTML> tags. As we move down to our next line we start the head of the document by placing the beginning <HEAD> tag. Since our title goes in the head of the document we place our <TITLE></TITLE> tag, with our title, and close out our </HEAD> tag.

Are you noticing a pattern yet? Let me reiterate: everything is always going in between tags.

Next, you start the <BODY> of the document. Notice the attribute BGCOLOR="#ffffff". All this does is set the background color of the document to white. For the rest of the attributes associated with the <BODY></BODY> tag refer to Table 3.2. Next you will use one of the heading tags. I usually use <H3></H3> for a heading tag because it does not seem too large or too small. Again, the text that I want displayed goes in between the opening and closing tags. Now you will start a new paragraph. Bold the first word, return to the next line, and insert the text.

Tip

A
 tag in HTML is exactly the same thing as putting in a \n or \n\r in C/C++.

Take a look at the results, which should look similar to Figure 3.2.

See, how easy was that?! Okay, now that you can dominate any text that might come your way let's add some graphics to it and make it look pretty.

Graphics and HTML

There are currently three types of image formats that HTML supports: .jpg, .gif, and .png. Each format has its own specifications and nuances. There are, of course, advantages and disadvantages to using each. Let's take a look at these formats and see their capabilities.

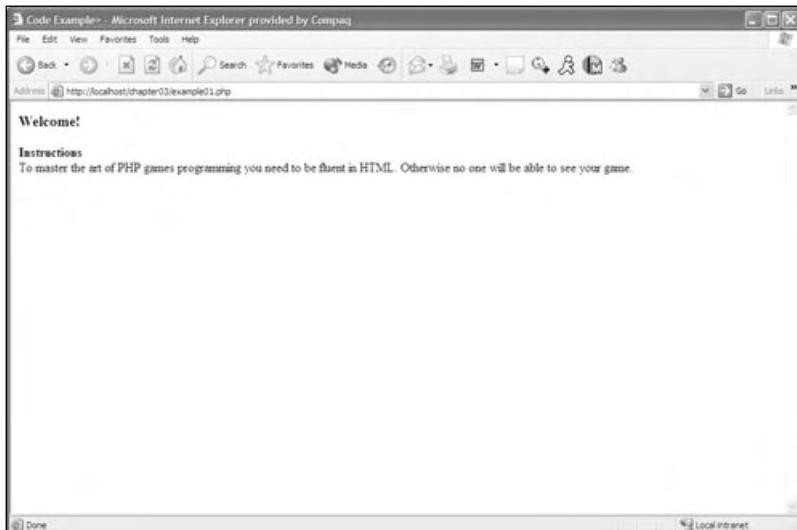


Figure 3.2 The results from the example HTML code.

The File Formats

The .jpg file format, pronounced “jay-peg”, is best used when you want to display photographs because it supports a full 24-bit color palette. It is a “lossy” compression algorithm, meaning that the algorithm that compresses the color information crunches the files in size but the image quality takes a hit. Most of the time you cannot see the difference in image quality, unless you’re printing the graphic on a professional image setter, which we aren’t going to do.

The .gif (Graphics Interchange Format) file format is designed to handle a 256-color palette. When a .gif compresses an image it uses a “lossless” compression algorithm. This means that the compression makes the file smaller in such a way that the original image can be reconstructed bit for bit. To do this it remaps the colors in an image, using what is called a Color Look Up Table (CLUT) to new colors in a 256-color palette. Then the image is compressed using Run Length Encoding (RLE). RLE is not an efficient way to compress images if there are many changes in color across a line of pixels, but it does a bang-up job when a row of pixels has many pixels of the same color.

.gifs also support transparency. You do this by defining one of the colors in the CLUT to be a transparent color. Then any pixels in the image that are mapped to that particular color become transparent. This is very handy for setting images on top of other images or having a single image that can be shown with several different backgrounds. Transparencies also allow you to make odd-shaped images, such as arrows or circles, that don’t show a rectangular frame.

Not only does .gif support transparencies, but it also supports multiple images. Meaning that you can have frames and multiple images compressed into the same file. This means that you can create little animated images.

Tip

When choosing a color for a transparency, use bright purple, bright pink, or bright green since these colors normally don’t show up in an image.

The final supported format for images is .png (Portable Network Graphics), pronounced “ping.” This format was developed to replace the older and simpler .gif format. .png has four main advantages over the .gif format: alpha channel support, gamma-correction, better interlacing, and better compression. However, .png does not support animated frames, so if you want an animated image you must use .gif. .png was developed to be a single-image format only.

.png’s compression is superb because it supports true color, grayscale, and palette-based images. .jpg supports the first two, and .gif supports the third. .pngs usually get 5% to 25% better compression than .gif files.

Caution

If your .png files are saved using the true-color mode, they will often be larger than expected. Make sure you use the palette-based mode when saving simple images.

.png supports transparencies like .gif does, but it does this in a different way. Instead of specifying a color in the CLUT, you specify an alpha channel or a mask channel. All three .png image types (true-color, grayscale, and palette) can store an alpha channel. So what does this mean? Well, where .gif stores a 3-bit RGB value for the transparency color for every pixel, .png stores a 4-bit RGBA value for every pixel. You may be wondering why this is any different from the .gif transparency. When a .gif is placed over a background and it has transparency values, you might see some jagged edges around the image where it meets the background. But when a transparent .png file is placed over a background, no artifacting occurs. A great example of this is when you have an image with a drop shadow, like in Figure 3.3.

Now that you see the difference between the images I know you are asking yourself, “Okay it looks better, but he said it stored an extra bit of information. Won’t this make the file size bigger?” And the answer is yes, but only if you use the alpha channel in an excessive way. So, if you have a .png alpha palette and you want four levels of transparency, you have to use four different palette entries. But since for the majority of cases you’ll need only one level of transparency, this won’t be a problem and the .png will always come out to a comparable size.

Now that all of these graphic formats are available to you, how will you know which one to use? The answer is very simple: use .jpgs if you are displaying something with a full color palette that has many changing colors. Use .gif if you want to do a small animation or transparency. Use .png if you don’t want to use .gif, since you can get the same results with .pngs as you can with .gifs.

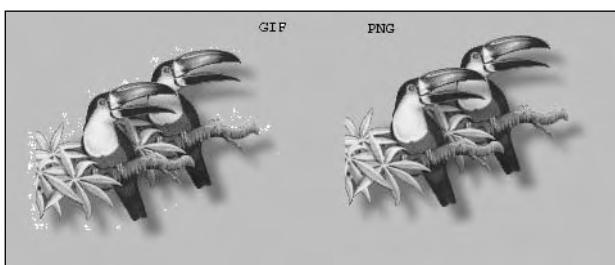


Figure 3.3 .gif versus .png transparency.

Using the File Formats in Your Document

All hail the image tag. Yes, that's right, the image tag. It allows you to display images on your page. What does it look like? Here is a breakdown of the image tag:

```
<img>
```

accessKey – Sets or retrieves the accessibility key.

align – Sets the alignment of the image, possible values are absBottom, absMiddle, baseline, bottom, left, middle, right, texttop, top.

alt – Alternate text of the image. This will be displayed if the image is not shown or if someone has their browser set to not show images.

border – The amount of border, in pixels, that you want around your image.

id – Sets the ID of the image.

height – Sets the height of the image.

name – Sets the name of the image.

src – This is the location and source of the image.

usemap – Sets the URL to use as a client-side image map.

width – Sets the width of the image.

The real heart of the tag is the src attribute. SRC is the source of the image. It can be a relative path or an absolute path. A relative path is relative to where you are in your directory structure. If you specify an absolute path it doesn't matter where you are in the directory structure. For instance, a relative path would look something like this: ``, where an absolute path can look like this: ``, or like this: ``. You will be using all relative paths for two reasons: 1) it makes your application more scalable, and 2) it makes your application self contained. If you were to use relative paths and the image on somewhere.com disappeared, you would now have a broken image. But, if you keep everything within your application you won't have this problem. The only way you will get a broken image is if you don't reference it correctly.

You should become very familiar with the image tag. You will be using it quite often, and knowing how to tweak the attributes of the tag will give you superb control over the appearance of the page. Take a look at an example using the image tag.

```
<html>
  <head>
    <title>Image Example</title>
  </head>
  <body bgcolor="#ffffff">
    
```

Here is an image with the image aligned to the right of the text.

```

```

Here is an image without the align tag in it at all.

```

```

Here is the same image, but really big.

```

```

Here is the same image, but really small.

```
</body>
```

```
</html>
```

The image tag is very easy to understand and use. In the code example, notice that the image tag has no ending tag—this is another special case in HTML. We basically have four images displayed on the screen; one is aligned to the right of some text. This was done by using the align attribute. You might want to play with the align attribute a bit to see what each of the properties can do to your image. This is very handy if you want text to wrap around an image. The next image is just placed on the screen with no align attribute. This is how you will use the image tag most often. The next two images use the width and height attributes. These allow you to change the width and height of an image. One is larger than the original image and one is a lot smaller than the original image. Take a look at Figure 3.4 for the results of our little code sample.

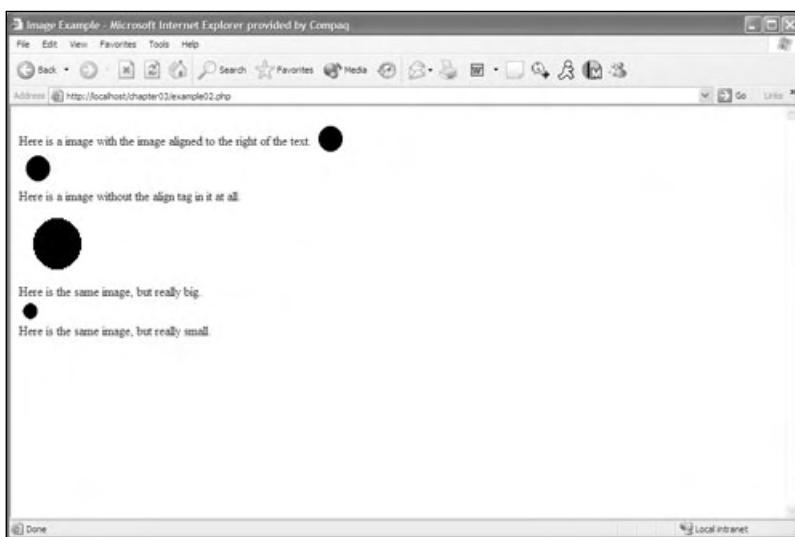


Figure 3.4 The image result from the code sample.

Now that you know almost everything there is to know about images, you can concentrate on laying out a framework for your games. To do this you must understand how tables work. I cannot stress enough how important this is for layout and for keeping your game together.

Tables

Believe it or not, tables are your best friends. You'll use tables to control where things end up on your page. You can make tables spread out and take up the whole page, no matter how big the browser window is, or you can make the table very rigid and occupy only part of the screen. The first step with tables is knowing what you can do with them.

Anything you can lay out in a grid should be laid out in tables. If you have an image that is curvy, lay a grid over it and find out where you can make slices to create a table. (I'll cover this in more detail later.)

There are three very basic elements in a table: there is the table itself, a table row, and a table cell. The tags for these are `<table></table>`, `<tr></tr>`, and `<td></td>`, respectively. You might wonder, "Why is the tag for a table cell `<td></td>`?" Well, "td" really stands for *table data*, so if I talk about the table data I am referring to the table cell.

To make a table, you start the table tag, start a table row tag, start a table data tag, end a table data tag, end the table row tag, and then end the table tag. Like this:

```
<table border="1">
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
    <td>Cell 3</td>
  </tr>
</table>
```

Make sense? For each cell you need you put a `<td></td>` tag between the `<tr></tr>` tags. For each row you need you put a `<tr></tr>` tag between the `<table></table>` tags. Just remember that you can have a cell span multiple cells, and you can have a row span multiple rows, but you can't have a cell without a span in a lower row wider than any row above it.

Remember that it has to be a grid. Take a look at the attributes of the table, table row, and table data tags.

`<table></table>`

Border – Sets the amount of border you want around your table. If the value is 0 then no border will be displayed. The default value of border is 0.

Cellspacing – Sets the amount of space, in pixels, between the cells. The default value is 2.

Cellpadding – Sets the amount of space, in pixels, between the cell's border and its content. The default value is 2.

Width – Sets the width of the table. It can be either a number, in pixels, or a percentage. The percentage is based on how large or small the client window is.

Height – Sets the height of the table. It also can be a number or a percentage, just like the width property.

Align – Sets the alignment of the table. Possible values are left, center, and right. By default, the tables will be aligned left.

Bgcolor – Sets the background color of the table.

<tr></tr>

Align – Sets the alignment of the table row. Possible values are center, justify, left, and right. By default, left is selected.

Bgcolor – Sets the background color of the table row. This overrides the bgcolor in the table tag for the particular row that it is specified on.

Width – Sets the width of the table row.

Height – Sets the height of the table row.

Valign – Sets the vertical alignment of the table row. Possible values are middle, baseline, bottom, and top. By default, middle is selected.

<td></td>

Align – Sets the alignment of data and the justification of text in the table cell. Possible values are center, justify, left, and right. By default, left is selected.

Bgcolor – Sets the background color of the table cell. This overrides the bgcolor in the table row tag for the particular cell that it is specified on.

Colspan – Sets the number of columns, or cells, that this cell will span.

Rowspan – Sets the number of rows that this cell will span.

Width – Sets the width of the cell.

Height – Sets the height of the cell.

Nowrap – Sets whether the browser will automatically perform word wrapping on the cell.

Valign – Sets the vertical alignment of data and text in the table cell. Possible values are middle, baseline, bottom, and top. By default, middle is selected.

Before continuing, I want to show you what these options can do. Let's create a table with a border of 1, no cellpadding, three rows, three cells in each row with some spanning multiple columns, and background colors on some cells.

```
<html>
  <head>
    <title>Table Example</title>
  </head>
<body bgcolor="#ffffff">





```

This code should look something like Figure 3.5.

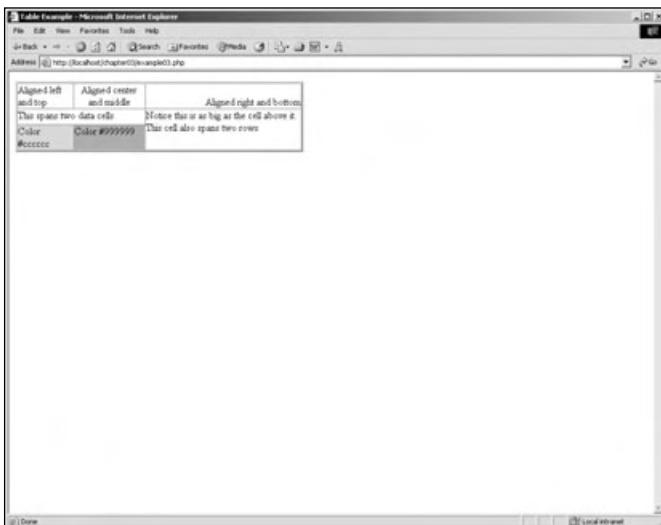


Figure 3.5 Example code results after using the table, table row, and table data tags.

See all the cool things you can do with tables? Once you get the hang of the tables and the way they flow you can start nesting tables inside of other tables to get different effects. For future games in this book, you will start out with a base table that will be called the *framework* table. This table encompasses the five major elements to a page: a header, a left bar, a content panel, a right bar, and a footer. Then, inside of each of those you may have some content, a menu, or nothing at all. But you will always start out with this basic base table:

```
<table border="0" cellpadding="0" cellspacing="0" width="750">
<tr>
    <td align="left" valign="top" colspan="3">HEADER</td>
</tr>
<tr>
    <td align="left" valign="top" width="100">LEFT</td>
    <td align="left" valign="top" width="550">GAME GOES HERE</td>
    <td align="left" valign="top" width="100">RIGHT</td>
</tr>
<tr>
    <td align="left" valign="top" colspan="3">FOOTER</td>
</tr>
</table>
```

For the purpose of this book, your games will be comprised of multiple tables or forms that are formatted with tables, and they will go into the content panel. You will write out your header, left, right, and footer panels with PHP itself. But let's not get ahead of ourselves; I will cover all of that in the next chapter. For now, get some practice with laying out tables with multiple graphics to make a cool-looking interface.

Layouts, Tables, and Graphics

As promised, you are going to make a cool-looking interface. You can use this interface on your games if you like or you can create your own. By the time you're done with this section you will be a master at laying out tables.

When laying out graphics to work with it is handy to have a graphics application such as Photoshop or Paint Shop Pro. These programs allow you to create the graphics for the Web and they will make slicing up graphics much easier. I have included Paint Shop Pro on the CD, but you can use whatever graphics application you prefer as long as it can save .gifs or .jpgs.

When laying out a table, always start in the upper left-hand corner of the image and make your way across the top in columns. This is what we call left-to-right order. Building a table in this manner will better allow you to see the final table layout in code. I have included an image on the CD under the Chapter 3 directory called interface.pcx. Go ahead and open the image so you can create a table layout with graphics.

Now you have to take the image and break it up into columns. I usually create three columns in a table. This seems to give very good flexibility in how the final product turns out—not to mention that it fits into our five-panel layout very nicely.

To create the columns and rows you will need to drag grid lines onto your image. In Paint Shop Pro this is done simply by dragging the mouse from the ruler to the image. Create two vertical grid lines in the image. Start the first grid line just to the right of the left-hand bar and work your way across. Place your second vertical grid line directly to the left of the right bar. Create a horizontal grid line just above the bottom bar. Now create a horizontal grid line just below the top bar of the graphic. Your results should look like Figure 3.6.

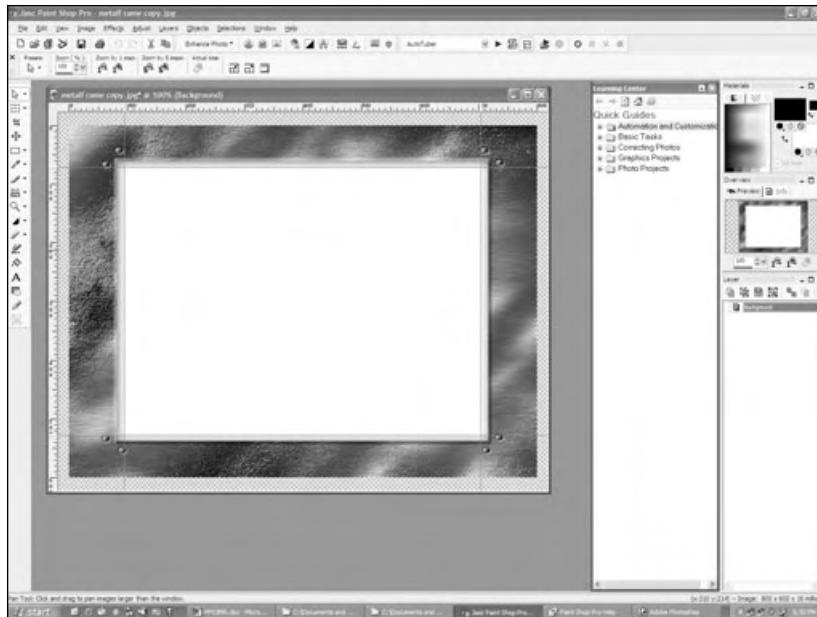


Figure 3.6 Creating the first set of grid lines.

Notice how you can start to see a table layout appearing. The top of the image makes up one row and three columns. So the general table layout in HTML will look like this:

```
<table border="0" cellpadding="0" cellspacing="0">
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
  </tr>
</table>
```

You don't necessarily have to slice up your image like this though; there are several options available to you. An alternative would be to slice out the top image as one big graphic so that you would then have a single row with a single column in it. If you needed the top row to span multiple columns you would just put a colspan in the table data.

Caution

Only slice out images as large images if there isn't a lot of color information in them. Otherwise you will end up with very large files and very long load times.

You should now have all of the grid lines in place and have a fairly good idea of how to create your table. The only thing left to do before we leave the graphics realm and head back to code is to slice out the images and save them in a format that we can use.

To slice out an image, simply click on the Select tool and drag your cursor in between the grid lines. The selection should snap to the grid lines. After you have a section of the image selected, copy the selection, start a new image the size of the selection that you have made, and paste the selection into the new image. Once you have done this you will need to save the image as a .gif or a .jpg. Since we have quite a bit of color information that we need we will use the .jpg file format for all the reasons described earlier.

Once you have sliced out and saved all of the images, you can create the table. When doing table layouts with images to recreate a specific look, you don't want to have any borders, cellpadding, or cellspacing in your table. If you did have cellpadding or cellspacing in your table, then when you put the images into it they would not butt-up against each other. This would totally defeat the effect that you are trying to achieve. Remember, all you are trying to do is create a shell for your game; you aren't trying to display some Excel spreadsheet where you would need borders and padding to make it readable.

Once created, your table should look something like the following:

```
<table border="0" cellpadding="0" cellspacing="0" width="750">
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
  </tr>
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top">GAME GOES HERE</td>
    <td align="left" valign="top"></td>
  </tr>
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
  </tr>
</table>
```

After all of this craziness you will end up with something that looks like Figure 3.7 when you hit it with your Web browser.

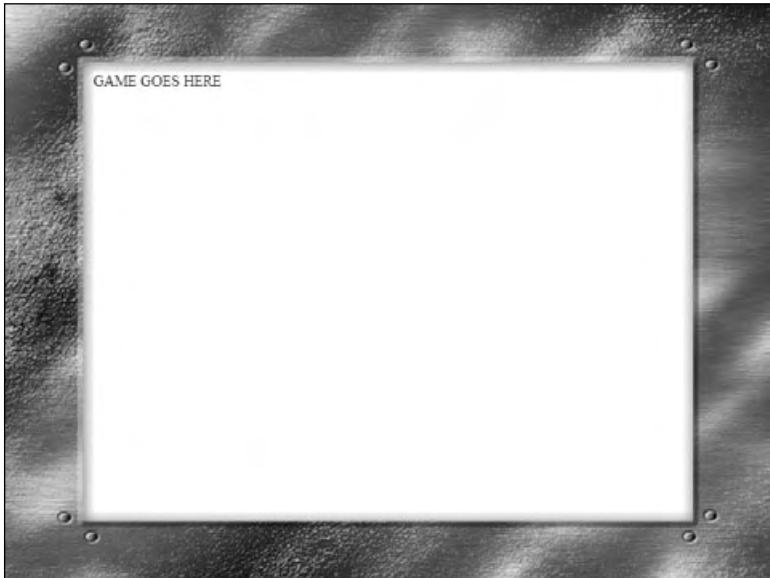


Figure 3.7 Results of creating your table with graphics.

Great, you now understand when to use the different graphic types and how to lay out simple or complex tables. The only thing left is getting input into your game. This is where forms come in. Forms give your user a way to enter in the information that they want.

Creating Forms for Input

Creating a form for input is quite simple. It is just like using every other HTML tag. A form essentially consists of three elements: the form itself, the fields in the form, and some element to submit the form. Take a look at the attributes of the Form tag.

`<form></form>`

Action – Sets the URL to which the form is posted for processing. Can be a relative or absolute path, or a mailto to process the form via email.

Enctype – Sets the MIME encoding type of the form. This is usually used only when uploading files to the server.

Method – Sets how the form is going to be retrieved by the processing page. Possible values for this are POST (will post the information to the server) and GET (will put the name value pairs of the form on the query string).

Name – Sets the name of the form.

Note

When you use a form in the examples in this book you will be using the POST value for the method. The main reason for this is that when you use the POST method you are not limited in the amount of data that can be sent to the HTTP server. The GET method limits you to the maximum length of a URL, which is currently 2048 bytes. 2048 bytes is not a lot of information; you could fill that up with just a few text fields.

To allow your user to enter information you must give them a way to type in that information. HTML offers several form controls to allow you to do this.

`<INPUT>`

Name – Sets the name of this element.

Type – Sets the type of input element. Possible values for this attribute are TEXT, BUTTON, CHECKBOX, RADIO, IMAGE, FILE, HIDDEN, RESET, SUBMIT, and PASSWORD.

Width – Sets the width in characters of the input field if it is of type text, otherwise sets the width of the control in pixels.

Value – Sets the initial value of the input field if it is of type text, checkbox, radio, or file. This attribute is optional except for the radio element.

Checked – If this element is of type checkbox or radio it will select this element.

Accept – Sets the content types this element will accept if it is of type file.

Readonly – Sets this element to be read only.

TabIndex – sets the order in which this element will be selected if the user hits the Tab key.

<TEXTAREA></TEXTAREA>

Name – Sets the name of this element.

Rows – Sets the number of rows in characters for this text area box.

Cols – Sets the number of columns in characters for this text area box.

TabIndex – Sets the order in which this element will be selected if the user hits the Tab key.

Wrap – Sets how the text in the element will wrap. Possible values are soft, hard, and off. Soft word wrapping means that the text will be displayed with word wrapping but submitted without carriage returns. Hard means text is displayed with word wrapping and submitted with carriage returns. Off means word wrapping is disabled.

There is one more control that needs to be covered: the SELECT control. This is a special control for two reasons: 1) it has its own child controls, called OPTIONS, and 2) it can create a drop-down or a list box.

Tip

A drop-down list is an element that allows you to select a single item from a drop-down box. A list box is an element that allows you to select multiple items in a list.

Take a look at the code example below to see how to use the SELECT control.

```
<SELECT name="dropDownList">
  <OPTION VALUE="1">Element 1</OPTION>
  <OPTION VALUE="2" SELECTED>Element 2</OPTION>
  <OPTION VALUE="3">Element 3</OPTION>
</SELECT>

<SELECT name="listBox" SIZE="4" MULTIPLE>
  <OPTION VALUE="1">Element 1</OPTION>
  <OPTION VALUE="2">Element 2</OPTION>
  <OPTION VALUE="3">Element 3</OPTION>
</SELECT>
```

Take a quick look at the very first SELECT element. It is just a drop-down list with three values in it. To specify the values in the drop-down list you simply use the OPTION tag. The OPTION tag has only two attributes that can be used: VALUE and SELECTED. VALUE sets the value for that element, and SELECTED specifies the element that is initially selected when the form loads. The text in between the OPTION tags is exactly what the user will see.

There isn't much difference between the first SELECT element and the second SELECT element. The major differences are the SIZE attribute and the MULTIPLE attribute. The SIZE attribute tells the select box how many items to display at once, while the MULTIPLE attribute tells the select box that the user is allowed to select multiple items.

There you have it! Those are all the form controls that you can have in a form to allow the user to enter data. Just to make sure you understand how it all works together, take a look at the following code example:

```
<HTML>
  <HEAD>
    <TITLE>Form Example</TITLE>
  </HEAD>
<BODY BGCOLOR="#ffffff">
<FORM name="frm_Example" action="#" method="post">
  <b>Text Box</b> <input type="text" name="txtSingleLine" width="25"><br>
  <b>Text Area</b> <textarea name="txtArea" rows="10" cols="15"></textarea><br>
  <b>Check Box</b> <input type="checkbox" name="chkBox" value="1"><br>
  <b>Radio Button</b> <input type="radio" name="rbButon" value="1">
  <input type="radio" name="rbButton" value="2" checked><br>
  <b>Drop Down List</b>
  <SELECT name="dropDownList">
    <OPTION VALUE="1">Element 1</OPTION>
    <OPTION VALUE="2" SELECTED>Element 2</OPTION>
    <OPTION VALUE="3">Element 3</OPTION>
  </SELECT>
  <br>
  <b>List Box</b>
  <SELECT name="listBox" SIZE="4" MULTIPLE>
    <OPTION VALUE="1">Element 1</OPTION>
    <OPTION VALUE="2">Element 2</OPTION>
    <OPTION VALUE="3">Element 3</OPTION>
  </SELECT>
</FORM>
<p>That's it...look at that form!!!</p>
</BODY>
</HTML>
```

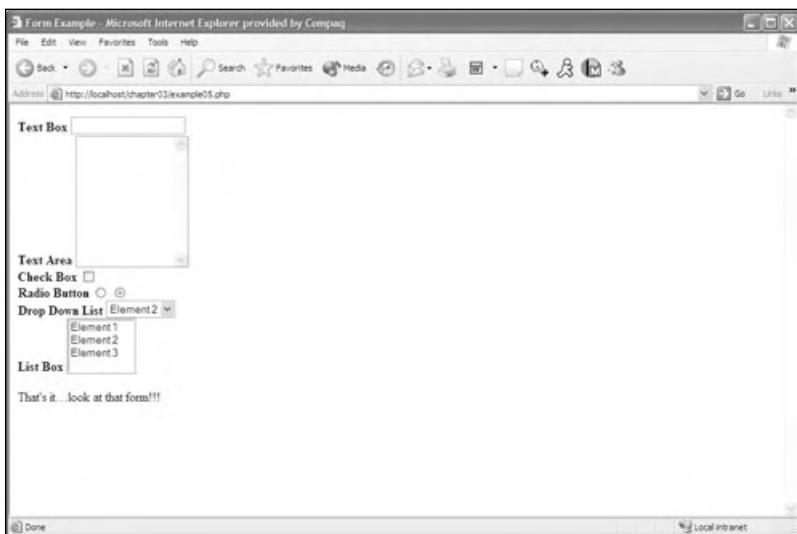


Figure 3.8 The example form.

You should now have something that looks similar to Figure 3.8.

There is a lot more you could do with this form to make it look better and read more clearly. One of the biggest steps you could take to improve this form is to put it in a table layout so that all of your labels and form elements line up. You could also arrange the form elements differently in the table.

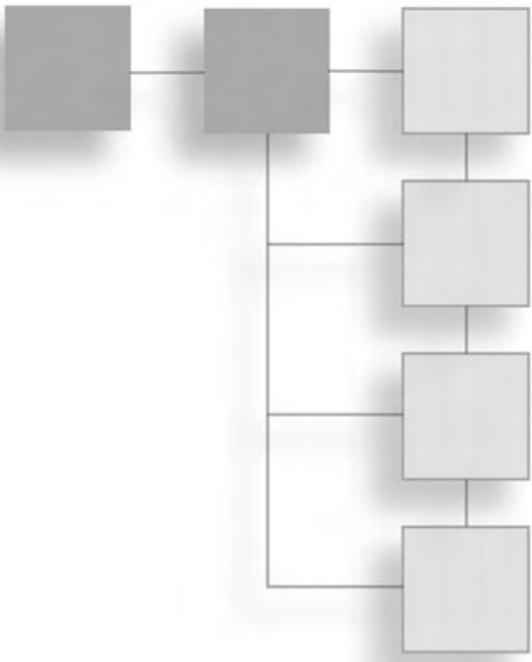
Conclusion

That was a lot of information to take in. You have done quite a bit in this chapter and should be very proud of yourself. You learned the basic structure of the HTML document, how to format text, how to work with tables to create layouts, how to use graphics in a Web page, how to use graphics and tables together to create a really great look and feel for your game, and how to create a form to take input from your user.

However, there is one last thing before you move on to combining your newfound HTML knowledge with PHP. This chapter provided a limited view of HTML; thus I strongly encourage you to investigate HTML further and learn more about it. Use the Web and other resources to learn about DHTML, JavaScript, and CSS (Cascading Style Sheets). All of these elements will allow you supreme control of your Web page.

Now on to PHP!

This page intentionally left blank



PART II

ENTER THE LANGUAGE

CHAPTER 4

Say Hello to PHP 55

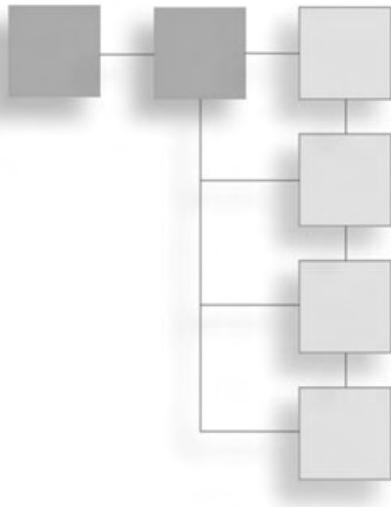
CHAPTER 5

Operators, Statements, and Functions 77

This page intentionally left blank

CHAPTER 4

SAY HELLO TO PHP



- Creating a PHP page
- Data Types
- Type Casting
- Variable Variables
- Constants
- Naming Conventions
- Functions for Variables
- Functions for Strings
- Regular Expressions and Pattern Matching
- Processing Forms with PHP

What can PHP do for you? PHP can create dynamic applications that can be delivered over the Web. In comparison, HTML can only create static content because the Web browser interprets the page when it is requested; the server doesn't do anything but send the page to the browser. But when a PHP page is requested on the server, the server itself processes the code that is in the proper delimiters and sends the final results to the Web browser. This allows you to pull different content into the page each time it is requested. This is very handy for you because now you can create an interactive game that is updated from user input.

Creating a PHP Page

Creating a PHP page is very similar to creating an HTML page, but there are two major differences. First, your page extension will be .php instead of .html and, second, you will also have PHP code. To start a block of PHP code you use the following delimiters: <?php, which will start the block of PHP code and ?>, which will end the PHP code block. Take a look at the following PHP page:

```
<HTML>
  <HEAD>
    <TITLE>First PHP Page</TITLE>
  </HEAD>
<BODY BGCOLOR="#ffffff">
<?php
  echo("This text was written out by PHP");
?>
</BODY>
</HTML>
```

Note

In ASP, the delimiters for code blocks are <% %>. If you like these better, you can set up your PHP interpreter to use them instead of the <?php ?> delimiters.

All this does is simply print the text “This text was written out by PHP” into the browser. You can also print out HTML tags in the echo statement, like this:

```
echo("<p><b>This text is bolded and written out by PHP</b></p>");
```

This is very important to keep in mind. Why? Well, if you needed some client-side code that used variables from the server side you could write out the client-side code with PHP. I will not cover any of the client-side languages such as JavaScript, but it is still important to keep in mind.

Another very important note is the ending of a PHP line of code. It ends with the semi-colon (;). If you don’t have this, your code will not work properly. If you haven’t been coding in C/C++ or Java then you need to get into the habit of putting semicolons at the end of your code.

These are the basics of creating a PHP page. Now you can move along to the data types available to PHP.

Data Types

PHP has many of the data types that a language such as C/C++ has. It uses the three basic types: integer, double, and string. PHP also uses constants, arrays, and objects. The unique part of PHP is that you do not explicitly declare the type for the variable in PHP. The reason for this is that PHP converts the variables on the fly to the appropriate type. This is called *type juggling*. Here is an example of creating variables using the three basic types:

```
$var1 = 1;      // integer  
$var2 = 2.0;    // double  
$var3 = "21";   // string
```

Tip

All variables start with a \$, and must then begin with a non-numeric character. It is good practice to declare all of your variables before using them; although this is not required, it makes that code a bit clearer for anyone who is reading it.

You might have noticed that the Boolean type was not mentioned. That is because PHP does not support it. Instead, PHP evaluates true and false like C/C++ does. Any integer that equals 0 will evaluate to false and anything that is non-zero will evaluate to true. With strings, any string that is blank ("") will evaluate to false, and any string with a character in it will evaluate to true.

Variables in PHP are also case sensitive. That means that \$someVar is not the same as \$SomeVar. However, built-in functions and structures are not case sensitive. So echo is the same as ECHO.

To further clarify the concept of type juggling, check out the following examples:

```
$somevar = 1;          // This is currently set to an integer.  
$somevar = 2.67;       // It is now set to a double.  
$somevar = "123A String"; // It is now set to a string.  
$x = 1;               // This is an integer.  
                      // This is still an integer. After the addition the  
                      // value is now 126.  
  
$x = 3 + $somevar;
```

See how that works? It can come in handy in some cases because it cuts back on the number of variables you will need. In the same respect it can get pretty crazy if you're trying to debug some code and it is juggling types on you.

When PHP is type juggling strings, it follows a few basic rules. If the string begins with a valid integer, the string will evaluate to that value. If the string includes an integer and it is not at the beginning, the string will evaluate to 0. A string will be type cast as a double if and only if the value of the string is the double. So the string "1.10" will evaluate to a

double with the value 1.10. But a string with the value 88.5 percent will evaluate as an integer with a value of 88.

Type Casting

PHP also offers you a way to explicitly type cast your variables. The major reason for this would be if you needed to make sure that you were comparing an integer to an integer.

```
if( (int)$somvar1 == (int)$somevar2)
{
    echo("These are the same");
}
```

Take a look at Table 4.1 for the available type casts.

Table 4.1 Type Casts

Type Cast	Result
(int)	Casts the variable to an integer.
(integer)	Casts the variable to an integer.
(double)	Casts the variable to a double.
(float)	Casts the variable to a double.
(real)	Casts the variable to a double.
(array)	Casts the variable to an array.
(object)	Casts the variable to an object.

* These change the data type of a variable

Variable Variables

PHP also supports variable variables. In most cases, variables have dynamic values (meaning they change). But variable variables make the variable name dynamic. Variable variables are not very practical and create quite a bit of confusion, but they are really cool. Take a look at what they can do:

```
$somevar = "Gun1";
$$somevar = "Gun2";
echo($Gun1);      // prints Gun2
echo($$somevar);  // prints Gun2
```

See what I mean? It's pretty confusing, but really cool. Here is what is happening. The first line of code creates a variable of type string called \$somevar. The second line of code uses the value of the first variable (\$somevar) to create the name of the new variable. So the new variable is named \$Gun1. When you print these two variables you end up with exactly the same results.

Constants

Constants are values in your application that will never change. For example, instead of including the literal string "My Cool Game", you create a constant and reference the constant throughout your code. This makes your code much easier to update. Instead of changing hundreds of lines of code where "My Cool Game" exists, you simply change the constant.

To define a constant in PHP you use the `define()` function.

```
define("GAME_TITLE", "My Cool Game");
```

The `define()` function takes two parameters. The first is the name of the constant, and the second is the value of the constant. To reference this constant later in your code you would simply use what you entered as the first parameter.

```
echo(GAME_TITLE);
```

You can also tell if a constant is defined by using the `defined()` function. This will return 1 if the constant is defined and 0 if the constant is not defined.

```
if(defined(GAME_TITLE))
{
    echo(GAME_TITLE);
}
```

PHP also includes several built-in constants that are available to you. PHP has defined TRUE as 1 and FALSE as 0. You can retrieve the version of PHP you are using by referencing the `PHP_VERSION` constant. `PHP_OS` will tell you what OS PHP is running on. You can retrieve the current file being parsed and the current line number by referencing the `_FILE_` and `_LINE_` constants.

Additionally, PHP gives you access to some constants for error reporting, such as `E_ERROR`, `E_PARSE`, and `E_WARNING`. You can also access several variables that are predefined. You can find out what these variables are by using the `phpinfo()` function. You used this function while testing your PHP installation in Chapter 2.

Naming Conventions

Now you and I both know that PHP type casts all of its variables. However, I still like to prefix my variables and form element names with what I will be using them for. For example, if I will be using the form element as a string I prefix the name with `str`. If I were going to use the variable as an integer I would prefix the variable with an `n`. You don't have to do this at all, but it does make reading code a whole lot easier, especially with all the type cast juggling that PHP does. It helps you keep things straight. If you don't already have a naming convention for variables, come up with one. Table 4.2 shows the notations I use most often.

Table 4.2 Variable Prefixes

Data Type	Prefix
Integer	n
float	f
double	dbl
string	str
char	c
char array	sz
array	arr
class	C
member variable	m_

Those are just a few of the prefixes I use. Again, you don't have to use them at all, but I do recommend that you come up with some sort of naming convention. It really does help out when you come back to a piece of code after a while, or when someone else is reading your code.

As you may have noticed, it is nothing more than Hungarian notation. If you are familiar with Windows programming then you are probably very comfortable with Hungarian notation.

Functions for Variables

Now you know what types of variables PHP can handle and how to use the majority of them. (I will cover objects and arrays later on.) Now take a look at the built-in functions that PHP gives us for working with variables.

`gettype()`

`gettype()` determines that data type of a variable. `gettype()` returns a string. Possible return values for `gettype()` are: integer, double, string, array, object, and unknown type (and since PHP 4: Boolean, resource, and NULL).

```
if(gettype($somevar) == "string")
{
    echo($somevar);
}
```

`settype()`

`settype()` explicitly sets the data type of a variable. The type is passed in as a string. Possible values are: integer, double, string, array, or object. If the type is successfully set, then `settype()` returns true; otherwise it returns false.

```
If(settype($somevar, "integer")
{
    echo("Successfully set the type to a integer");
}
```

`isset()`

`isset()` is used to determine whether a variable has been given a value. If the variable has been given a value, then `isset()` returns true; otherwise it returns false.

```
if( isset($somevar) )
{
    echo("The variable has a value");
}
```

unset()

`unset()` is used to unset a variable, or destroy it. This essentially frees all the memory that is associated with that variable.

```
unset($somevar);
if(isset(($somevar))
{
    echo("This text will never print!");
}
```

empty()

`empty()` is the exact opposite of the `isset()` function. It will return true if the variable has not been set, and false if it has been set. This would be the same as saying:

```
if( !isset($somevar) )
```

is_datatype()

`is_int()`, `is_integer()`, and `is_long()` all do exactly the same thing. They tell you if the variable in question is an integer. The double data type also has three functions: `is_double()`, `is_float()`, and `is_real()`. The `is_string()` function tells you the variable is a string, and the `is_array()` and `is_object()` functions work the same way with their respective data types.

```
$somevar = "this is a string";
if(is_string*$somevar))
{
    echo("The variable is a string");
}
```

Functions for Strings

Understanding how to manipulate strings in PHP will help you tremendously. It will allow you to validate user input and extract data from text files in an efficient manner. PHP gives you an enormous number of functions to manipulate strings—take a look at Table 4.3.

Most of the functions are fairly self explanatory, such as `strlen()`. But others need a little bit more attention. `printf()` and `sprintf()` are good examples of these; let's take a closer look at these two functions.

Table 4.3 String Functions

Function	Return Data Type	Description
addslashes(string)	string	Adds escape slashes to a string.
bin2hex(string)	string	Converts a binary string into ASCII hexadecimals.
chop(string)	string	Removes the trailing white space from a string.
chr(ASCII)	string	Returns the character for the specified ASCII code.
chunk_split(string, [chunklen], [end])	string	Inserts the string end at every chunklen in the specified string.
convert_cyr_string(string, from, to)	string	Converts the specified string from one Cyrillic character set to another.
crypt(string)	string	DES-encrypts the string.
echo(string)	void	Prints the specified string.
explode(separator, string)	array	Splits the specified string into an array.
flush()	void	Flushes anything that is waiting in the output buffer.
get_meta_tags(filename, [use_include_path])	Array	Returns an array of all the <META> tags in the specified file.
htmlentities(string)	string	Converts the characters in the specified string to their HTML equivalent.
htmlspecialchars(string)	string	Converts any special characters in the string to their HTML equivalent.
implode(delimiter, array)	string	Uses the delimiter to join together each element in an array into a string.
ltrim(string)	string	Strips the white space from the beginning of the string.
md5(string)	string	Calculates the MD5 hash of the specified string.
nl2br(string)	string	Inserts the tag before all the line breaks in the string.
ord(string)	int	Returns the specified ASCII value of the first letter of the string.
parse_str(string)	void	Parses the string into variables like it was a query string.
print(string)	void	Prints the specified string.
printf(string, [arg])	void	Outputs a formatted string.
quoted_printable_decode(string)	string	Converts a quoted printable string to an 8-bit string.
QuoteMeta(string)	string	Escapes meta characters in the string.

Table 4.3 String Functions (*continued*)

Function	Return Data Type	Description
rawurldecode(string)	string	Decodes a URL-encoded string.
rawurlencode(string)	string	Encodes a string to a URL-encoded string.
setlocale(category, locale)	string	Sets the locale information for functions in the specified category.
similar_text(string1, string2, [percent])	int	Calculates the similarity between string1 and string2.
soundex(string)	string	Calculates the soundex key for the string.
sprintf(format, [args])	string	Returns a formatted string.
str_replace(pattern, replacement, string)	string	Replaces all occurrences of pattern with replacement in string.
strchr(string1, string2)	string	Finds the first occurrence of string2 in string1.
strcmp(string1, string2)	int	Compares string1 against string2.
strcspn(string1, string2)	int	Returns the number of characters in the beginning of string1 that do not match any of the characters in string2.
strip_tags(string)	string	Strips all the HTML and PHP tags from the specified string.
stripslashes(string)	string	Strips all escape character slashes from the string.
strlen(string)	int	Returns the length of the string in characters.
strpos(string1, string2)	int	Finds the first occurrence of string2 in string1.
strrev(string)	string	Returns the specified string in reverse order.
strrpos(string1, string2)	int	Finds the last occurrence of string2 in string1.
strstr(string1, string2)	string	Finds the first occurrence of string2 in string1.
strtok(string1, string2)	string	Tokenizes string1 into segments separated by string2.
strtolower(string)	string	Converts all characters in the string to lowercase.
strtoupper(string)	string	Converts all characters in the string to uppercase.
strtr(string, from, to)	string	Replaces all occurrences of from in the string with to.
substr(string, start, [length])	string	Returns the characters in string from the specified start point.
ucfirst(string)	string	Converts the first character of the string to uppercase.
ucwords(string)	string	Converts the first character of each word to uppercase.

printf() and sprintf()

Each of these functions produces a formatted string. `printf()` prints the formatted string to the page, whereas `sprintf()` returns you the formatted string without printing it. Take a look at the `sprintf()` function:

```
string sprintf(string format, mixed [args]...);
```

Note

In PHP, `printf()` and `sprintf()` function like `printf()` and `sprintf()` in C/C++.

The format string indicates how each of the arguments should be formatted. There are 10 different formatting arguments.

- d Decimal integer
- b Binary integer
- o Octal integer
- x Hexadecimal integer with lowercase letters
- X Hexadecimal integer with uppercase letters
- c Character whose ASCII code is the integer value of the argument
- f Double
- e Double using exponential notation
- s String
- % A literal percent sign

The format string uses these arguments to create a string. For example:

```
// The following line gets a string for printing later
$strFormatted = sprintf("You fired your gun %d times", $numFired);
```

See how that works? You simply create a string, and then put in the appropriate argument where you want one of your variables to show up. All arguments start with a % sign. So to print the literal percent sign, your string would look like this:

```
printf("This is 40%");
```

Notice how the percent sign doesn't take any additional arguments. You can also add padding or number formatting to your string. The delimiter to add padding is a single quote (') and then the specified padding element. For instance, to add a string of periods to a line after a string you would use '.' as the padding argument. Padding arguments are not required, and by default they are a single space. Take a look at the code example below to see how to use the padding argument.

```
printf("%'.-80.80s%'.3d%s", $title, $number, "<BR>");
```

Take a closer look at the arguments for this string. It consists of three directives: a string, an integer, and a string. The first argument, '% -80.80s', shows that periods should be used, the hyphen tells the string to be aligned to the left, and it sets the minimum and maximum widths to 80. The second argument produces a right-justified, period-padded, integer element with a maximum width of three characters. The final argument is simply a string that you specify to put in an HTML
. If you had specified values for the title and number variables, the output would look something like this:

Appendix.....456

Earlier I mentioned the ability to format numbers too. This is the same principle as the padding in a string. You simply specify how many digits you would like to see. If you specify a width for an integer you will see only that many characters, and if you specify a width for a float it will specify how many digits will appear after the decimal point. Take a look at the following code sample:

```
$someint = 4356;  
$dollars = 20;  
printf("%3d %.2f", $someint, $dollars);  
// The output looks like 435 20.00
```

In the example above, %3d limits that argument to printing three characters. The %.2f argument specifies that it is a floating point number, and that it should print two more characters after the decimal place.

If this isn't enough control over your number formatting, PHP offers you a number formatting function called `number_format()`.

```
string number_format(float num, int precision, string dec_point, string thousands_sep)
```

The `number_format()` function can take up to four arguments and returns a string. Look at the following example:

```
$num= 987654321.1234567;  
echo(number_format($num));  
echo(number_format($num, 2));  
echo(number_format($num, 7, chr(44), " "));
```

The output from the preceding code looks like this:

```
987,654,321  
987,654,321.12  
987 654 321,1234567
```

Now that you have complete control over formatting and the majority of functions for string manipulation, I'll move on to regular expressions.

Regular Expressions and Pattern Matching

Regular expressions are used to provide advanced string matching and manipulation. A *pattern* in a regular expression is nothing more than a set of characters that describes the nature of a string. For instance, you could find a literal string or validate that the input the user entered was actually an e-mail address. Regular expressions have a mini-language of their own. Once you learn it, you can apply it to many other languages in addition to PHP. You will use regular expressions sparingly in your games. Most of the time it will be to simply validate that the input entered is in the form that you expect it in, which is very important, isn't it?

Examine the following regular expression pattern:

```
^create
```

The carat (^) in this pattern tells the regular expression engine that it should only match this pattern on the beginning of strings. So the string “Create a new character” would meet the criteria for the match, but “To create a new character” would not meet the criteria. Regular expressions also offer you a dollar sign (\$) character to match strings that end with the pattern.

If you take the previous example of ^create and turn it into the string created\$, then it would no longer find a match on “Create a new character” but it would find a match on “Your character has been created.” And if you combine the carat (^) and the dollar sign (\$) together like this:

```
^create$
```

you can now match on an exact word. So, “To create a new character” would now meet the criteria of the pattern. You are not limited to matching on literal characters either. You can also match on special characters such as a new line or a tab. To do this, you would simply use the appropriate escape character, in your string. For instance, to match a tab at the beginning of a line you would do this:

```
^\t
```

To match on a new line, a carriage return, or a form feed you would use the respective escape characters \n, \r, or \f. For punctuation marks you would also escape the character; for example, a literal period would be \., and a literal backslash would be \\.

So far you know how to match literals only, but you'll need a way to describe the pattern more loosely. You can describe this pattern with character classes. Creating a character class is very simple: you simply place the content within brackets. For example, if you wanted to match all vowels you would create a character class that looks like this:

```
[AaEeIiOoUu]
```

You can also specify ranges in your character classes by using a hyphen, like this:

```
[a-z]      // match any lowercase letter
[A-Z]      // match any uppercase letter
[0-9]      // match any digit
[\f\t\r\n] // match any white space
```

Let's say you want to create a character class to forbid a digit from being the first character in a string. To do this you would again use the carat (^). Inside a character class the carat (^) means "not" instead of the beginning of the string.

```
^[^0-9][a-z]$
```

This will match any strings such as "all23" or "u47". But it will not match strings such as "7all" or "8teen". PHP has several of these character classes already built in. Take a look at Table 4.4.

To allow even more flexibility in your patterns, you can use curly braces ({{}}) to match multiple cases of characters or character classes. So to match exactly x number of occurrences of the previous character or character class you would do something like this:

```
^a{1,5}$ // matches: a, aa, aaa, aaaa,  
or aaaaa
```

You can also find a string with x or more occurrences of a character or character class. For instance:

```
^b{2,}
```

This will match a string with two or more b's in it. So it would find a match on bbooo, or bbblood, etc.

There are five more special characters in regular expressions you should know about. They are:

period	.
question mark	?
star	*
plus	+
pipe	

Table 4.4 PHP Character Classes

Character Class	Description
[:alpha:]	Matches any letter.
[:digit:]	Matches any digit.
[:xdigit:]	Matches any hexadecimal digit.
[:alnum:]	Matches any letter or any digit.
[:space:]	Matches any white space.
[:upper:]	Matches any uppercase letter.
[:lower:]	Matches any lowercase letter.
[:punct:]	Matches any punctuation mark.

* These classes are defined in PHP and may not work correctly if you try to use them in other languages.

The period is used in regular expressions to represent any non-new-line character. So the pattern `^.s$` will match any two character strings that end in “s” and begin with any non-new-line character.

The question mark means that the previous character is optional. So if you were matching doubles in a string your pattern would look like this:

```
^-?[0-9]{0,}\.?([0-9]{0,})$
```

This may look a bit confusing, but it is actually quite simple to explain. The “`^-?`” means look for a string that begins with an optional minus sign, followed by zero or more digits, “[0-9]{0,}”. Now look for an optional period, “`\.?`”, followed by zero or more digits, “[0-9]{0,}”.

The star is just like a wild card. It means match anything with zero or more of the previous character. So you could further simplify the matching doubles pattern to:

```
^-?[0-9]*\.?([0-9]*)$
```

Make sense? The star is the exact equivalent to saying {0,}, which means zero or more.

The plus symbol means match one or more of the previous characters, or the same thing as saying {1,}. A simple example of this would be matching any integer number:

```
^-?[0-9]+$
```

This is looking for a string that begins with an optional minus sign, followed by one or more digits. Another very handy function of the plus sign would be to validate e-mail addresses, which is something you might want to do quite often in your Web-based games.

```
^.+@.+\\..+$
```

This pattern might look complex, but if you break it down you will find that it is actually quite simple. First, it is looking for a string that begins with any non-white-space character, followed by any non-white-space character. Then it is looking for the literal character “@”, followed by any non-white-space character. Then it looks for the literal character “.” followed by any non-white-space character.

Now this pattern isn’t perfect, but it is close enough. To match on any non-white-space character is a broad scope, but for the most purposes you just want to make sure that the user entered in a valid looking e-mail address.

The final character to take note of in regular expressions is the pipe (|). The pipe behaves exactly like a logical OR operator. This is extremely useful because you can check a string for certain words or characters. For instance:

```
(G|St)un$
```

This will match any string that has the words “Gun” or “Stun” in them.

Using the Regular Expression Functions

PHP has five functions for handling regular expressions. Two of them are used for searching and matching, two are used for searching and replacing, and one is used for splitting.

The most basic of the five functions is `ereg()`. It takes two parameters with an optional third parameter, and returns true if the pattern is found and false if the pattern is not found.

```
bool ereg(string pattern, string source, array [regs]);
```

Let's go back to the e-mail validation pattern that was presented earlier to see how to use the `ereg()` function.

```
$email = "ruts@datausa.com";
$nResult = ereg("^.+@.+\\..+", $email);
if($nResult)
{
    echo("This is a valid email address");
}
else
{
    echo("This is a invalid email address");
}
```

The optional third argument, array [regs], can store the matching substrings of the pattern for later use. This means, for example, that when you pass in an e-mail address it can take the username, domain name, and top-level domain name and store them in the array. Take a look at the following example:

```
$email = "ruts@datausa.com";
$nResult = ereg("^(.+)@(.+)\.(.+)", $email, $arrEmail);
if($nResult)
{
    echo("$arrEmail[0] is a valid email address" .
        "<br>Username: $arrEmail[1]<br>Domain: $arrEmail[2]<br>Top Level: $arrEmail[3]");
}
else
{
    echo("This is an invalid email address");
}
```

The results of the preceding example are:

```
ruts@datausa.com is a valid email address  
Username: ruts  
Domain: datausa  
Top Level: com
```

So what did this do exactly? After `ereg()` verified the pattern, the original string was stored in the first index of the array. Then the first parenthetical substring from the pattern is stored in the second index of the array. The first parenthesized substring, of course, would be the username, `ruts`. Then it matched the “@” symbol and proceeded to match the third argument in the pattern. After it found a match for the third argument it created another index in the array containing the domain name, `datausa`. Finally it matched the top-level domain name, `com`, and created the final index in the array.

`ereg()` also has a sister function called `ereg_i()`. `ereg_i()` functions exactly the same as `ereg()` but ignores case when looking for matching patterns.

Now take a look at the two functions for searching and replacing strings: `ereg_replace()` and `ereg_i_replace()`. Both of these functions search for a given pattern and replace all occurrences of that pattern with the new string that you specify. `ereg_i_replace()` does exactly the same thing as `ereg_replace()`, but it isn’t case sensitive. Each of these functions takes three arguments:

```
string ereg_replace(string pattern, string replacement, string source);
```

You specify a pattern that you would like to look for, a replacement for any occurrences of that pattern, and the source that you are searching. These functions do not have an optional array argument, but they do have something similar. Any parenthetical substring in the pattern will be stored in a buffer that you can access by referencing it as `\1`. There are nine slots that you can access, (`\1...\\9`). Take a look at the following example:

```
$strSource = "Games Are Great";  
$strModified = ereg_replace("G(ame)s", "g\\1S", $strSource);  
echo($strModified);
```

This example takes the string, “Games Are Great”, and searches for a pattern of “G(ame)s”, where it finds a match at the beginning of the string and replaces the match with “gameS”, and returns the modified string. The results look like this:

```
gameS Are Great
```

You can reference the “ame” characters by using the `\1` because it is a parenthetical substring. If the pattern were broken up like, “G(am)(es)”, then “am” would be referenced as `\1`, and “es” would be referenced as `\2`.

The fifth and final function that uses regular expressions is the `split()` function. The `split()` function searches a source string for a pattern and breaks up the source string into an array based on matches of the pattern. A great use of the `split` function is if you are reading a comma-separated list in and need to break apart the strings so you can work with them.

```
$strSource = "e1, d4, e5, e6";
$arr = split(",", $strSource);
echo("$arr[0]<br>$arr[1]<br>$arr[2]<br>$arr[3]");
```

The above example splits the source string on the literal character `,` and creates an array with each string in its own index. The results look like this:

```
e1
d4
e5
e6
```

Unlike `ereg_replace()` and `eregi_replace`, `split()` has an optional third argument. You can specify a limit of how many elements you would like to split. Here is the full `split` function:

```
array split(string pattern, string source, int [limit]);
```

Processing Forms with PHP

In Chapter 3 you learned how to create a form with HTML. Now it is time to learn how to get the data out of the form so you can use it. Getting the data in PHP is very easy. When a form is submitted to the server using a method of GET, each form element creates a PHP variable. When a form is submitted with a method of POST, you need to access the global PHP array `$_POST`. When using GET, the value of the variable created is the content of the form element. When using POST, the index into the array is the form element's name. This is where the value of that form element is stored. Take a look at the following example:

```
<!-- htmlform.php -->
<HTML>
    <HEAD>
        <TITLE>Example Form Processing</TITLE>
    </HEAD>
<BODY BGCOLOR="#ffffff">
<FORM name="frmSample" action="htmlform.php" method="post">
    <b>What is your character's name?</b><br>
    <input type="text" name="strCharacter">
```

```
&nbsp;<input type="submit" value="Submit">
</FORM>
<?php
if(isset($_POST["strCharacter"]))
{
    echo("Welcome, " . $_POST["strCharacter"] . "!");
}
?>
</BODY>
</HTML>
```

If you were using the GET method, the example would look like this:

```
<!-- htmlform.php -->
<HTML>
    <HEAD>
        <TITLE>Example Form Processing</TITLE>
    </HEAD>
<BODY BGCOLOR="#ffffff">
<FORM name="frmSample" action="htmlform.php" method="get">
<b>What is your character's name?</b><br>
<input type="text" name="strCharacter">
&nbsp;<input type="submit" value="Submit">
</FORM>
<?php
if(isset($strCharacter))
{
    echo("Welcome, " . $_REQUEST["strCharacter"] . "!");
}
?>
</BODY>
</HTML>
```

Notice that for each of these forms an action is specified. You told the form to go to “htmlform.php” for processing. The form is pointing back to the same page it lives on for processing, but you could have pointed it to a different page if you wished. The results of this example should look similar to Figure 4.1.

Most of the time you will always point the page back to itself for processing. Since you are developing games, you need just about everything to occur on one page. There are exceptions to this, however—for example, if you were developing a Massive Multiplayer Online Game (MMORPG), which I will discuss in Chapter 12, you would use multiple pages to process the building of units and structures. A good rule of thumb for breaking up forms is, “do it in sections.” This means break up your game into sections. If you are simply

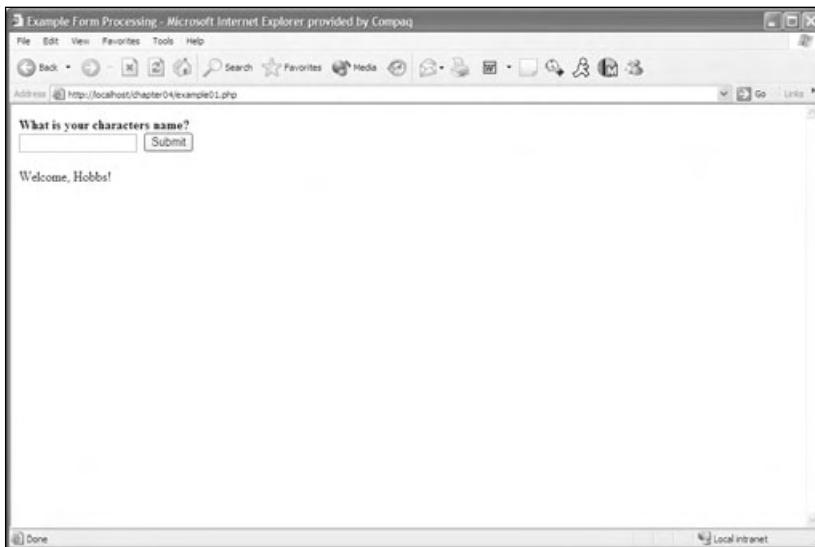


Figure 4.1 Processing form example using the GET method.

taking in coordinates for a chess game, then all the processing should be done on one page. But if you are going to a section of your game to manage units and to a completely different section to manage buildings, then your entire unit processing should be done on one page and all of your building processing should be done on another. Basically, keep everything modular.

Now take a look at a more complex example so you can see how to process all of the form elements.

```
<!-- allformelements.php -->
<HTML>
    <HEAD>
        <TITLE>Processing All Form Elements</TITLE>
    </HEAD>
<BODY BGCOLOR="#ffffff">
<FORM name="frm_Example" action=" allformelements.php" method="post">
<table border="0" cellpadding="2" cellspacing="0">
    <tr>
        <td align="left" valign="top"><b>Character's Name</b></td>
        <td align="left" valign="top"><input type="text" name="strCharacter"></td>
    </tr>
    <tr>
        <td align="left" valign="top"><b>Starting Amount of Gold</b></td>
```

```
<td align="left" valign="top">
    <select name="dblGold">
        <option value="1000">1000</option>
        <option value="2000">2000</option>
        <option value="3000">3000</option>
    </select>
</td>
</tr>
<tr>
    <td align="left" valign="top"><b>Starting Location</b></td>
    <td align="left" valign="top">
        <input type="radio" name="strLocation" value="mountains">Mountains
        <input type="radio" name="strLocation" value="plains" checked>Plains
        <input type="radio" name="strLocation" value="swamp">Swamp
    </td>
</tr>
<tr>
    <td align="left" valign="top"><b>Equipment</b></td>
    <td align="left" valign="top">
        <input type="checkbox" name="nEquipmentID[]" value="1" checked>Sword
        <input type="checkbox" name="nEquipmentID[]" value="2" checked>Chain Mail
        <input type="checkbox" name="nEquipmentID[]" value="3" checked>Shield
        <input type="checkbox" name="nEquipmentID[]" value="4">Cross Bow
        <input type="checkbox" name="nEquipmentID[]" value="5">Arrows
    </td>
</tr>
<tr>
    <td align="left" valign="top" colspan="2"><input type="submit" value="Submit"></td>
</tr>
</table>
</FORM>
<?php
if($_POST)
{
    $strCharacter = $_POST['strCharacter'];
    $dblGold = $_POST['dblGold'];
    $strLocation = $_POST['strLocation'];

    echo("You Chose:<BR>$strCharacter<BR>$dblGold<BR>$strLocation<BR>");
    $listvals = $_POST['nEquipmentID'];
}
```

```
for($i=0;$i<count($_POST['nEquipmentID']);$i++)  
    echo "Equipment ID $i=".listvals[$i]."<br>\n";  
}  
?>  
</BODY>  
</HTML>
```

The results should look like Figure 4.2.

There is one nuance in this form: the naming of the check boxes. Notice how the names are all the same and at the end of the name, “nEquipmentID”, I have two brackets []. This is simply to make this form element an array. Without the two brackets at the end of the form element name you would not be able to access the form elements correctly. Other than that, you access the other form elements as discussed before, by using the global variable \$_POST. To access the individual elements of the check box, “nEquipmentID”, an array named “listvals” is created by requesting the proper form element. Then you loop through each element in the “listvals” array and print out the value. You will learn more about arrays in Chapter 6.

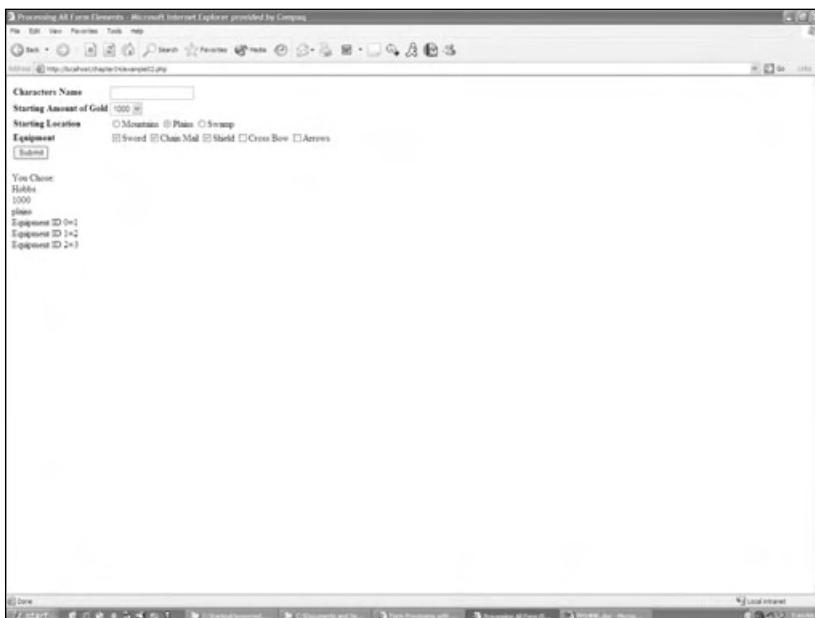


Figure 4.2 Processing form example using the POST method.

Conclusion

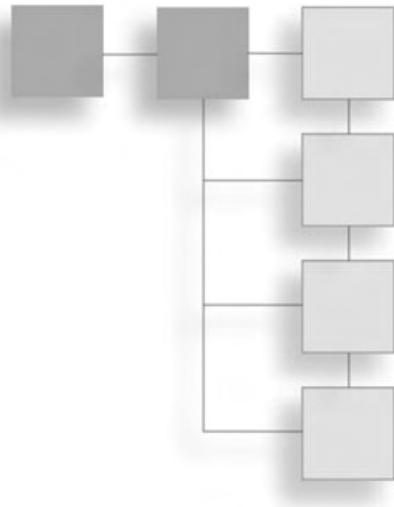
Pretty cool, huh? Well, okay, I'm sure it could be much cooler, but don't worry, you are going to do a lot of cool things in the near future. Next up you will be learning how to control the flow of your code, the basic operators that you have access to, and how to make all your code modular by using functions.

Now would probably be a good time for you to make one of your own forms and process the data using all of your newfound knowledge. If you are still feeling a bit shaky with all of this, don't worry, practice makes perfect. Believe me, you will get plenty of practice.

Get to it!

CHAPTER 5

OPERATORS, STATEM AND FUNCTIONS



- Arithmetic Operators
- Logic Operators
- Bitwise Operators
- Conditional Statements
- Loops
- Functions
- Including Files

In this chapter you will learn the PHP operators, how to create logical statements in PHP to control the flow of your code, and how to create functions to aid you in code reuse. This is the last of the basics of PHP. After this, you will be ready to create your first game!

Operators

An operator is used to determine a value by performing an operation on one or more values. Without operators you would not be able to compare values of variables, perform mathematic operations, concatenate strings, manipulate bits, or even assign a value to a variable. That doesn't leave much of anything that you can do, does it?

Note

Operators in PHP are similar to those in other languages such as C/C++.

Arithmetic Operators

Just like every other programming language, PHP uses the basic mathematical operators (see Table 5.1).

You have probably seen all of these before; the only one that might be foreign to you is the modulus operator. All the modulus operator does is calculate the remainder of an operation. For example:

```
$x = 7 % 2; // Set $x to 1
```

In the example above, 7 is divided by 2 and the remainder, which in this case is 1, is set to the value of the variable \$x.

As you may have noticed by now, you assign variables with the equals sign. This is called the assignment operator. Let's say you wanted to create a variable with a value of negative 1. Here is how you would accomplish this:

```
$x = -1;
```

All fairly straightforward? Now take a look at the expressions used to compare values.

Comparison Operators

Comparison operators are used to test a condition. The results of a comparison operation will always be a Boolean value—i.e., either true or false.

```
$i = 350;
echo($i == 250);
```

Here a variable, \$i, is set to some number, in this case 350. Then the variable is compared to another value, 250. Because the two values are not equal, the example prints false to the screen. Take a look at Table 5.2 for a list of all the comparison operators available to PHP.

I would like to stress a word of caution here: when using the equals (==) operator be careful to make sure you use two equals

Table 5.1 Arithmetic Operators

Operator	Operation Performed
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Table 5.2 Comparison Operators

Operator	Operation Performed
==	Equals
!=	Not equals
<>	Not equals
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

signs. Otherwise the variable will be assigned to whatever you thought you were comparing it to.

```
$i = 25;  
// This will assign 350 to the  
// variable $i and always be true  
if($i = 7)  
echo("It is equal to 7"); // This will print every single time
```

The assignment in an if statement is perfectly legal in PHP. In the example above, instead of checking the value of the variable with 7, it simply assigned the value 7 to the variable and evaluated to true, giving results that were totally unexpected. I cannot stress enough the importance of using the proper operators.

A very simple way of solving the problem would be to put the literal value on the left-hand side of the operand, like this:

```
if(350 = $i)
```

This will generate an error because a value cannot be assigned to a literal. However, you still have the same issue if you are comparing two variables. If you find your code isn't working like you thought it would, make sure you are using the proper operator.

Logical Operators

Logical operators are used to combine conditions, so multiple expressions can be evaluated in a single statement. Take a look at Table 5.3 for a complete list of the logical operators.

Notice that there are two operators for the “logical and” and the “logical or” operators. These operators give the same end results but have different execution orders. The double ampersands will execute before the “and,” and the double pipes will execute before the “or.” I recommend picking one or the other and using it consistently. Not only will this help the readability of the code, it will also help you when trying to debug problems that may occur.

Table 5.3 Logical Operators

Operator	Operation Performed
&&	And
	Or
and	And
or	Or
xor	Exclusive Or
!	Not

Just to make sure that everything is clear, take a look at a few examples:

```
$x = 2;  
$y = 5;  
$z = 0;  
if($x == 2 && $y == 5 && $z == 0)  
{  
    echo("This text will print because all statements evaluate to true.");  
}
```

In this example the statement will print to the screen because all of the statements evaluate to true. If any one of these statements evaluated to false, the string would never print to the screen.

```
if($x == 3 || $z == 0)  
{  
    echo("This text will print to the browser");  
}
```

The text in this example will print to the browser, even though the statement `$x == 3` evaluates to false. The reason for this is the logical or operator; if one of the conditions evaluates to true, the whole statement is true.

```
if((($x == 2 || $z == 3) xor ($x == 5 || $y == 3) xor ($x == 1 || $z == 1))  
{  
    echo("This text will print to the browser");  
}
```

This example is a little bit more complex than the previous examples. Each parenthetical expression is first evaluated to determine if it is true or false. Once determined to be true or false, that value is used in the overall xor statement. Because the first statement in this expression evaluates to true, the whole expression evaluates to true, and in turn the text is printed to the browser.

That's basically it for logical operators. Next you will take a look at the ternary operator.

Ternary Operator

All of the logical operators that were just discussed are considered to be *binary* operators, meaning that they perform their logic using two operands. The ternary operator is special. It uses three operands to perform a single operation. The ternary operator is used for quick one-liner if statements. You may have seen it before; it looks like this:

```
$x == 0 ? echo("yes") : echo("no")
```

The first of the three parts of the ternary operator is a Boolean condition before the question mark (?). The second of the three parts is a statement between the ? and the colon (:), which is executed if the condition in front of the ? evaluates to true; and a value after the colon, which is returned if the condition evaluates to false.

Note

For those of you who are familiar with C/C++, the ternary operator in PHP acts exactly like the ternary operator in C/C++.

As mentioned earlier, the ternary operator is a shortcut for an if...else statement. Take a look at how the following if...else statement is converted to using the ternary operator:

```
if($nKingStatus == 1)
{
    echo("check");
}
else
{
    echo("move = " . $from . " . " . $to);
}

// Now for the same statement using the ternary operator
$nKingStatus == 1 ? echo("check") : echo("move = " . $from . " . " . $to)
```

See how much shorter that is? You get the same results from one line of code that you did from four lines of code. However, the ternary operator doesn't do much for readability, so it is completely up to you as to how you would like to use it.

There is one little thing that I would like to mention. As you may have noticed in the previous chapters and in the previous example, I put a period (.) in between the strings in the example. All this does is join the strings together. The period, when used in between strings, is called the *string concatenation* operator.

Note

Concatenation is just a fancy word for joining, or adding, two objects together.

Let's move on to bitwise operators and then to some variable assignment shortcuts that will save you some typing.

Bitwise Operators

Before getting into the operators themselves, I want to give you a little lesson in binary representation, and how the computer stores numbers in memory. Otherwise none of what I am going to show you is going to make sense.

A binary number uses only 1s and 0s to comprise a number. Each 1 and 0 is referred to as a *bit*. It is assumed that when storing a 32-bit integer in memory that it is to use 4 bytes. There are 8 bits to a single byte. Using these 1s and 0s you can create any number. Take a look at Table 5.4.

Take a look at Figure 5.1 to see how you count a binary number.

As you can see, you really count from right to left, going by multiples of 2. So the first bit on the right-hand side is 1, then the next is 2, then 4, 8, 16, and so on.

Now that you have a basic understanding of binary representations, take a look at the bitwise operators you can use in PHP. There are six bitwise operators all together; four of them allow you to compare bits, and two of them allow you to shift bits either to the left or to the right. Take a look at Table 5.5 for all six of the bitwise operators.

One of the biggest reasons to understand bitwise operators is that they are used all the time in the programming of board games. In the next chapter you will create a chess game that uses bit-boards. If you were not clear about how to use bitwise operators, you would not understand how the program will work.

Table 5.4 Binary Number Representations

Binary Number	Corresponding Integer
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

* For simplicity's sake I used only 4 bits, or 1 byte, to represent numbers in this example.

0	0	0	0	0	0	0	0
...and so on	128	64	32	16	8	4	2

Figure 5.1 How to count binary numbers.

Table 5.5 Bitwise Operators

Operator	Name	Quick Example
&	And operator	11 (1011) & 13 (1101) = 9 (1001)
	Or operator	11 (1011) 13 (1101) = 15 (1111)
^	Exclusive Or	11 (1011) ^ 13 (1101) = 6 (0110)
~	Not operator	~11 (1011) = -12 (100000000001100)
>>	Shift bits to the right by	11 (1011) >> 2 = 2 (0010)
<<	Shift bits to the left by	11 (1011) << 2 = 44 (101100)

All that the bitwise operators do is perform operations analogous to the logical and, or, xor, and not on each set of bits. Imagine that the bits are lined up one on top of another. When you “&” two bits together, if they are both 1 (true `&&` true), the result is true, or the logical bit 1. Take a look at Figure 5.2.

See how the least significant bit in both numbers is set to 1. When they are “anded” together the result is true, or 1. But when the next bits are “anded” together, the result is false, or 0, because both bits are false to begin with. Now do the same exact thing with the “|” operator to see the differences. Figure 5.3 shows the results.

In this example you can see that the “|” operator evaluates to true anytime a true bit appears in the value. So when the least significant bit in both numbers (1 and 1) is “or’ed” the logical expression would be: true or true; the result is true. But when the next bits are

$$\begin{array}{r}
 1001 \quad (8) \\
 \&0101 \quad (5) \\
 \hline
 0001 \quad (1)
 \end{array}$$

Figure 5.2 Using the “&” bitwise operator.

$ \begin{array}{r} 1001 \quad (8) \\ 0101 \quad (5) \\ \hline 1101 \quad (13) \end{array} $
--

Figure 5.3 Using the “|” bitwise operator.

“or’ed” together (0 and 0) the logical expression would be false or false, so the result is false. The exclusive or (^) operator acts like the “|” operator but will evaluate to true only if one of the bits is set to 1 and the other bit is set to 0. So the expression

`1 ^ 1`

evaluates to 0, but the expression

`1 ^ 0`

evaluates to 1, because the expression can only be true one way.

The “~” (not) operator is very unique because it doesn’t operate on two bits. Instead it operates on only one bit at a time. Remember the example in Table 5.5 for the “~” operator?

`~11 (1011) = -12 (1000000000001100)`

This is taking the binary number 1011 (11 in real numbers) and making each bit the opposite number and making it negative. It has the same effect as multiplying a decimal number by negative one (-1) and subtracting 1. The reason for this is because the most significant bit in a 32-bit value tells the number whether or not it is negative. The “~” operator won’t be used that much, but it is handy to know.

The final two bitwise operators that PHP supports are the left shift (<<) and the right shift (>>) operators. Each of these operators take two values: the left value is the number that you want to shift bits on, and the second value is the number bits you are going to shift the first value by. For example:

`11 << 2`

This will take the decimal number 11, which is the binary number 1011, and shift it by two bits to the left. To make this example clearer I'll extend the number 11 to 16-bits.

```
0000000000001011 << 2 = 00000000101100
```

See how it shifted all the bits to the left by two, and then two trailing zeros were added to the binary number? Any bits beyond the 32-bit value fall off the left side and are lost, keeping the number a 32-bit value. The right shift operator performs the equivalent of the left shift operator but the opposite way.

```
0000000000001011 >> 2 = 0000000000000001
```

In this case zeros were added to the beginning of the value and the bits on the right side fell off the number.

Variable Assignment Shortcuts

Just like in C/C++, PHP has shortcut operators for assignment statements where the first operand is the variable and the result is stored in the same variable. Take a look at this example:

```
$a = $a + 25;  
$a += 25; // This is the same exact statement as the line above.
```

Take a look at Table 5.6 for a list of the variable assignment shortcuts available in PHP.

Table 5.6 Variable Assignment Shortcuts

Shortcut	Example	Equivalent
<code>+=</code>	<code>\$x += \$i</code>	<code>\$x = \$x + \$i</code>
<code>-=</code>	<code>\$x -= \$i</code>	<code>\$x = \$x - \$i</code>
<code>*=</code>	<code>\$x *= \$i</code>	<code>\$x = \$x * \$i</code>
<code>/=</code>	<code>\$x /= \$i</code>	<code>\$x = \$x / \$i</code>
<code>%=</code>	<code>\$x %= \$i</code>	<code>\$x = \$x % \$i</code>
<code>&=</code>	<code>\$x &= \$i</code>	<code>\$x = \$x & \$i</code>
<code> =</code>	<code>\$x = \$i</code>	<code>\$x = \$x \$i</code>
<code>^=</code>	<code>\$x ^= \$i</code>	<code>\$x = \$x ^ \$i</code>
<code>.=</code>	<code>\$x .= \$i</code>	<code>\$x = \$x . \$i</code>
<code>>>=</code>	<code>\$x >>= \$i</code>	<code>\$x = \$x >> \$i</code>
<code><<=</code>	<code>\$x <<= \$i</code>	<code>\$x = \$x << \$i</code>
<code>++</code>	<code>\$x++</code>	<code>\$x = \$x + 1</code>
	<code>\$x--</code>	<code>\$x = \$x - 1</code>

The increment operator `++` and the decrement operator `--` can appear before the variables, just as they can appear after the variables. The placement of the operator can change the order of events. For example, if the `++` is placed before the variable, the value is incremented and the new value is returned. But if the `++` is placed after the variable, then the variable is pre-incremented and then the value is assigned.

```
/* When used by itself, the increment operator has the same effect whether it appears  
before or after the variable */  
  
$x = 10;  
$x++; // x = 11  
  
$x = 10;  
++$x; // x = 11  
  
$x = 10;  
$y = $x++; // y = 10, and x = 11  
  
$x = 10;  
$y = ++$x; // y = 11, and x = 11
```

Caution

This can affect your loops, so make sure you understand the order of operations occurring here.

Before moving on to statements, I will cover operation precedence. I have mentioned it several times before and just want to make sure that you know what it means.

Operation Precedence

Precedence is simply the order in which operations will occur, just like in the previous example where the increment operator was before or after the variable. Precedence can change the value of the expected outcome. Every operator in PHP has certain precedence; Table 5.7 shows each operator with the highest precedence first.

Statements

What are statements? Statements provide the logic structure of your application. Conditional statements and loops give your programs the ability to make decisions based on limited rules.

Table 5.7 Operation Precedence

Operator	Operation
()	Precedence establishment
new	Object instantiation
[]	Array index access
!	Logical not
~	Bitwise not
++	Incrementation
--	Decrementation
@	Functional error suppression
*	Multiplication
/	Division
%	Modulus
+	Addition
-	Subtraction
.	Concatenation
<<	Bitwise left shift
>>	Bitwise right shift
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to
&	Bitwise &
^	Bitwise XOR
	Bitwise Or
&&	Logical AND
	Logical Or
:?	Conditional Ternary Operator
= += -= *= /= .= %= &= != ~= <<= >>=	Assignment
and	Logical AND
xor	Multiple XOR
or	Logical OR
,	Multiple evaluation

if Statements

The if statement is probably one of the most important features of every programming language. It allows you to execute lines of code only when the specified conditions are true. Remember all the logical operators discussed earlier in this chapter? This is where you get to put them to good use. You can make if statements as complex or as simple as you would like. I recommend making them fairly simple to read; otherwise when you go back to your code a month later it might take you a while to figure out what is going on. Take a look at the following example:

```
<?php
$bInitialized = 1;
if($bInitialized == 1)
{
?>
<table border=1 cellpadding=0 cellspacing=0 width=256 height=256>
<tr>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
</tr>
</table>
<?php
}
echo("This printed a row of alternating
colored blocks because the if statement evaluated to true");
?>
```

You can create branched conditions too, meaning that if one of the statements evaluates to false, you can drop into another code block to be executed. This is accomplished by using the if...else statement. Here is an example:

```
<?php
$bInitialized = 1;
if($bInitialized == 1)
{
?>
<table border=1 cellpadding=0 cellspacing=0 width=256 height=256>
<tr>
```

```
<td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
<td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
</tr>
</table>
<?php
echo("This printed a row of alternating
colored blocks because the if statement evaluated to true");
}
else
{
    echo("Sorry the program is not initialized");
}
?>
```

You can also test multiple conditions if one of the statements evaluates to false using the elseif keyword, like this:

```
<?php
if($bInitialized == 1)
{
    echo("The program is initialized");
}
elseif($bInitialized == 2)
{
    echo("Starting game");
}
elseif($bInitialized == 3)
{
    echo("The game is running");
}
else
{
    echo("Sorry the program is not initialized");
}
?>
```

Note

Make sure that when using the `elseif` keyword you do not put a space in between the `else` and the `if`. It is all one word in PHP, unlike in other languages such as C/C++.

PHP also offers an alternative syntax for the `if` statement, `if...endif`. You would normally use this syntax if you were printing large blocks of HTML in between PHP blocks of code. Although you could do the same things with the C/C++-like syntax. Take a look at the following example:

```
<?php
$bInitialized = 1;
if($bInitialized == 1):
?>
<table border=1 cellpadding=0 cellspacing=0 width=256 height=256>
<tr>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#ffffff">&nbsp;</td>
    <td align="left" valign="top" bgcolor="#000000">&nbsp;</td>
</tr>
</table>
<?php
endif;
?>
```

That's it for the `if` statement. Now you can move on to the `if` statement's brother, the `switch` statement.

The switch Statement

The reason I call the `switch` statement the `if` statement's brother is because the `switch` statement acts like a bunch of `if...elseif...else` statements. Take the following `if...elseif...else` statement for example:

```
<?php
if($bInitialized == 1)
{
    echo("The program is initialized");
}
```

```
elseif($bInitialized == 2)
{
    echo("Starting game");
}
elseif($bInitialized == 3)
{
    echo("The game is running");
}
else
{
    echo("Sorry the program is not initialized");
}
?>
```

In this example you are repeatedly checking the variable \$bInitialized for a certain value, even though the value of \$bInitialized does not change from line to line. To make this more efficient you would want to check the value of \$bInitialized only once. You can do that with the switch statement. Here is how you can convert the example above into a switch statement:

```
<?php
switch($bInitialized)
{
    case 1:
    {
        echo("The program is initialized");
        break;
    }
    case 2:
    {
        echo("Starting Game");
        break;
    }
    case 3:
    {
        echo("The game is running");
        break;
    }
    default:
    {
        echo("Sorry the program is not initialized");
    }
}
?>
```

The switch statement will compare the values of each case with the value of the variable, just like if you used an if...elseif...else statement, but as soon as it finds a match it executes only that block of code. If the switch statement does not find a match, then the default case is executed. Notice the breaks at the end of each case. These are very important because if you do not use a break, then the code will fall through to the next case. All the break does is tell the code to stop executing the block and continue to the next statement.

If you are familiar with other languages, such as C/C++ or Java, you will be very familiar with the switch statement. In fact, the syntax is exactly the same. However, PHP's switch statement is much more flexible for two major reasons: 1) Each case does not have to be a scalar value—it can be any value at all, and 2) You can use variables in each of your cases. The following example demonstrates using variables in a case statement:

```
<?php
switch(_SESSION["GAME_STATE"])
{
    case $bInitialized:
    {
        echo("The program is initialized");
        break;
    }
    case $bStarting:
    {
        echo("Starting Game");
        break;
    }
    case $bRunning:
    {
        echo("The game is running");
        break;
    }
    default:
    {
        echo("Sorry the program is not initialized");
    }
}
?>
```

Note

The only variable types that are not valid in a case statement are arrays and objects.

In this example there is a variable in the global session data called GAME_STATE. When the switch statement executes it will check the value of each of the case variables against the variable GAME_STATE and when it finds a value that matches, it will execute the block of code.

while and do...while Loops

A while loop is the simplest loop in PHP; it acts similar to an if statement. A while loop evaluates a Boolean expression. If the expression is true, then the code inside the loop is executed. To break out of a while loop you have to make the Boolean expression you are testing for evaluate to false, or explicitly use the break statement. Otherwise you will end up in an infinite loop, which is not good.

```
<?php  
$i = 0;  
while($i < 8)  
{  
    if( ($i %2) != 1)  
        $color = "#ffffff";  
    else  
        $color = "#000000";  
    echo("<td align=left valign=top bgcolor=" . $color . ">&nbsp</td>");  
    $i++;  
}  
?>
```

Note

The while loop also offers an alternative syntax exactly like the if statement does.

```
<?php while: ?>HTML HERE<?php endwhile; ?>
```

Let's say you want the code in the block to execute on the first time, and if a Boolean expression evaluates to true you would like to execute the code block again. You can accomplish this with the do...while loop.

The do...while loop will always execute the loop at least once. Then if the Boolean expression evaluates to true, it will execute the loop until the Boolean expression evaluates to false.

```
<?php  
$i = 0;  
echo("<SELECT NAME=level>");  
do  
{  
    echo("<OPTION VALUE=" . $i . ">Level" . $i . "</OPTION>");
```

```

} while(++$i < $nLevels);
echo("</SELECT>");
?>

```

In this example the option, Level 0, will always be printed even if the variable \$nLevels equals 0. Once the variable \$i_equals \$nLevels the while loop will stop and move on and print the last line of the HTML select statement.

The for Loop

The for loop takes three expressions separated by semicolons. The first expression is an assignment statement to initialize the controlling loop variable; the second is a Boolean expression. As long as the Boolean expression evaluates to true, the for loop will execute. The third and final expression is any statement that will execute at the end of each iteration of the loop.

Note

The for loop in PHP has the same exact syntax as in C/C++.

Now take the do...while loop used in the last example and convert it into a for loop.

```

<?php
echo("<SELECT NAME=level>");
for($loopCount = 0; $loopCount < $nLevels; $loopCount++)
{
    echo("<OPTION VALUE=" . $loopCount . ">Level" . $loopCount . "</OPTION>");
}
echo("</SELECT>");
?>

```

This produces the exact same results as the do...while loop but in fewer lines of code. You may use whatever loops you feel comfortable with, but I generally use a while loop when the number of loops is unknown, and a for loop when the number of loops are known.

Note

The for loop also offers an alternative syntax like the if statement and the while loop.

```
<?php for($i = 0; $i < 10; $i++): ?> some HTML <?php endfor; ?>
```

Functions

What is a function? A function is a block of code that can be defined once and reused in multiple parts of the program. This is a very handy tool for a programmer to have. With-

out functions you would be forced to rewrite large chunks of your program every time you needed to repeat an operation. Without functions you would most likely go insane or at least you would not be coding.

Functions often take multiple parameters; a parameter is nothing more than a localized variable that the function will operate on. These parameters allow you to pass data in and out of your functions. But a function does not have to take any parameters at all. Suppose you have a function to write out the header of your HTML document; it would look something like this:

```
function WriteHTMLHeader()
{
    echo("<HTML>\n");
    echo("<HEAD><TITLE>My Application</TITLE></HEAD>\n");
    echo("<BODY bgcolor=#ffffff>\n");
}
```

You declare functions by using the keyword `function` followed by the name you want for the function. In the above example the name of the function is `WriteHTMLHeader`. Technically this function will not do anything at all until you invoke it from another point in the code. To invoke a function, you simply call it by name followed by any parameters it has. If the function doesn't have any parameters, you simply leave the parentheses blank. Here is an example of calling the `WriteHTMLHeader` function.

```
// some code here
WriteHTMLHeader();
// more code
```

You could call this function several times in a row if you like and it would print the same HTML header to the browser every time.

Passing Parameters to a Function

Passing parameters to a function is extremely simple. All you need to do is declare a variable for each parameter you want to receive. For example, say you want a function that pops up a JavaScript error message with whatever message you pass to it.

```
function ErrorPopup($msg)
{
    // Create a JavaScript alert
    echo("<SCRIPT LANGUAGE='JavaScript'>\n");
    echo("<!--\n");
    echo("alert(\"$msg\");\n");
    echo("//-->\n");
    echo("</SCRIPT>\n");
}
```

Now when you invoke this function you will write the following line of code:

```
ErrorPopup("This is an error message!");
```

As you can see, this time when you invoked the function you passed it a parameter. When the function was called, the value of the function variable \$msg was set to the string, "This is an error message!", so any time you reference \$msg in the function its value will be whatever you passed in.

In the previous example the variable was passed by value, meaning that anything done to the variable in the function did not affect the value of what was passed to it. For example:

```
<?php
function square($num)
{
    $num *= $num;
    return $num;
}

$someNum = 6;
echo("Variable before going into the function = " . $someNum);
$results = square($someNum);
echo("<br>Variable after going into the
    function = " . $someNum . "<br>Results of
    function = " . $results);
?>
```

The results of this chunk of code look like Figure 5.4.

See how the value of \$someNum was not affected by the operations that the function square() did? Suppose that you want the function to change the value of the variable you are passing in. How would you do that? You would pass the variable in by reference. To pass a variable into a function by reference you need to have an ampersand (&) in front of the parameter. Let's take the same example from above and make the function change the value of the variable by passing it by reference.

```
<?php
function square(&$num)
{
    $num *= $num;
}

$someNum = 6;
echo("Variable before going into the function = " . $someNum);
square($someNum);
echo("<br>Variable after going into the function = " . $someNum);
?>
```

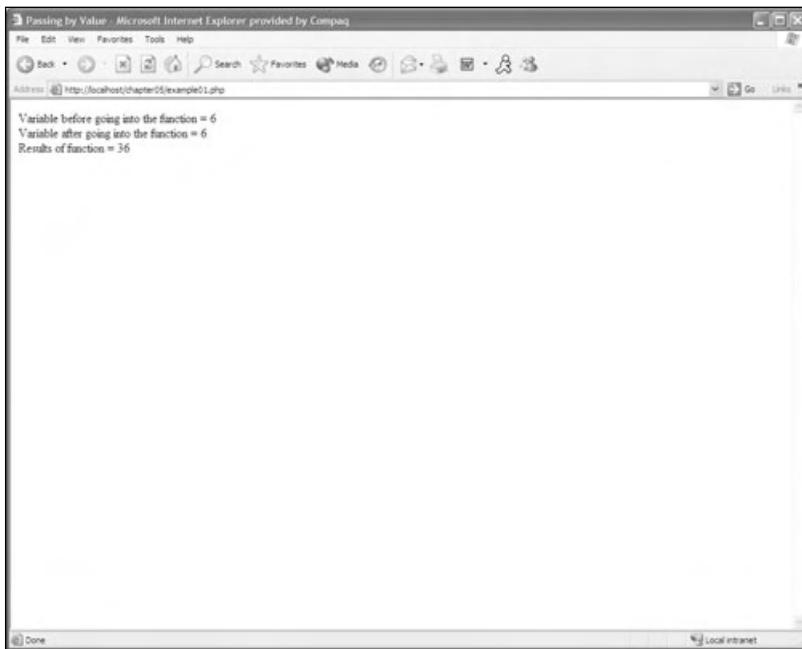


Figure 5.4 Results of passing variables by value.

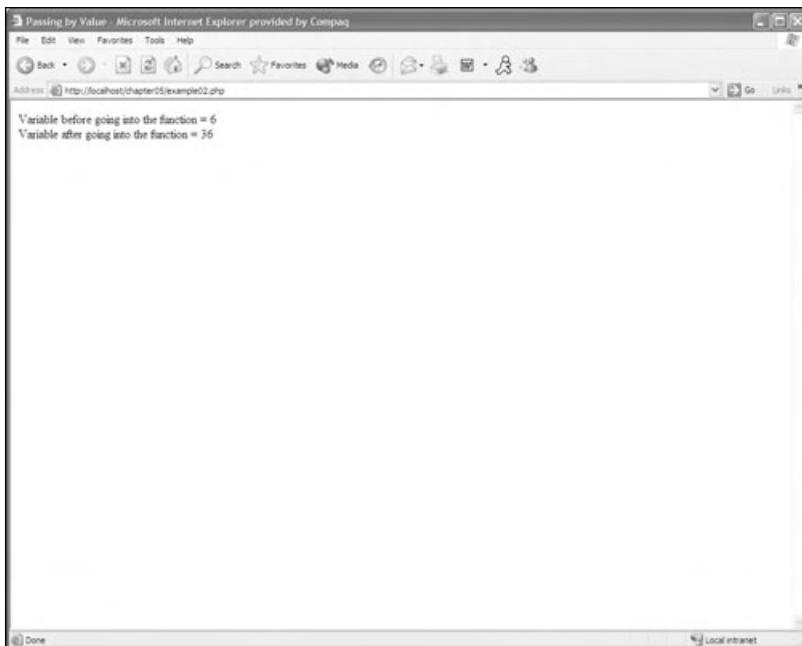


Figure 5.5 Results of passing variables by reference.

See how the value of \$someNum changed? Its value is the result from the operations inside the function. Passing variables by reference is a very handy way to return results for multiple variables because a function can return only one value if you are using the return keyword.

Recursion

PHP functions also support recursion. Recursion is simply when a function calls itself. One of the easiest ways to explain recursion is just to show you; take this, for example:

```
function power($base, $exponent)
{
    if($exponent)
    {
        return $base * power($base, $exponent - 1);
    }
    return 1;
}
```

Notice how the return statement is calling the power function, but it is subtracting one from the variable \$exponent every time it is called. Eventually \$exponent will hit zero and the if statement will not execute. At this point is when the recursion ends. Take an in-depth look at how this works.

```
echo(power(2,3));
```

1. The function power is called with the base set to 2 and the exponent set to 3.
2. Next, the \$exponent is tested; since it is non-zero it succeeds and proceeds to the next line.
3. The return statement calls the power function again, except this time the base is 2 and the exponent is also 2.
4. The \$exponent variable is tested again. It is still a non-zero value.
5. Power is called again with the value 2 as the base and 1 as the exponent.
6. The exponent variable is tested again; it is still a non-zero value.
7. Power is called a final time with the value 2 as the base and 0 as the exponent.
8. This time the test fails and the function returns 1 to the third invocation of the function.
9. 1 is now multiplied by the base and the third invocation of the function returns 2 to the second invocation of the function.

10. 2 is now multiplied by the base value, giving us 4, and 4 is returned to the first invocation of the power function.
11. 4 is now multiplied by the base, giving us the final value of 8, and 8 is returned from the power function back to your program.

Pretty confusing, huh? Recursion can be a nasty beast but you can do some pretty cool things with it.

The Magic of Including Files

Now that you have all this cool knowledge about how to make functions, you need a way to include libraries (so to speak) in your PHP pages. Basically what you can do is make a file full of common functions that you use all the time and put them in all of your games. For instance, if you take a look at the CD, I have included a PHP file (common.php) that has some base functions that will be used in all of the games in this book.

PHP gives you two ways to include files in your application. You can use the require statement or the include statement. When the PHP interpreter encounters the require statement it puts the file in the code and it is now generally available to your page. The include statement evaluates and executes the code each time the include statement is encountered. This allows you to have dynamic includes in your files. For example:

```
for($loopCounter = 0; $loopCounter < 5; $loopCounter++)  
{  
    include("file" . $loopCounter . ".php");  
}
```

If you did the above example with the require statement, then the only file that would be included would be file4.php. Instead of worrying about which one is better I would just always use the include statement. There is no performance hit and you can be a lot more flexible in your coding.

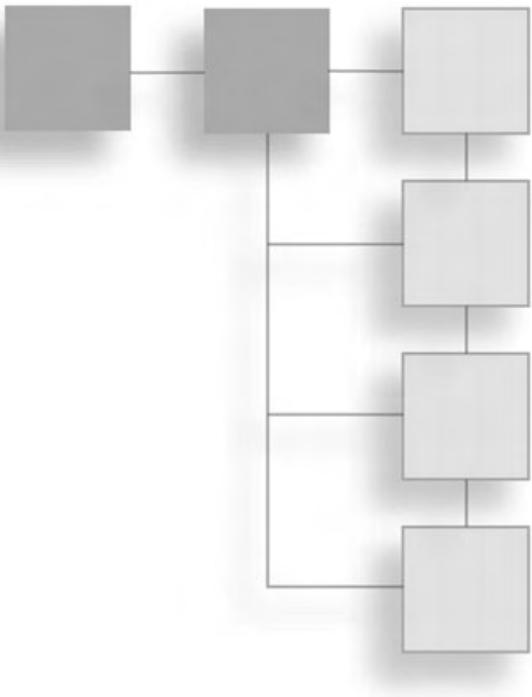
Note

PHP assumes the include files are in the directory specified by the include_path directive in the php.ini file. If your include files are not in this directory you need to specify the full path along with the filename to include it.

Conclusion

You have covered a ton of information in this chapter. You now know all about operators and how to control the flow of your code with sound logic. You also know how to create blocks of reusable code using functions. You even know how to make a mini-library and include it in all of your PHP pages.

Next up: arrays, and then your first PHP game!



PART III

S, GAMES, GRAPHICS

CHAPTER 6

Arrays!	103
---------	-----

CHAPTER 7

Playing with Chess and Databases	133
----------------------------------	-----

CHAPTER 8

GD Graphics Overview	157
----------------------	-----

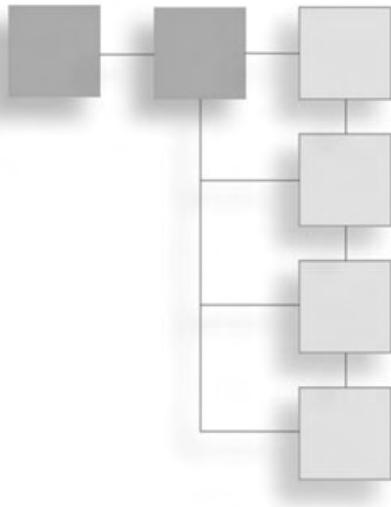
CHAPTER 9

Creating Battle Tank and Using Dynamic Terrain	193
--	-----

This page intentionally left blank

CHAPTER 6

ARRAYS!



- Initializing Arrays
- Using Strings for Indexes
- Looping through Sequential Arrays
- Looping through Non-Sequential Arrays
- Multi-Dimensional Arrays
- Sorting Arrays
- Your First PHP Game

In Chapter 4, when you were making the form processing example, the check boxes were named `nEquipmentID[]`. Those two brackets mean that it is an array. An array consists of several elements, each of which has a value. You can access each element in the array by using an index. In PHP your index can be either an integer or a string, which allows a lot of flexibility in your code.

Note

As in C/C++, arrays in PHP are zero based, meaning that the first index in the array starts at 0.

Take a look at Figure 6.1. This representation should make it clear how an array stores its data.

\$board[0]	\$board[1]	\$board[2]	\$board[3]
"A"	"B"	"C"	"D"

Figure 6.1 How an array stores data.

In this case the array is named \$board, and it has four elements. The first element of the array, \$board[0], is equal to "A". The second element of the array, \$board[1], is equal to "B," and so on. So how do you create arrays?

Initializing Arrays

There are several ways that you can initialize arrays. You could simply list the variable with the value you would like to add to the array, like this:

```
$board[] = "A":  
$board[] = "B":  
$board[] = "C":  
$board[] = "D":
```

Because you did not specify an index, the elements are automatically added in sequential order. You could have also specified what index you would like the element to appear in, like this:

```
$board[0] = "A":  
$board[1] = "B":  
$board[5] = "C":  
$board[7] = "D":
```

This is more impractical, but valid all the same. Usually when assigning elements to an array you should do it in a sequential order. This way there is no confusion about where the element is. However, there are times when you will want to create a hash key and use that for your index.

If you ever assign items in a non-sequential order and then later add an element to the array without specifying the index it should appear in, then the element will be automatically added to the next highest index. For example:

```
$board[0] = "A":  
$board[1] = "B":
```

```
$board[60] = "C":  
$board[] = "D": // "D" would be added to index 61, not 2.
```

The final way to declare an array is to use the `array()` function. You simply pass the values you want in the array as parameters to the function.

```
$board = array("A", "B", "C", "D");
```

This will create an array exactly like the first example did. “A” is the first element in the array with an index of 0. “B” is the second element in the array with an index of 1, and so on. If you wish to change the index of an element using the `array()` function, you can do so like this:

```
$board = array(1 => "a", "b", 7 => "c", "d");
```

In the example above, “a” will be at index 1, “b” will be located at index 2, “c” will be located at index 7, and “d” will be located at index 8. You can put the `=>` operator before any element of the array to change its index.

Using Strings for Indexes

As mentioned earlier in this chapter, you can use strings as indexes into your arrays. The global variable `_SESSION` in PHP uses strings as its indexes. The syntax for accessing an array that uses strings as an index looks like this:

```
_SESSION["gTurn"];
```

Creating an array that uses strings for an index is very simple. Recall the first example I used to initialize an array with elements in a specific index.

```
$board[0] = "A":  
$board[1] = "B":  
$board[5] = "C":  
$board[7] = "D":
```

Instead of using an integer as an index you would simply replace it with a string, as the following example demonstrates:

```
$board["element1"] = "A":  
$board["element2"] = "B":  
$board["element3"] = "C":  
$board["element4"] = "D":
```

You can also use the => operator along with the array() function to create an array with strings as indexes.

```
$board = array("element1" => "A", "element2" => "B",
    "element3" => "C", "element4" => "D");
```

If you use strings as indexes into your array, you have to access the array with the specified string indexes. In other words, if you typed in the following code:

```
<?php
$board = array("element1" => "A", "element2" => "B",
    "element3" => "C", "element4" => "D");
echo($board["element1"]);
echo($board[0]);
?>
```

You would get an error when the PHP interpreter tried to print the last line, echo(\$board[0]), that looks like Figure 6.2.

Looping through Sequential Arrays

The easiest way to loop through sequential arrays is to use the for loop. I know you are probably thinking, “How do I know how many elements are in the array?” You can access

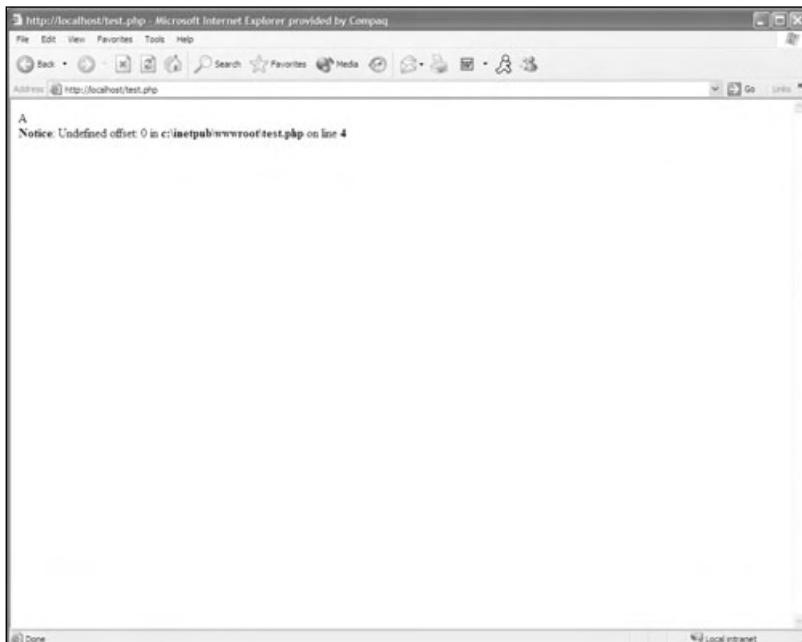


Figure 6.2 Trying to access a string-indexed array with an integer.

the number of elements in the array by using the count() function. Your for loop should look like the following:

```
<?php  
$board = array("a", "b", "c", "d");  
for($index = 0; $index < count($board); $index++)  
{  
    // Print each element on its own line  
    echo("Index = " . $index . ", Element = " . $board[$index] . "<BR>");  
}  
?>
```

First you initialize your array with all the elements you need. Then, in the first statement of the for loop, you initialize your indexing variable (\$index). The second statement of the for loop is your Boolean expression. This tells the loop when to stop; in this case the for loop will stop when it reaches one less than the total number of elements in the array \$board. Why don't you specify to stop when the loop reaches the total count of the elements? Because you have to remember that PHP uses a starting index of zero for arrays. If you went to the total count of the elements you would get an error when you tried to print the value because that index does not exist in the array. The third and final statement of the for loop simply increments the indexing variable every time the loop restarts. The results should look like Figure 6.3.

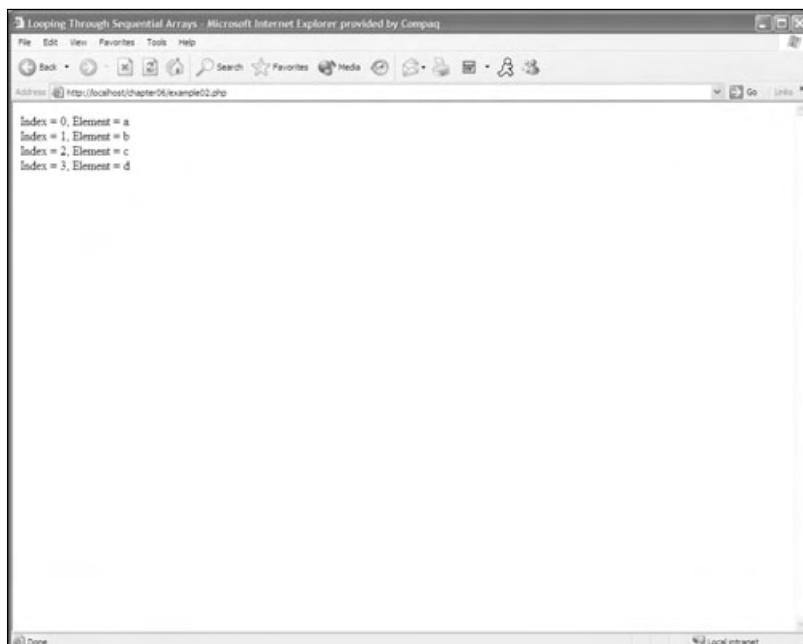


Figure 6.3 Results of looping through a sequential array.

Looping through the array with a for loop is perfect, as long as you know that the array is zero-based and ordered sequentially. So how do you loop through an array that is not ordered sequentially?

Looping through Non-Sequential Arrays

This is a perfect time to use a while loop. Of course, you will be using the while loop in conjunction with three other functions: `reset()`, `list()`, and `each()`. The `reset` function sets the current index of the array back to the first element available. It takes one argument, and that is the array that you would like to reset.

```
reset($someArray);
```

The `list` function takes two parameters: a variable to store the index of the element and a variable to store the value of the element.

```
list($index, $value);
```

The function `each()` takes one argument, and that is the array you wish to iterate on.

```
each($someArray);
```

Take a look at the following example of how to use these three functions together in a while loop to iterate through a non-sequential array.

```
<?php
$board = array("a", 7 => "b", 25 => "c", 50 => "d");
reset($board);
while(list($index, $value) = each($board))
{
    echo("Index = " . $index . ", Element = " . $value . "<BR>");
}
?>
```

The results of the example above appear exactly like Figure 6.3, even though the indexes in this particular array are not in a sequential order. Calling the `reset()` function in this example is quite unnecessary because, since the array has just been created, the current index is obviously the first available element. But it is good practice to get into, because you might end up trying to loop through an array and it could be starting in the middle of it, and that would be no good.

PHP also offers several other functions to operate on arrays, such as current(), key(), next(), and prev() to name just a few. For a complete list of functions that PHP supports, please refer to Appendix B. Take a look at the following example and Figure 6.4 to see what the key() and current() functions do:

```
<?php  
$board = array("a", "b", "c", "d");  
$index = key($board);  
$value = current($board);  
echo("Index = " . $index . ", Element = " . $value . "<BR>");  
?>
```

Note

Key and index mean exactly the same thing; the function in PHP is called key() instead of index().

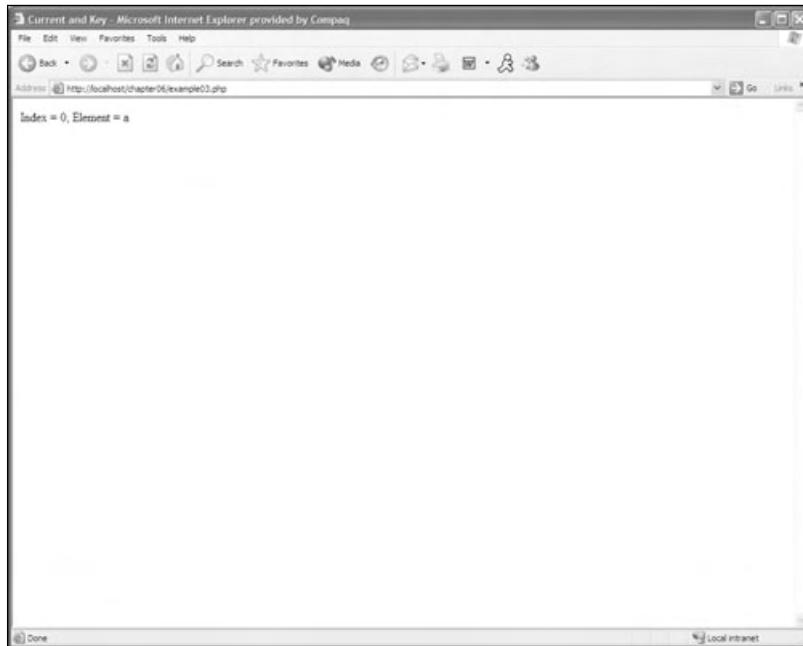


Figure 6.4 An example of the key() and current() functions.

Earlier I mentioned `next()` and `prev()`. These two functions are another way to navigate through an array. As you might imagine, `next()` goes to the next available index in the array, and returns the element. `prev()` does the same thing but instead of going to the next available index, it goes to the previous index. Take a look at the following loop:

```
<?php
$board = array("a", "b", "c", "d");
reset($board);
for($i = 0; $i < 4; $i++)
{
    $index = key($board);
    $value = current($board);
    next($board);
    echo("Index = " . $index . ", Element = " . $value . "<BR>");
}
```

This will print the same results to the browser as Figure 6.3 did, but it does so in a different way. But be careful if you use the `next()` and `prev()` functions. Remember I said it returns the value of the element of the next index in the array? Well, what if the value is 0? Take a look at this example:

```
<?php
$board = array(3, 0, 2, 4);
do
{
    $index = key($board);
    $value = current($board);
    echo("Index = " . $index . ", Element = " . $value . "<BR>");
} while(next($board));
?>
```

This loop will stop when it gets to the second element of the array because the `while` statement will evaluate to false. So be very careful if you use the `next()` and `prev()` functions. If you ever run into a loop that is stopping abruptly, take a look to make sure that your Boolean expression isn't being evaluated to false.

Multi-Dimensional Arrays

Creating multi-dimensional arrays in PHP is very similar to creating multi-dimensional arrays in C/C++. In C/C++ you have support for single-dimension arrays, but you create a multi-dimensional array by nesting an array as an element in a parent array. This is

exactly how you create multi-dimensional arrays in PHP. You make an array that holds an array.

```
<?php  
$board = array("Row1" => array("a", "b"), "Row2" => array("c", "d", "e"));  
echo($board["Row1"][0]); // Prints "a"  
echo($board["Row2"][2]); // Prints "e"  
?>
```

One great use for multi-dimensional arrays is that you can represent a game board using a two-dimensional array. Then each position in the game board, at say location e1, would hold a value that would tell you if there is a piece at that square or not. Or you could use a loop, like in the following example, to print the game board:

```
<?php  
$board = array(array("r", "k", "b", "q", "k", "b", "k", "r"),  
array("p", "p", "p", "p", "p", "p", "p", "p"),  
array("nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;",  
    "nbsp;", "nbsp;"),  
array("nbsp;", "nbsp;", "nbsp;", " ", " ",  
    " ", "nbsp;"),  
array("nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;",  
    "nbsp;"),  
array("nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;", "nbsp;"),  
array("p", "p", "p", "p", "p", "p", "p", "p"),  
array("r", "k", "b", "q", "k", "b", "k", "r"));  
  
echo("<table border=1 cellpadding=2 cellspacing=0 width=250>");  
for($yIndex = 0; $yIndex < count($board); $yIndex++)  
{  
    echo("<tr>");  
    for($xIndex = 0; $xIndex < count($board[$yIndex]); $xIndex++)  
    {  
        echo("<td align=center valign=middle>" .  
            $board[$yIndex][$xIndex] . "</td>");  
    }  
    echo("</tr>");  
}  
echo("</table>");  
?>
```

The results of this loop look like Figure 6.5.

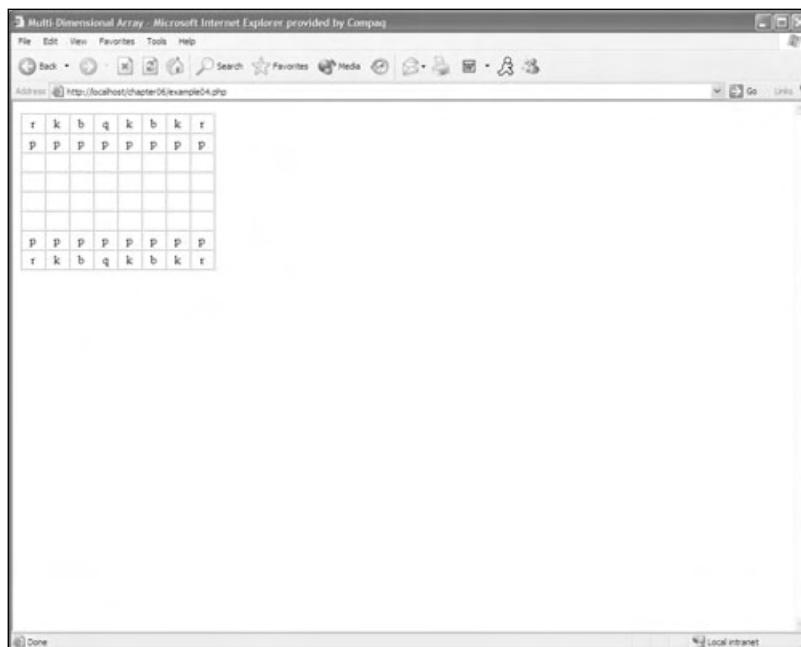


Figure 6.5 Results of looping through a multi-dimensional array.

Sorting Arrays

PHP provides tons of functions with which you can manipulate the arrays. Some of the handiest are the sorting functions. Take a look at Table 6.1 to see the list of sorting functions.

Table 6.1 Sorting Functions

Function Name	Description
sort()	Sorts arrays in alphabetical order.
asort()	Sorts arrays in alphabetical order without changing indexes.
rsort()	Sorts arrays in reverse order.
arsort()	Sorts arrays in reverse order without changing indexes.
ksort()	Sorts arrays by key values.
krsort()	Sorts arrays by key values in reverse order.
usort()	A customizable sort function.
uasort()	A customizable sort function that keeps string indices.
uksort()	A customizable sort function that sorts by key.

*A complete list of PHP functions is available in Appendix B.

Now take a closer look at each of these sorting functions. The `sort()` function sorts the elements in the array by numeric and then alphabetical order. Numbers come first, then punctuation marks, and finally letters. While the function is sorting, it reassigned the indexes to reflect the new order.

```
<?php
$items = array("Sword", "Medpac", "Advanced Medpac",
               "Armor", "Blaster", "Shotgun");

// Print the unsorted items
for($index = 0; $index < count($items); $index++)
{
    echo("Index = " . $index . ", " . $items[$index] . "<BR>");
}

// Print the sorted items
sort($items);
for($index = 0; $index < count($items); $index++)
{
    echo("Index = " . $index . ", " . $items[$index] . "<BR>");
}
?>
```

The results of this example should look like Figure 6.6.

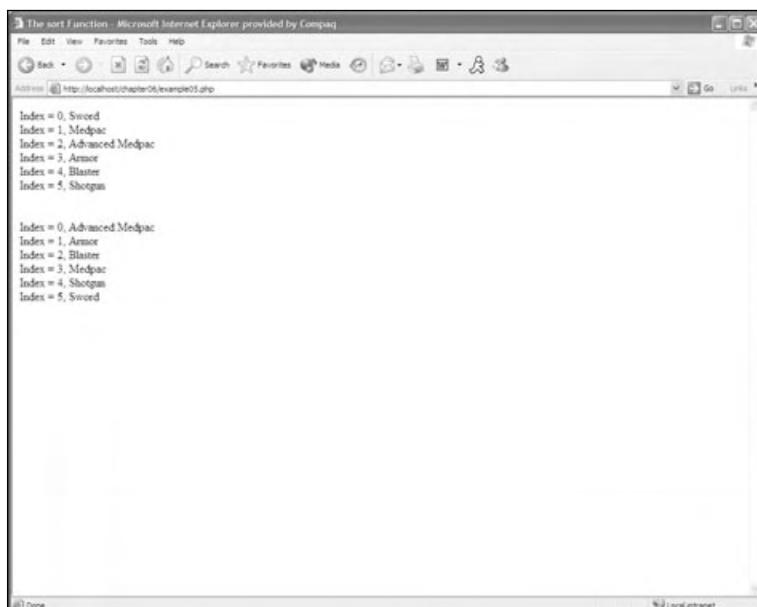


Figure 6.6 The `sort()` function in action.

Take a look at what happens when you have an array with specified indexes when you use the sort function.

```
<?php  
$items = array("Item1" => "Sword", "Item2" => "Medpac", "Item3" => "Advanced Medpac",  
    "Item4" => "Armor", "Item5" => "Blaster", "Item6" => "Shotgun");  
  
// Print the unsorted items  
while(list($index, $value) = each($items))  
{  
    echo("Index = " . $index . ", " . $value . "<BR>");  
}  
echo("<br>");  
// Print the sorted items  
sort($items);  
while(list($index, $value) = each($items))  
{  
    echo("Index = " . $index . ", " . $value . "<BR>");  
}  
?>
```

If you use the sort function with specified indexes, like in the example above, it will lose its indexes. Take a look at Figure 6.7 to see the results.

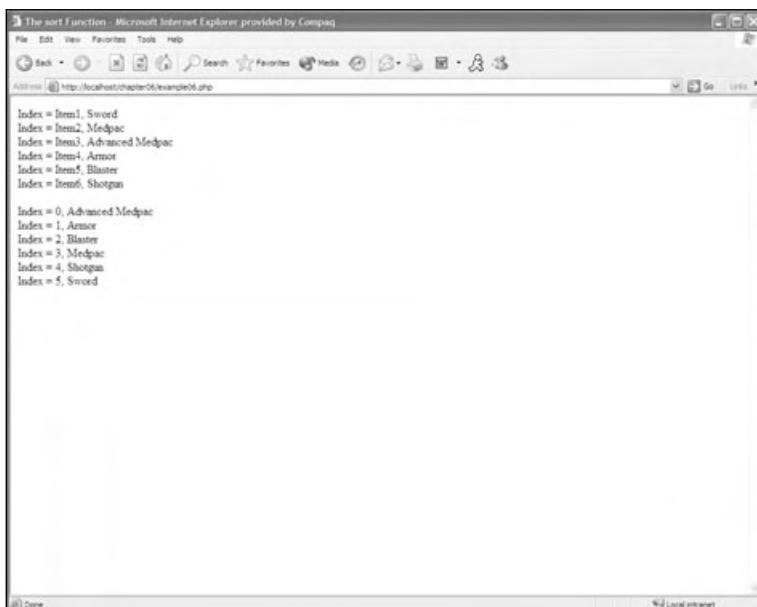


Figure 6.7 The sort() function with specified indexes.

To avoid losing your specified string indices you need to use the `asort()` function. This will sort the array and its elements without taking away your string indices. If you wanted to sort this array in reverse you would use the `rsort()` function. However, the `rsort()` function will also lose specified string indices, so to sort an array in reverse with string indices you need to use the `arsort()` function. You can also sort by your string indices by using `ksort()`.

```
<?php
$items = array("Item6" => "Sword", "Item5" => "Medpac", "Item3" => "Advanced Medpac",
"Item4" => "Armor", "Item2" => "Blaster", "Item1" => "Shotgun");

// Print the unsorted items
while(list($index, $value) = each($items))
{
    echo("Index = " . $index . ", " . $value . "<BR>");
}
echo("<br>");
// Print the sorted items
ksort($items);
while(list($index, $value) = each($items))
{
    echo("Index = " . $index . ", " . $value . "<BR>");
}
?>
```

The results of this example look like Figure 6.8.

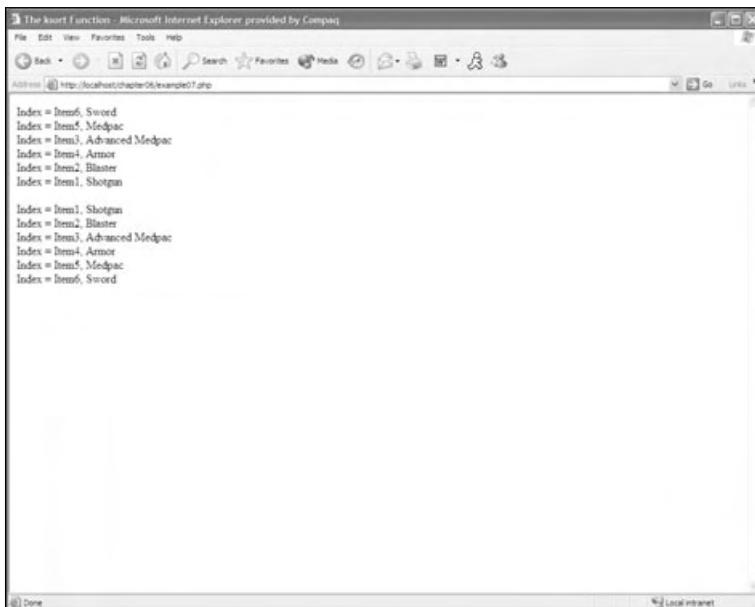


Figure 6.8 The `ksort()` function in action.

Note

You can sort in reverse order by key by using the `krsort()` function.

The last of the sorting functions for arrays has to be the coolest. With `usort()` you can organize your arrays however you want. The `usort()` function takes two arguments: the first is the array you would like to sort, and the second is a name of a function that contains your own sorting logic.

```
usort(someArray, someFunction);
```

The following example will sort the array `$items` by the length of each string:

```
<?php
function SortByLength($element1, $element2)
{
    $lengthOfElement1 = strlen($element1);
    $lengthOfElement2 = strlen($element2);
    if($lengthOfElement1 == $lengthOfElement2)
        return 0;
    return ($lengthOfElement1 < $lengthOfElement2) ? -1 : 1;
}

$items = array("Item6" => "Sword", "Item5" => "Medpac", "Item3" => "Advanced Medpac",
"Item4" => "Armor", "Item2" => "Blaster", "Item1" => "Shotgun");

// Print the unsorted items
while(list($index, $value) = each($items))
{
    echo("Index = " . $index . ", " . $value . "<BR>");
}
echo("<br>");
// Print the sorted items
usort($items, 'SortByLength');
while(list($index, $value) = each($items))
{
    echo("Index = " . $index . ", " . $value . "<BR>");
}
?>
```

The results of the above example are displayed in Figure 6.9.

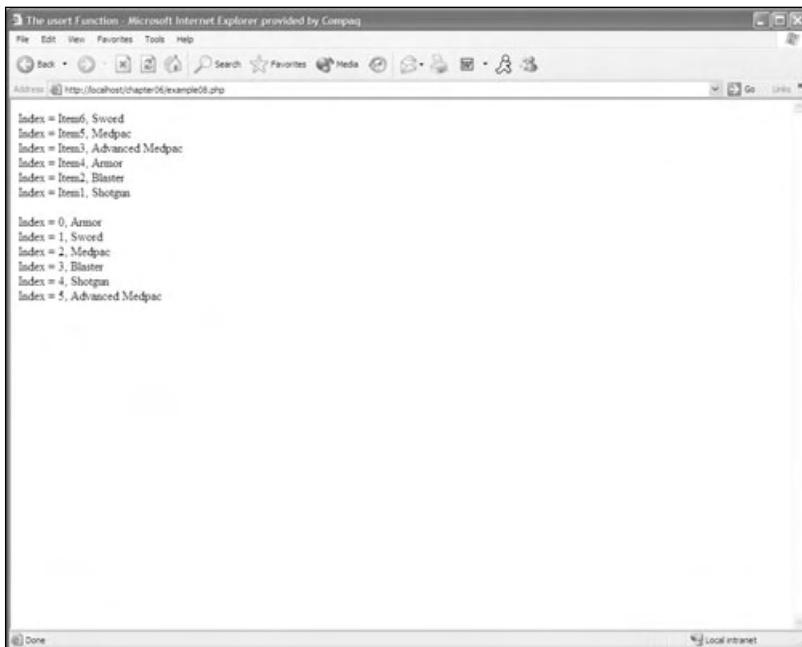


Figure 6.9 The usort() function in action.

Caution

Make sure you pass the function name to usort() as a string; otherwise you will get a warning display in the middle of your page.

Your First PHP Game

Now it's time to program your first game in PHP! You will start off with a simple game of tic-tac-toe. This game will use all the aspects that you have learned in all the previous chapters. It contains session variables, arrays, functions, and PHP working together with HTML.

The first step you need to take to set up a new game is to create a directory for it inside your Web server. For IIS the directory would be C:\Inetpub\wwwroot, and if you are using an Apache Web server the directory would be /usr/web. If you are on Windows the Apache directory would look something like C:\Apache depending on where you installed Apache. You can name the directory whatever you like. I am going to name it tictactoe.

Now that you have a directory set up you can create a new PHP file. I recommend following a regular naming scheme. The default page in a directory should be named index.php or default.php. You can use whichever you like. Since the game will only be in one file I will just name it index.php.

The next step you need to take is to define all of your constants and globals, and include the common functions file that you use in all of your games. The first set of constants that you need is four game states. The first of the four states is to tell you when the game is starting, the second lets you know when the game is in play, the third tells you if someone has won the game, and the fourth and final state tells you if someone has lost the game.

```
// Includes
include("common.php");
// Game States
define("GAME_START", 0);
define("GAME_PLAY", 1);
define("GAME_WIN", 2);
define("GAME_OVER", 3);
```

For tic-tac-toe, the only other two defines needed are one for the “X” image and one for the “O” image.

```
// Images
define("X_IMAGE", "images/X.gif");
define("O_IMAGE", "images/O.gif");
```

The only reason you define these is because it makes it easy if you want to change the images later, plus it saves you some typing when you create the render function to display your board to the browser. You will also need three global variables for all of your functions to use. The first stores what state the game is in, the second stores the state of the board, and the third stores the difficulty level of the computer A.I.

```
// Globals
global $gGameState;
global $gBoard;
global $gDifficulty;
```

Now that the global information for the game is defined, you will set up the HTML framework for the game. The HTML framework is the form you will use to get user input for your game. The following is all the HTML you will need for the game:

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
```

```
<head>
    <title>Tic-Tac-Toe</title>
    <link rel="stylesheet" href="style.css" type="text/css">
</head>
<body>

<form action="index.php" method="post">
<input type="hidden" name="turn" value="<? printf($turn) ?>">
    <?php WriteTableHeader(); ?>
    <div align="center">
        <input type="submit" name="btnNewGame" value="New Game">&ampnbsp&ampnbsp&ampnbsp;
        <b>Difficulty Level</b>
        <select name="dlDifficulty">
            <option value="1">Easy</option>
            <option value="2" SELECTED>Normal</option>
            <option value="3">Not-Likely</option>
        </select><br><br>
        <?php
            // Render the game
            Render();
        ?>
    </div>
    <?php WriteTableFooter(); ?>
</form>

</body>
</html>
```

Note

Remember that all of this code is provided for you on the CD.

The framework for the game is very simple; the form is redirecting back to itself. Then the header for the cool table layout is written. After that, some form elements are created. One allows the user to start a new game, and the drop-down box is used to choose the difficulty level of the computer A.I. After all that, the render function is called. Because you don't yet have a render function, you'll need to create one now.

The render function needs to determine what state the game is in by using the global game state variable that was declared earlier. If the state of the game is starting, then the render function needs to start a new game and change the state to playing. If the state of the game

is playing then the render function needs to take the user input, process it, calculate the move for the computer, and update the board. If one of the players has won the game, the render function needs to tell the user that he either has won or lost the game.

As you can see, the render function needs to do a ton of work. To accomplish all these tasks the render function will need some helper functions. You will need a function to start a new game, a function to draw the board, a function to check for a win, a function to check to see if the board is full, and, finally, a function to end the game and free your session. Let's start off with the render function itself and then continue on to the child functions that render() will call.

```
function Render()
{
    global $gGameState;
    global $gBoard;
    global $gDifficulty;

    switch($gGameState)
    {
        case GAME_PLAY:
        {
            // Get the move if the user made one
            if($_POST['btnMove'] != "")
            {
                $gBoard[$_POST['btnMove']] = "x";
                $_SESSION['gBoard'] = $gBoard;
            }

            // Check for a win
            if(CheckWin() == "X")
            {
                $gGameState = GAME_WIN;
                Render();
                return;
            }

            // Check to see if the board is full
            if(CheckFull() == 1)
            {
                $gGameState = GAME_OVER;
                Render();
                return;
            }
        }
    }
}
```

```
}

// Compute the computer's move if we can still move
if($gGameState == GAME_PLAY && $_POST['btnMove'] != "")
{
    if($gDifficulty == 1)
    {
        ComputerRandomMove();
    }
    elseif($gDifficulty == 2)
    {
        $computerMove = ComputerMove();
        if($computerMove == "")
        {
            ComputerRandomMove();
        }
        else
        {
            $gBoard[$computerMove] = "o";
            $_SESSION['gBoard'] = $gBoard;
        }
    }
    elseif($gDifficulty == 3)
    {
        $computerMove = ComputerMove();
        if ($computerMove == '')
        {
            if($gBoard[4] == '')
                $computerMove = 4;
            elseif($gBoard[0] == '')
                $computerMove = 0;
            elseif($gBoard[2] == '')
                $computerMove = 2;
            elseif($gBoard[6] == '')
                $computerMove = 6;
            elseif($gBoard[8] == '')
                $computerMove = 8;

            if($computerMove == '')
                ComputerRandomMove();
        }
    }
}
```

```
}

// Check for a win
if(CheckWin() == "0")
{
    $gGameState = GAME_OVER;
    Render();
    return;
}

// Check to see if the board is full
if(CheckFull() == 1)
{
    $gGameState = GAME_OVER;
    Render();
    return;
}

// Draw the board
DrawBoard();
break;
}

case GAME_WIN:
{
    EndGame();
    printf("<br><br><br><img src=\"images/youWin.jpg\""
        " border=\"0\">");
    break;
}

case GAME_OVER:
{
    EndGame();
    printf("<br><br><br><img src=\"images/gameOver.jpg\""
        " border=\"0\">");
    break;
}
}

// Update our game state
$_SESSION['gGameState'] = $gGameState;
}
```

```
if($_POST['d1Difficulty'] != "")  
{  
    $gDifficulty = $_POST['d1Difficulty'];  
  
    EndGame();  
  
    $gGameState = GAME_START;  
    StartGame();  
}  
  
if($gGameState == GAME_START)  
{  
    StartGame();  
}  
  
// Check to see if the user is starting a new game  
if($_POST['btnNewGame'] != "")  
{  
    EndGame();  
  
    $gGameState = GAME_START;  
    StartGame();  
}
```

As you can see, a switch statement is used to determine the state of the game. You also may have noticed that there is not a state for the game starting; this is because all of the game start events take place outside of the render function. The game checks to see if the New Game button has been clicked or if the difficulty level has been changed every time the page is loaded. That is what the three if statements that live outside of the render function are used for.

Here is a complete list of the functions that still need to be created.

- StartGame()
- EndGame()
- DrawBoard()
- CheckWin()
- CheckFull()
- ComputerRandomMove()
- ComputerMove()

The StartGame() and EndGame() functions control the session variables. The StartGame() function creates all the variables that will be stored in the session and changes the game state to play. The EndGame() function simply unsets all the variables and destroys the active session.

```
function StartGame()
{
    global $gGameState;
    global $gBoard;

    if($gGameState == GAME_START)
    {
        $gGameState = GAME_PLAY;
    }

    // use $_SESSION instead of session_register due to security issues
    session_start();
    $turn = $_SESSION['turn'];
    if(!isset($turn))
    {
        $turn = 1;
        $gBoard = array("", "", "", "", "", "", "", "");
        $_SESSION['gGameState'] = $gGameState;
        $_SESSION['gBoard'] = $gBoard;
        $_SESSION['gDifficulty'] = $gDifficulty;
        $_SESSION['turn'] = $turn;
    }

    // Retrieve the board state
    $gBoard = $_SESSION['gBoard'];

    // Get the difficulty level
    $gDifficulty = $_SESSION['gDifficulty'];
}

function EndGame()
{
    global $gGameState;
    global $gBoard;

    $gGameState = GAME_OVER;

    unset($gBoard);
```

```
unset($gGameState);
unset($turn);
session_destroy();
}
```

The first lines of the `StartGame()` function tell it that the function will use global variables. Then the function changes the game state and starts the session. Starting the session creates a new file in the directory where the sessions are stored, which is specified in the `php.ini` file. After the session is started it checks to see if the `$turn` variable has been set. If it has, then it doesn't bother to create new session variables, but if it hasn't been set then the function creates the array for the board, a session variable to store the game state, a session variable to store the state of the board, and a session variable to store the difficulty level; finally, it sets the variable `$turn`.

The first lines of the `EndGame()` function also tell it what global variables it should use. Then the function changes the game state to game over. After the game state has been changed, it unsets all of the variables and destroys the session.

`DrawBoard()` simply draws the board. Fantastic, isn't it? It uses HTML and PHP together to write out the appropriate board to the browser.

```
function DrawBoard()
{
    global $gBoard;

    // Start the table
    printf("<table border=0 cellpadding=0 cellspacing=0>");

    $iLoop = 0;
    for($iRow = 0; $iRow < 5; $iRow++)
    {
        printf("<tr>\n");
        for($iCol = 0; $iCol < 5; $iCol++)
        {
            if($iRow == 1 || $iRow == 3)
            {
                printf("<td width=\"12\" height=\"5\""
                    " align=\"center\" valign=\"middle\""
                    " bgcolor=\"#000000\">&nbsp;</td>\n");
            }
            else
            {
                if($iCol == 1 || $iCol == 3)
                {
```

```

printf("<td width=\"12\" height=\"115\" align=\"center\""
      valign=\"middle\" bgcolor=\"#000000\">&nbsp;</td>\n");
}
else
{
    printf("<td width=\"115\" height=\"115\""
          align=\"center\" valign=\"middle\">");

    if($gBoard[$iLoop] == "x")
    {
        printf("<img src=\" . X_IMAGE . \">");
    }
    elseif($gBoard[$iLoop] == "o")
    {
        printf("<img src=\" . O_IMAGE . \">");
    }
    else
    {
        printf("<input type=\"submit\" name=\"btnMove\" value=\""
              $iLoop . "\">");
    }
    printf("</td>\n");
    $iLoop++;
}
}

printf("</tr>\n");
}

// End the table
printf("</table>");
}

```

The DrawBoard() function uses only one global `$theBoard` itself. It has two nested loops; the first loop is the number of rows in the table that should be rendered, and the second loop is the number of columns in each row that should be rendered. If the count of `$iCol` is 1 or 3, then it draws the divider between the squares; otherwise it checks to see if there is a piece on the board in that particular row and that particular column. If there is a piece it renders it to the browser; otherwise it puts in a button.

The next two functions, `CheckFull()` and `CheckWin()`, check to see if the board is full or if someone has won the game.

```
function CheckFull()
{
    global $gGameState;
    global $gBoard;

    $gGameState = GAME_OVER;
    for($iLoop = 0; $iLoop < count($gBoard); $iLoop++)
    {
        if($gBoard[$iLoop] == "")
        {
            $gGameState = GAME_PLAY;
            return 0;
        }
    }

    return 1;
}

function CheckWin()
{
    global $gGameState;
    global $gBoard;

    $player = 1;
    while($player <= 2)
    {
        if ($player == 1)
            $tile = "o";
        else
            $tile = "x";
        if (
            # horizontal
            ($gBoard[0] == $tile && $gBoard[1] == $tile &&
             $gBoard[2] == $tile) || ($gBoard[3] == $tile && $gBoard[4] == $tile &&
             $gBoard[5] == $tile) ||
            ($gBoard[6] == $tile && $gBoard[7] == $tile &&
             $gBoard[8] == $tile) ||
            # vertical
            ($gBoard[0] == $tile && $gBoard[3] == $tile &&
             $gBoard[6] == $tile) ||
            ($gBoard[1] == $tile && $gBoard[4] == $tile &&
             $gBoard[7] == $tile) ||
            ($gBoard[2] == $tile && $gBoard[5] == $tile &&
             $gBoard[8] == $tile) ||
            ) )
            $gGameState = GAME_WIN;
    }
}
```

```

# diagonal
($gBoard[0] == $tile && $gBoard[4] == $tile &&
 $gBoard[8] == $tile) ||
($gBoard[2] == $tile && $gBoard[4] == $tile &&
 $gBoard[6] == $tile))
{
    return strtoupper($tile);
}
$player++;
}
}

```

The `CheckFull()` function starts off assuming that the game is finished. It then loops through each square on the board checking to see if there is a blank square. If the `CheckFull()` function finds a blank square it sets the game state back to play.

The `CheckWin()` function checks three tiles in a row for each player. It first checks to see if there are three horizontal tiles in a row with the same marker in them. It does the same exact procedures for three vertical tiles in a row and for three diagonal tiles in a row. If any of these tests evaluate to true, then the winning tile is returned to the render function.

The final two functions deal with the computer A.I. `ComputerRandomMove()` and `ComputerMove()` both calculate where the computer should place its piece. `ComputerRandomMove()` is used for the easiest difficulty level and `ComputerMove()` is used for the normal and impossible difficulty levels.

```

function ComputerRandomMove()
{
    global $gBoard;
    $computerMove = "";

    srand((double) microtime() * 1000000);
    while($computerMove == "")
    {
        $test = rand(0, 8);
        if($gBoard[$test] == "")
        {
            $computerMove = $test;
            $gBoard[$computerMove] = "o";
            $_SESSION['gBoard'] = $gBoard;
        }
    }
}

```

```
function ComputerMove()
{
    global $gBoard;
    $computerMove = "";

    for($player = 0; $player <= 1; $player++)
    {
        if($player == 0)
        {
            $tile = "o";
        }
        else
        {
            $tile = "x";
        }

        if ($gBoard[0] == $tile && $gBoard[1] == $tile &&
            $gBoard[2] == '')
            $computerMove = 2;
        if ($gBoard[0] == $tile && $gBoard[1] == '' && $gBoard[2] == $tile)
            $computerMove = 1;
        if($gBoard[0] == '' && $gBoard[1] == $tile &&
            $gBoard[2] == $tile)
            $computerMove = 0;
        if($gBoard[3] == $tile && $gBoard[4] == $tile &&
            $gBoard[5] == '')
            $computerMove = 5;
        if($gBoard[3] == $tile && $gBoard[4] == '' &&
            $gBoard[5] == $tile)
            $computerMove = 4;
        if($gBoard[3] == '' && $gBoard[4] == $tile &&
            $gBoard[5] == $tile)
            $computerMove = 3;

        if($gBoard[6] == $tile && $gBoard[7] == $tile &&
            $gBoard[8] == '')
            $computerMove = 8;
        if($gBoard[6] == $tile && $gBoard[7] == '' &&
            $gBoard[8] == $tile)
            $computerMove = 7;
        if($gBoard[6] == '' && $gBoard[7] == $tile &&
            $gBoard[8] == $tile)
            $computerMove = 6;
```

```
if($gBoard[0] == $tile && $gBoard[3] == $tile &&
   $gBoard[6] == '')
  $computerMove = 6;
if($gBoard[0] == $tile && $gBoard[3] == '' &&
   $gBoard[6] == $tile)
  $computerMove = 3;
if($gBoard[0] == '' && $gBoard[3] == $tile &&
   $gBoard[6] == $tile)
  $computerMove = 0;

if($gBoard[1] == $tile && $gBoard[4] == $tile &&
   $gBoard[7] == '')
  $computerMove = 7;
if($gBoard[1] == $tile && $gBoard[4] == '' &&
   $gBoard[7] == $tile)
  $computerMove = 4;
if($gBoard[1] == '' && $gBoard[4] == $tile &&
   $gBoard[7] == $tile)
  $computerMove = 1;
if($gBoard[2] == $tile && $gBoard[5] == $tile &&
   $gBoard[8] == '')
  $computerMove = 8;
if($gBoard[2] == $tile && $gBoard[5] == '' &&
   $gBoard[8] == $tile)
  $computerMove = 5;
if($gBoard[2] == '' && $gBoard[5] == $tile &&
   $gBoard[8] == $tile)
  $computerMove = 2;

if($gBoard[0] == $tile && $gBoard[4] == $tile &&
   $gBoard[8] == '')
  $computerMove = 8;
if($gBoard[0] == $tile && $gBoard[4] == '' &&
   $gBoard[8] == $tile)
  $computerMove = 4;
if($gBoard[0] == '' && $gBoard[4] == $tile &&
   $gBoard[8] == $tile)
  $computerMove = 0;

if($gBoard[2] == $tile && $gBoard[4] == $tile &&
   $gBoard[6] == '')
  $computerMove = 6;
```

```

        if($gBoard[2] == $tile && $gBoard[4] == '' &&
           $gBoard[6] == $tile)
            $computerMove = 4;
        if($gBoard[2] == '' && $gBoard[4] == $tile &&
           $gBoard[6] == $tile)
            $computerMove = 2;

        if($computerMove <> '')
            break;
    }

    return $computerMove;
}

```

`ComputerRandomMove()` simply calculates a random number from the server's timer. Once it gets this random number it tests to see if that square is blank. If the tested square is blank, then the computer puts its piece in that square. If the square tested is not blank, `ComputerRandomMove()` is run again until an empty square is found.

`ComputerMove()` looks at every single square on the board. If there are two tiles in a row with the opposing tile in them, the computer will place its tile on the third open square, thus blocking the opponent. This level of difficulty is increased in the render function with the following lines:

```

$computerMove = ComputerMove();
if ($computerMove == '')
{
    if($gBoard[4] == '')
        $computerMove = 4;
    elseif($gBoard[0] == '')
        $computerMove = 0;
    elseif($gBoard[2] == '')
        $computerMove = 2;
    elseif($gBoard[6] == '')
        $computerMove = 6;
    elseif($gBoard[8] == '')
        $computerMove = 8;
    if($computerMove == '')
        ComputerRandomMove();
}

```

First it calculates the computer's move. If the computer's move is blank it tries to pick the first available square. If it doesn't find an available square it randomly calculates the move. With these three methods of calculating moves, it is nearly impossible to win. It makes for a great challenge.

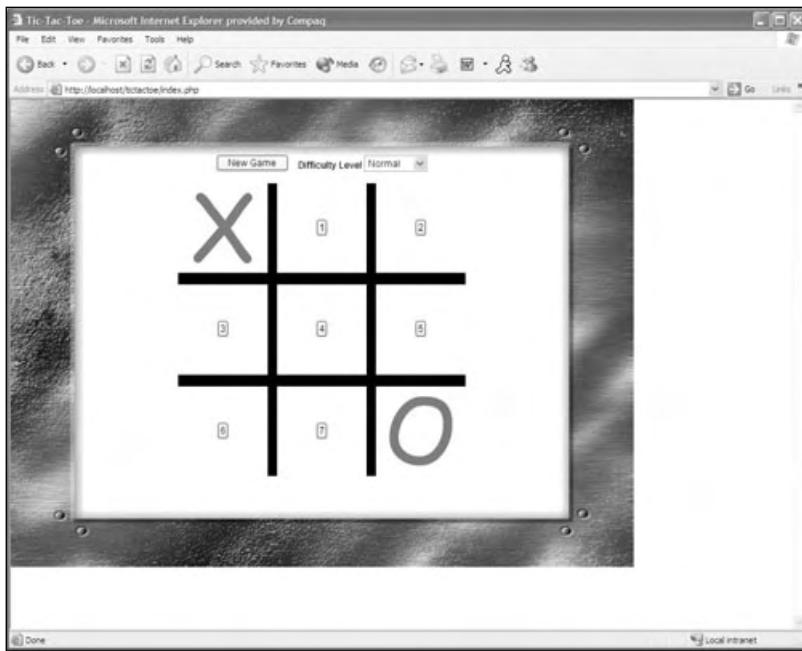


Figure 6.10 Your first PHP game!

Congratulations! You have successfully completed your very first PHP game. The results of your efforts should look like Figure 6.10.

Note

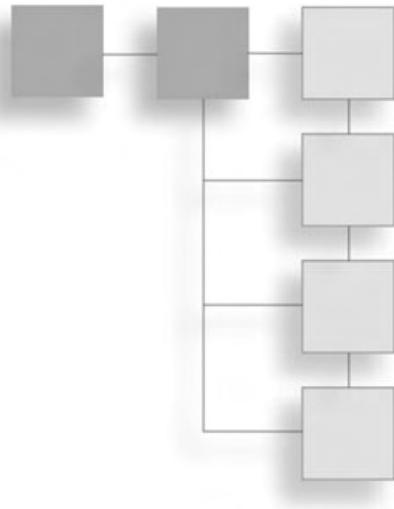
At this point you will want to go into your php.ini file and turn display_errors to off. The reason for this is because you only want to display errors when debugging; otherwise you will have warning notices all over the place that really don't affect the gameplay.

Conclusion

You have covered your last basic concept in PHP arrays. You learned how to initialize arrays, use strings for indexes, loop through arrays, create multi-dimensional arrays, and sort your arrays. You have also created your very first PHP game. It can only get more exciting from here. Next you will take a tour through databases and create your very own chess game. I know that tic-tac-toe was a lot of fun but not very difficult. It is time to really test your skills.

CHAPTER 7

PLAYING WITH CHESS AND DATABASES



- Non-Relational Databases
- Creating and Opening a Database
- Looping through Databases
- Modifying Your Database
- Chess Programming Basics
- Starting Your Chess Game
- Working with the Pieces
- Adding the Database

What is a database? A *database* provides persistent storage for information. There are three types of databases: relational databases, non-relational databases, and object oriented databases. In a relational database the data is stored in a set of tables and the information is related to each other by some unique identifier. Some examples of a relational database system are mySQL, SQL Server, and Oracle. Relational databases typically use a language called T-SQL to access the information inside a table.

Object oriented databases use objects to represent data much like you would if you were creating classes and objects to represent your program's internal data. Object oriented databases are fairly new, so the algorithms for searching aren't as well defined as a relational database, but this type of database gives great promise to the object oriented development community. Some of the most popular object oriented databases are Versant (www.versant.com), GemStone (www.gemstone.com), and ObjectStore (www.odi.com/odilive/).

For your purposes with this book you will use a non-relational database. Non-relational databases use files to store their information. They are called *non-relational* databases because the stored data is not related to any other data. Pretty logical name, if you ask me.

Non-Relational Databases

As mentioned, a non-relational database stores its information in files. So how does it do this? PHP supports something called DBA. DBA stands for *database abstraction*, and this can handle many different database formats. When a non-relational database stores its data, it stores it in a key/value pair. In other words, each stored entry consists of a value with an associated key. These key/value pairs are simple strings, not char arrays, so you can also store binary information in these databases. You are not limited to the keys that you can choose because the keys are not stipulated by the database.

DBA allows you to create databases; update existing databases; insert, update, and delete new entries; and traverse the entire database. The biggest bonus for you is that non-relational databases don't require any additional software. All you have to do is enable DBA support in the PHP interpreter by editing the php.ini file. This should already be completed; you went through this step when you installed the PHP interpreter.

Creating and Opening a Database

PHP provides several functions for DBA. The first function you must know is the `dba_open()` function. This will allow you to create a database, write to a database, read a database, or truncate a database.

```
int dba_open(string sPath, string sMode, string sHandler, [int nMod]);
```

The first parameter of the `dba_open()` function is the path to the database; the second parameter is the mode in which you wish to open the database. Possible values for this can be any of the following:

- `r` -Open the database for reading.
- `w` -Open the database for writing.
- `c` -Create a new database.
- `n` -Truncate a database.

The third parameter is the name of the handler for the type of database you would like to use. The third parameter can be any of the following values:

- `dbm` -Berkley DB-style database (deprecated).
- `ndbm` -A newer Berkley DB-style database. This is very limited and is also deprecated.

- **gdbm** –The GNU database.
- **db3** –The DB3 database toolkit from Sleepycat Software. This will be the database type that you will use in your projects.
- **cdb** –Qmail database. This format only supports reading operations.

The optional fourth parameter specifies the mode in which the database should open. This is not a commonly used parameter and you won't be using it in your games.

The `dba_open()` function will return a handle for the database if it was successful; otherwise it returns 0 or false when it fails. This will allow you to verify that the database is open. If the open fails you can trap for it and either print a warning message or exit the application altogether. Take a look at the following code example to see how to use the `dba_open()` function.

```
<?php  
// Set the db parameters  
$dbPath = "myDatabase.db";  
$dbType = "db3";  
  
function CreateDatabase($thePath, $theType)  
{  
    $db = dba_open($thePath, "c", $theType);  
    if(!$db)  
    {  
        printf("Could not create the database");  
        return 0;  
    }  
  
    return $db;  
}  
  
function OpenDatabase($thePath, $theType)  
{  
    $db = dba_open($thePath, "r", $theType);  
    if(!$db)  
    {  
        printf("Could not open the database");  
        return 0;  
    }  
  
    return $db;  
}
```

```
// Open the database, if it isn't there, create it
$db = OpenDatabase($dbPath, $dbType);
if(!$db)
{
    $db = CreateDatabase($dbPath, $dbType);
    if(!$db)
    {
        exit;
    }
}

// If you get here the database has opened successfully and you can do what you want
with it.
?>
```

Looping through the Database

Now that you know how to create and open a non-relational database it would be handy to know how to loop through the data in the database to display it. To loop through a database record set you need to start at the very first key. PHP provides you with `dba_firstkey()` to get the first key of a specified database.

```
string dba_firstkey(int databaseHandle);
```

Once you retrieve the first key of the database you need to loop through each record until there are no more records. For this you will use a while loop because you won't know the count of elements in the record set. To get the value, you use the `dba_fetch()` function.

```
string dba_fetch(string key, int databaseHandle);
```

The `dba_fetch()` function will return a string of false if it was unable to get the data. Once you have retrieved the data you will need to unserialize the data, just like you will have to serialize the data to put it into the database.

```
string unserialize(string someString);
```

After you have done this you can do what you want with the unserialized string. Then you need to retrieve the next key in order to move to the next record. Take a look at the following example to see how it all works together:

```
<?php
$dbPath = "myDatabase.db";
$dbType = "db3";

$db = OpenDatabase($dbPath, $dbType);
```

```
if(!$db)
{
    $db = CreateDatabase($dbPath, $dbType);
    if(!$db)
    {
        exit;
    }
}

// Get the first record
$key = dba_firstkey($db);

// Loop through the whole database
while($key != false)
{
    $value = dba_fetch($key, $db);
    $entry = unserialize($value);
    // Do something with $entry
    $key = dba_nextkey($db);
}
dba_close($db);
?>
```

Note

Remember to always close the database you're working with by using the `dba_close()` function.

Inserting an Entry into Your Database

To insert an entry, PHP provides you with the `dba_insert()` function. This function takes three arguments. The first is the key that you want to give the record, the second argument is the value for the key, and the third argument is the handle to the database.

```
bool dba_insert(string key, string value, int handle);
```

If the insert to the database is successful `dba_insert()` will return true. If the insert into the database fails `dba_insert()` will return false. When inserting a value into the database make sure you serialize the data first.

```
$results = dba_insert($myKey, serialize($myData), $db);
```

All the `serialize()` function does is create a serialized string from a mixed data type that you pass in to it, so you can pass in an array and the results will be a serialized string. Then when you retrieve the data from the database, you unserialize the string by using the

unserialize() function. Take a look at the following code example to see how you put it all together:

```
<?php  
$dbPath = "myDatabase.db";  
$dbType = "db3";  
  
$data = "My Data";  
  
$db = OpenDatabase($dbPath, $dbType);  
if (!$db)  
{  
    $db = CreateDatabase($dbPath, $dbType);  
    if (!$db)  
    {  
        exit;  
    }  
}  
// Now that the database is open you need to find the next available key  
$nextID = 0;  
$key = dba_firstkey($db);  
while ($key != false)  
{  
    if ($key > $nextID)  
    {  
        $nextID = $key;  
    }  
    $key = dba_nextkey($db);  
}  
$nextID++; // This is the next largest available key  
$result = dba_insert($key, serialize($data), $db);  
dba_close($db)  
  
if (!$result)  
{  
    printf("Insert into database failed");  
}  
else  
{  
    printf("Added successfully");  
}  
?>
```

So what is this example doing, exactly? First it opens the database using the functions that you created earlier in this chapter. Once the database is opened, you need to find the next available ID. This ensures that you are inserting your data at the end of the file. To do this you need to get the first key in the database, then loop through the entire database until you reach the end. If a key is found that is greater than the current \$nextID then the \$nextID is set to the largest key. Once this loop is finished you have the largest key in the database, so the next available key is obviously \$nextID + 1. Now you can freely insert your record into the database. Once you have called the dba_insert() function you need to check to see if it is successful. That is all there is to it.

Updating an Entry in Your Database

Updating records in your database is very similar to inserting records, but instead of calling dba_insert() you call dba_replace(). The dba_replace() function also takes three arguments. Can you guess what they are? That's right, the key you want to update, the data you are going to update the record with, and the database handle.

```
bool dba_replace(string key, string data, int database);
```

Now, once you call this function you must call the dba_sync() function or else your data will not be saved to the database. This can be a pain in the butt to debug if you miss putting in this little function.

```
bool dba_sync(int database);
```

So, to update a record you start off exactly like you were inserting a record—you need to open the database. But this time you do not need to find the next available record because you already know what record you want to update. If you don't know what record you want to update then you probably shouldn't be updating the database.

```
<?php
function UpdateRecord($id, $value)
{
    global $dbPath, $dbType;
    $db = OpenDatabase($dbPath, $dbType);
    dba_replace($id, serialize($value), $db);
    dba_sync($db);
    dba_close($db)
}
?>
```

Deleting an Entry from Your Database

With the power to create, you also need the power to destroy. To delete an entry from your database you use the `dba_delete()` function. Amazing name, isn't it? The `dba_delete()` function takes two parameters. The first is the id of the record you wish to delete, and the second is the handle to the database.

```
bool dba_delete(string key, int database);
```

Just like when you update the database, you must call the `dba_sync()` function or else your database will not be updated. Take a look at this handy-dandy function.

```
<?php
function DeleteRecord($id)
{
    global $dbPath, $dbType;
$db = OpenDatabase($dbPath, $dbType);
    dba_delete($id, $db);
    dba_sync($db);
    dba_close($db)
}
?>
```

Caution

Remember to always call `dba_close()` after you are done operating on the database. And always remember to call `dba_sync()` to update your database.

Chess Programming: A Quick Overview

Before you begin programming your chess game you need to understand how you would go about it. Take a look at the minimum software requirements of a chess game.

- A way to represent a chess board in memory. This is where you will store the whole state of the game.
- Some sort of system to determine if an illegal move was made.
- Some sort of user interface so you can make your moves.

Remember these are the bare minimum software components you would need to create a chess game. This doesn't include a move evaluation system so the computer can make moves. Now that you have a good direction to go in to start programming a chess game, take a look at how you would represent a board.

There are several ways of representing a chess board, but the most obvious way is to use some sort of array to represent the board. Why an array? Because you want to have a vari-

able that you can loop through to render the board, and an array is the most logical data structure. To make a move in chess you give the board a from square (e.g., a1) and a to square (e.g., a3). If you are using an array, then you can loop through these columns and rows and draw the board on the screen quite easily. Ideally you would use bit boards to represent positions of pieces on the board itself.

A bit board is a 64-byte array that holds a bit. A 1 would be in a square if the square is taken and a 0 would be in a square if the square is free. So you can represent an entire chess game by using 12 bit boards. One for the position of all the white pawns, white rooks, white bishops, white knights, white queen, white king, black pawns, and so on. With these bit boards your validation of moves and calculations of moves will be a whole lot quicker than looping through one 64-byte array for the whole board. However, PHP does not support 64-bit integers so you cannot use bit boards unless you have a 64-bit system.

This is a very quick overview of chess programming. If you would like more in-depth information about chess programming, check out other books on the subject. Let's start programming our chess game.

Starting the Chess Game

The first step you should take when starting any game is to create your game states and general globals. For this chess game you will have only three game states. One tells you when the game is starting, the second tells you the game is running, and the third tells you when the game is over.

```
// Game States
define("GAME_STARTING", 1);
define("GAME_RUNNING", 2);
define("GAME_OVER", 3);
// Globals
global $gGameState;
global $gBoard;
global $gCurrentPlayer;
```

Now let's create the HTML framework for the game. It should not be anything too fancy. (As a matter of fact, it will be pretty much like the tic-tac-toe game you created in the last chapter.)

```
<!doctype html public "-//W3C//DTD HTML 4.0 //EN">
<html>
<head>
    <title>Chess</title>
    <link rel="stylesheet" href="style.css" type="text/css">
</head>
<body>
```

```
<form action="chess.php" method="post">
<input type="hidden" name="player" value="<? printf($gCurrentPlayer) ?>">
<input type="hidden" name="turn" value="<? printf($turn) ?>">
    <?php WriteTableHeader(); ?>
    <div align="center">
        <input type="submit" name="btnNewGame" value="New Game">&nbsp;&nbsp;&nbsp;
        <b>Move From:</b><input type="text" name="fromSquare">&nbsp;&nbsp;
        <b>Move To:</b><input type="text" name="toSquare"><br><br>
    <?php
        // Render the game
        Render();
    ?>
    </div>
    <?php WriteTableFooter(); ?>
</form>

</body>
</html>
```

This little chunk of HTML renders our cool framework—a button to start a new game, a place to enter in your moves, and the rest of the game. Take a look at Figure 7.1 to see a general layout of what the game will look like.

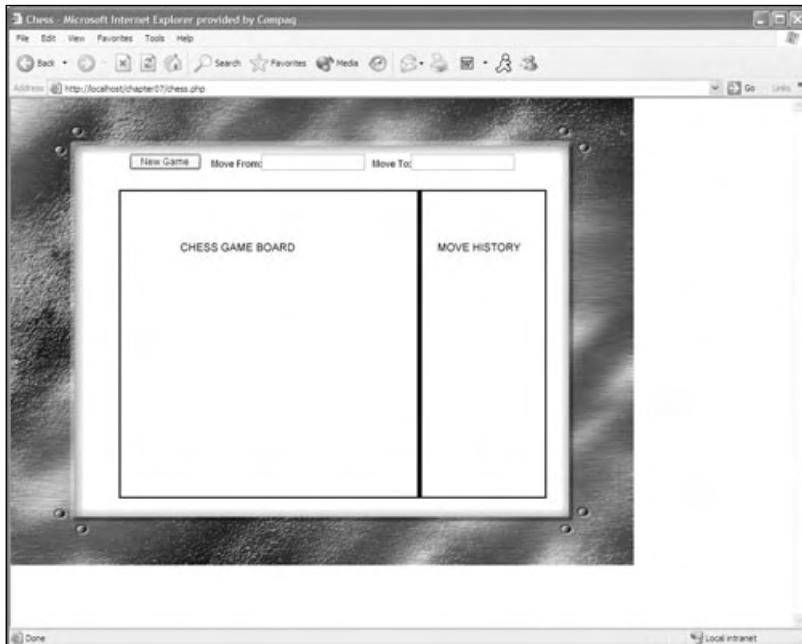


Figure 7.1 General layout of the chess game.

Working with the Pieces

Now that you know how you generally want to lay out the chess game, you can create the constants for the pieces and a few functions to move the pieces. Also you will want to create the general render function to start out your game.

```
<?php
function Render()
{
    global $gGameState;
    global $gBoard;

    switch($gGameState)
    {
        case GAME_RUNNING:
        {
            DrawBoard();
        }
        case GAME_OVER:
        {
            printf("Game is Over");
        }
    }
    // Update our game state
    $_SESSION['gGameState'] = $gGameState;
}
?>
<!--pieces.php -->
<?php
// Constants for piece definitions
define("PAWN", 0);
define("KNIGHT", 2);
define("BISHOP", 4);
define("ROOK", 6);
define("QUEEN", 8);
define("KING", 10);
define("EMPTY_SQUARE", 12);

// Stuff for the bitboards
define("ALL_PIECES", 12);define("ALL_SQUARES", 64);
define("ALL_BITBOARDS", 14);

// White pieces
```

```
define("ALL_WHITE_PIECES", ALL_PIECES);
define("WHITE_PAWN", PAWN);
define("WHITE_KNIGHT", KNIGHT);
define("WHITE_BISHOP", BISHOP);
define("WHITE_ROOK", ROOK);
define("WHITE_QUEEN", QUEEN);
define("WHITE KING", KING);

// Black pieces
define("ALL_BLACK_PIECES", ALL_PIECES + 1);
define("BLACK_PAWN", PAWN + 1);
define("BLACK_KNIGHT", KNIGHT + 1);
define("BLACK_BISHOP", BISHOP + 1);
define("BLACK_ROOK", ROOK + 1);
define("BLACK_QUEEN", QUEEN + 1);
define("BLACK KING", KING + 1);

// Piece Values
$pieceValues[WHITE_PAWN] = 100;
$pieceValues[WHITE_KNIGHT]      = 300;
$pieceValues[WHITE_BISHOP]      = 350;
$pieceValues[WHITE_ROOK]        = 500;
$pieceValues[WHITE_QUEEN]       = 900;
$pieceValues[WHITE KING]        = 2000;
$pieceValues[BLACK_PAWN] = 100;
$pieceValues[BLACK_KNIGHT]      = 300;
$pieceValues[BLACK_BISHOP]      = 350;
$pieceValues[BLACK_ROOK]        = 500;
$pieceValues[BLACK_QUEEN]       = 900;
$pieceValues[BLACK KING]        = 2000;
function StartBoard()
{
    global $gBoard;

    $gBoard = array(
        BLACK_ROOK,    BLACK_KNIGHT, BLACK_BISHOP, BLACK_QUEEN,
        BLACK KING,    BLACK_BISHOP, BLACK_KNIGHT, BLACK_ROOK,
        BLACK_PAWN,    BLACK_PAWN,    BLACK_PAWN,    BLACK_PAWN,
        BLACK_PAWN,    BLACK_PAWN,    BLACK_PAWN,    BLACK_PAWN,
        EMPTY_SQUARE,  EMPTY_SQUARE, EMPTY_SQUARE, EMPTY_SQUARE,
        EMPTY_SQUARE,  EMPTY_SQUARE, EMPTY_SQUARE, EMPTY_SQUARE,
```

```
    EMPTY_SQUARE, EMPTY_SQUARE, EMPTY_SQUARE, EMPTY_SQUARE,
    WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,
    WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,   WHITE_PAWN,
    WHITE_ROOK,   WHITE_KNIGHT, WHITE_BISHOP, WHITE_QUEEN,
    WHITE_KING,   WHITE_BISHOP, WHITE_KNIGHT, WHITE_ROOK);
}

// Puts a piece on the board
// $square is the square of the piece 0-63
// $piece is the piece that is moving
function PutPiece($square, $piece)
{
    global $gBoard;

    // Put the piece on the board
    $gBoard[$square] = $piece;
}

// Takes a piece off the board
// $square is the square of the piece 0-63 you are removing
function TakePiece($square)
{
    global $gBoard;

    // Take the piece off the bit board
    $gBoard[$square] = EMPTY_SQUARE;
}

// This moves a piece
function MovePiece($fromSquare, $toSquare, $piece)
{
    PutPiece($toSquare, $piece);
    TakePiece($fromSquare);
}

function UpdateMoveList($fromSquare, $toSquare)
{
    $data = $fromSquare + " - " + $toSquare;
```

```

// Open the database
$db = dba_open("chess.db", "c", "db3");
if(!$db)
{
    dba_close();
    $db = dba_open("chess.db", "w", "db3");
    if(!$db)
    {
        printf("Unable to open the database.");
        WriteTableFooter();
    }
}

// Now that the database is open you need to find the next available key
$nextID = 0;
$key = dba_firstkey($db);
while($key != false)
{
    if($key > $nextID)
    {
        $nextID = $key;
    }
    $key = dba_nextkey($db);
}
$nextID++; // This is the next largest available key

dba_insert($nextID, serialize($data), $db);

// Close the database
dba_close($db);
}
?>

```

First you create a shell for the `render()` function. This is fairly straightforward and nothing new to you. Next you create a new file called `pieces.php`. This is what you will include along with the `common.php` file. In `pieces.php` will be all of the logic for starting a board, moving a piece, and taking a piece.

Notice that in `pieces.php` you have several constants. This makes it easy to create the necessary pieces for both sides. Next you create each piece type that is on the board using the constants that you created. If you haven't guessed yet, the white player is the constant 0 and the black player is the constant 1. You can create defines for these if you like, and you probably should.

The StartBoard() function is fairly straightforward. It initializes the entire board and places the pieces in the proper places. Since you will be the white player, you are technically at the bottom of the board and black is at the top of the board. Since black is at the top of the board it takes up the first locations of the array and white takes up the last locations of the array.

The PutPiece() function takes a square and a piece as its parameters. The square is the square you would like to put the piece on, and the piece is the piece you are placing on that square. The TakePiece() function simply takes a square that you are taking a piece off of. The MovePiece() function takes a square and a piece then calls the PutPiece() function and the TakePiece() function.

Now you have the basic logic for putting a piece on the board, taking a piece off the board, and starting a new board. Plus you have all the variables you need for the bit boards and all the variables you need to represent the pieces. Next you need a way to render the board to the browser so you can see it.

Think about how a chess board is laid out: it is an 8×8 board with alternating colors for squares. The square in the lower right of the board and the square in the upper left of the board must be the lighter-colored square. Your brain should now be saying things like, “A loop would be good. But wait, an 8×8 board two loops would be better.” That’s right, you need two for loops, one for the eight rows, and one for the eight columns in the board. Let’s use two shades of brown for the board. For the lighter color use #E4B578 and for the darker-colored squares let’s use #B88645.

```
function DrawBoard()
{
    global $gBoard;

    printf("<table border=\"0\" cellpadding=\"0\" cellspacing=\"1\">
        <tr><td align=\"center\" valign=\"middle\" width=\"300\">");
    // Start our table to contain the board
    printf("<table border=\"0\" cellpadding=\"0\""
        cellspacing=\"1\" width=\"240\">");

    $currColor = "#E4B578";

    for($row = 0; $row < 8; $row++)
    {
        // Print a new table row
        printf("<tr>");

        for($col = 0; $col < 8; $col++)
            if($currColor == "#E4B578")
                printf("<td style=\"background-color: #B88645; width: 30px; height: 30px;\">");
```

```
{  
    // Get the piece to render  
    $piece = $gBoard[$row * 8 + $col];  
  
    // Start a new table cell for this piece  
    printf("<td bgcolor=\"$currColor>");  
  
    // Place the piece on the board  
    switch($piece)  
    {  
        case WHITE_PAWN: {  
            printf("<img src=\"images/wp.gif\""  
                  " border=\"0\""); break; }  
        case WHITE_KNIGHT: {  
            printf("<img src=\"images/wn.gif\""  
                  " border=\"0\""); break; }  
        case WHITE_BISHOP: {  
            printf("<img src=\"images/wb.gif\""  
                  " border=\"0\""); break; }  
        case WHITE_ROOK: {  
            printf("<img src=\"images/wr.gif\""  
                  " border=\"0\""); break; }  
        case WHITE_QUEEN: {  
            printf("<img src=\"images/wq.gif\""  
                  " border=\"0\""); break; }  
        case WHITE KING: {  
            printf("<img src=\"images/wk.gif\""  
                  " border=\"0\""); break; }  
        case BLACK_PAWN: {  
            printf("<img src=\"images/bp.gif\""  
                  " border=\"0\""); break; }  
        case BLACK_KNIGHT: {  
            printf("<img src=\"images/bn.gif\""  
                  " border=\"0\""); break; }  
        case BLACK_BISHOP: {  
            printf("<img src=\"images/bb.gif\""  
                  " border=\"0\""); break; }  
        case BLACK_ROOK: {  
            printf("<img src=\"images/br.gif\""  
                  " border=\"0\""); break; }  
    }  
}
```

```
    case BLACK_QUEEN:  {
        printf("<img src=\"images/bq.gif\""
              "border=\"0\""); break; }
    case BLACK_KING:   {
        printf("<img src=\"images/bk.gif\""
              "border=\"0\""); break; }
    case EMPTY_SQUARE: {
        printf("<img src=\"images/blank.gif\" border=\"0\"");
              break; }
    default:           {
        printf("<img src=\"images/blank.gif\" border=\"0\"");
              break; }
    }

    // End this table cell
    printf("</td>\n");

    // Switch the color
    if($currColor == "#E4B578")
    {
        $currColor = "#B88645";
    }
    else
    {
        $currColor = "#E4B578";
    }

}

// End this table row
printf("</tr>\n");

//Switch the color
if($currColor == "#E4B578")
{
    $currColor = "#B88645";
}
else
{
    $currColor = "#E4B578";
}
}
```

```
// End the table
printf("</table>");

printf("</td><td align=\"center\" valign=\"middle\""
      "width=\"100\">"DATABASE STUFF HERE</td></tr></table>");
WriteTableFooter();
}
```

The DrawBoard() function uses one global: the board itself. It contains two for loops; one is to loop through the rows of the board and the other is to loop through the columns of the board. Since you are using a single-dimension array to store the state of the board, you need to calculate the row and column. To do this you multiply the row you are currently on by the number of columns you have, which in this case is 8. Then you add the column that you are currently on, and this will give you your proper index into your array. Once you have this index you can retrieve the piece that is in that position and draw it on the board. After one column is done, it changes the colors of the squares and repeats for all rows and columns. The results of this should look like Figure 7.2.

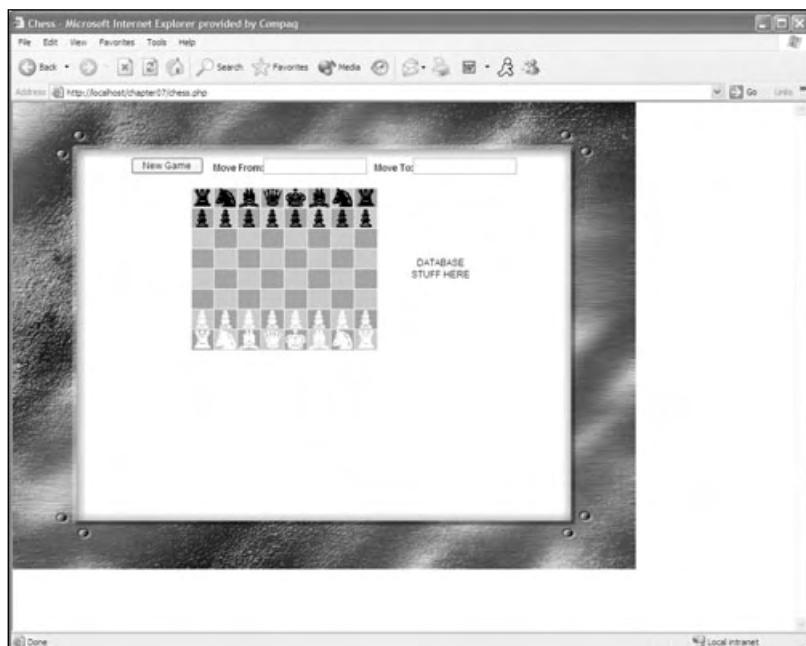


Figure 7.2 Results of your render function.

Now that you have the basic shell of the game, let's take the user's input, store it in a database, update the board and the "DATABASE STUFF HERE" section accordingly.

Getting the User Input and Modifying the Database

To retrieve the user's desired moves you need to keep track of a few things. One is which player's turn it is. Remember the global variable \$gCurrentPlayer that you created when you first started this game? That's what you will use to keep track of the player. You'll want to take in your user input in one central location for your programming, and then you will want to update the board so the user can visually see what is happening.

```
function ProcessInput()
{
    global $gBoard, $gCurrentPlayer;

    if($_POST['fromSquare'] == "")
        return;

    // Get the to and from square
    $fromSquare = GetSquare($_POST["fromSquare"]);
    $toSquare = GetSquare($_POST["toSquare"]);

    // Get the piece to be moved
    $piece = $gBoard[$fromSquare];
    if(($piece % 2) != 0 && $gCurrentPlayer == "white")
    {
        printf("It is not your turn");
        return;
    }
    // Move the piece
    MovePiece($fromSquare, $toSquare, $piece);

    // Update the database
    UpdateMoveList($_POST["fromSquare"], $_POST["toSquare"]);

    // Change the current player
    if($gCurrentPlayer == "white")
    {
        $gCurrentPlayer = "black";
    }
}
```

```
        }
    else
{
    $gCurrentPlayer = "white";

}

// Store the updated board in the session
$_SESSION['gBoard'] = $gBoard;
$_SESSION['gCurrentPlayer'] = $gCurrentPlayer;
}

function GetSquare($strSquare)
{
    $col = substr($strSquare, 0, 1);
    $row = substr($strSquare, 1, 1);

    switch($col)
    {
        case "a": { $col = 0; break;}
        case "b": { $col = 1; break;}
        case "c": { $col = 2; break;}
        case "d": { $col = 3; break;}
        case "e": { $col = 4; break;}
        case "f": { $col = 5; break;}
        case "g": { $col = 6; break;}
        case "h": { $col = 7; break;}
    }

    switch($row)
    {
        case "8": { $row = 0; break;}
        case "7": { $row = 1; break;}
        case "6": { $row = 2; break;}
        case "5": { $row = 3; break;}
        case "4": { $row = 4; break;}
        case "3": { $row = 5; break;}
        case "2": { $row = 6; break;}
        case "1": { $row = 7; break;}
    }

    return($row * 8 + $col);
}
```

The ProcessInput() function first retrieves the square that you are moving from and the square that you are moving to by using the GetSquare() function. The GetSquare() function gets the row and the column that the user entered by using the substr() function. The column is the first character in the string, and the row is the second character in the string. Since white is on the bottom and white always moves first, you need to flip the board to get the proper numbers. That is what the two switch statements are used for. The first switch statement relates a letter to a column number. The second switch statement reverses the order of the rows. The return statement is the exact calculation you used in the DrawBoard() function.

After the source square and the destination square are retrieved you need to check to see if the current user is even allowed to move the piece that he is trying to move. If it is not his turn you let him know. You can determine if it is white's turn or not because the piece number that you defined in the pieces.php file are all even for white and odd for black. So if you perform a modulus by two on the current piece and the result is zero, then it is white's turn; otherwise it is black's turn. The last step to making a move is to move the piece on the board, update the database, switch the current player's turn, and update the session data.

The UpdateMoveList() function takes two arguments: the from square and the to square. This is the function that inserts the information into the database. Take a look at it:

```
function UpdateMoveList($fromSquare, $toSquare)
{
    $data = $fromSquare + " - " + $toSquare;

    // Open the database
    $db = dba_open("chess.db", "c", "db3");
    if (!$db)
    {
        dba_close();
        $db = dba_open("chess.db", "w", "db3");
        if (!$db)
        {
            printf("Unable to open the database.");
            WriteTableFooter();
        }
    }

    // Now that the database is open you need to find the next available key
    $nextID = 0;
    $key = dba_firstkey($db);
    while ($key != false)
```

```

{
    if($key > $nextID)
    {
        $nextID = $key;
    }
    $key = dba_nextkey($db);
}
$nextID++; // This is the next largest available key

dba_insert($nextID, serialize($data), $db);

// Close the database
dba_close($db);
}

```

The first thing `UpdateMoveList()` does is create a string with the data that you want to insert into the file. Then it creates a new database called `chess.db`. If `chess.db` already exists, then it opens it with read/write permissions. To insert any data into the database you need to find the first available key. To do this you must loop through the database and find the last key and add one. After this is done you can insert your record and close the database.

Now that you're storing the moves in a database you obviously will want to show the user the move history. To do this you will want to use the `<IFRAME></IFRAME>` tag and point the source page to a new PHP page called `move.php`. Here is the whole `move.php` file:

```

<?php
// Display the moves in the page
$db = dba_open("chess.db", "r", "db3");

// Get the first record
$key = dba_firstkey($db);

$move = 1;
// Loop through the whole database
while($key != false)
{
    $value = dba_fetch($key, $db);
    $entry = unserialize($value);
    printf($move . ". " . $entry . "<BR>");
    $key = dba_nextkey($db);
    $move++;
}

```

```
}
```

```
dba_close($db);
```

```
?>
```

Now that you have the page to display the moves you will need to edit your DrawBoard() function to write out the <IFRAME></IFRAME> tag. All you will need to do is specify the source page for the inlaid frame, the width, and the height, like this:

```
printf("<IFRAME src=\"move.php\" width=\"200\"  
height=\"300\">Unable to display move history.</IFRAME>");
```

This line should go where you originally had the “DATABASE STUFF HERE” line. The results of all of your hard work are shown in Figure 7.3.

Note

At this point you should go into your php.ini file and turn display_errors to off. The reason for this is because you only want to display errors when debugging; otherwise you will have warning notices all over the place that really don’t affect the game play.

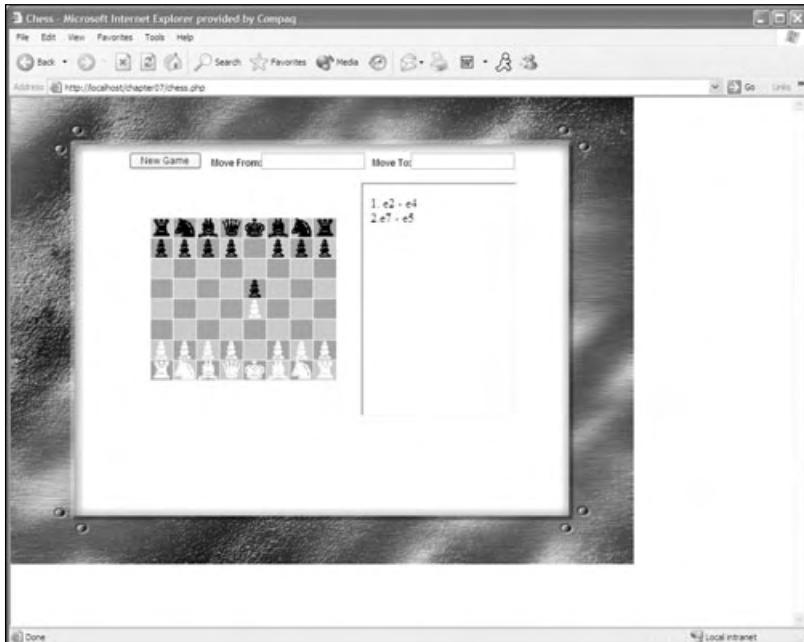


Figure 7.3 Results of using the database.

Conclusion

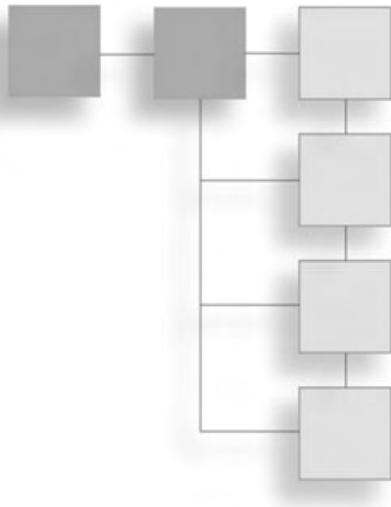
This game is far from over. There are so many more elements you could add, such as computer A.I., move validation, and network support—all of which would be great features for your game.

However, you have covered a great deal of information in this chapter. You have learned what a non-relational database is, how to create a non-relational database, how to modify a database, some chess programming basics, and, on top of all that, you have created a basic chess game with database support to show the move history. Although there are several elements you could add to improve the chess game, you have completed a fantastic job of creating a really cool basic chess game.

Next up: working with dynamic graphics.

CHAPTER 8

GD GRAPHICS OVERVIEW



- What Is GD?
- Installing GD
- Creating an Image
- Drawing Shapes on the Canvas
- Manipulating Color Information
- Adding Text to Images
- Resizing Images

Boutell's GD library is a great tool that provides you with ways to manipulate graphics on the fly. In this chapter you will be learning just that. You will start off by learning how to create images and draw simple shapes on the canvas. Then you will learn how to manipulate the color information of an image, add text to an image, and, finally, how to resize an image. This knowledge will allow you to add cool graphics on the fly to your on-line game.

What Is GD?

GD is a C graphics library that allows you to create and manipulate .jpegs, .pngs, and .wbmps. What's that you say? No support for .gifs? That is correct. There used to be a GD library out there at one time that supported the .gif file format, but since Unisys (the company that owns the patent on LZW compression that .gifs use) changed their licensing agreements, GD has stopped supporting the .gif file format. This could change in the next

year, because Boutell (the company that makes the GD library) is planning to support .gifs as soon as the patent expires. Meanwhile, have no fear because .pngs support all of the features of a .gif (as was discussed in Chapter 3).

Installing GD

Just in case you missed installing the GD library when you were setting up PHP, I'll quickly go over it again. I will also show you a great little chunk of code that you can use to load the library dynamically.

To install GD on a Linux or UNIX installation you will need to recompile PHP with the --with-gd option. You can also compile the GD library as a shared object so you can load the GD library dynamically whenever you need it. To do this you would use the --with-gd=shared option. Then, to load in the library dynamically at run time, you would use the following line of code:

```
dl('gd.so');
```

To install GD on a Windows platform you need to copy the php_gd2.dll that came with the installation package into the extensions directory specified in the php.ini file. Then you will need to edit the php.ini file and uncomment the following line:

```
extension=php_gd2.dll
```

This will enable access to the GD library. If you would like to load the extension dynamically with script you may do so with the following line of code:

```
dl('php_gd2.dll');
```

Caution

In order to dynamically load an extension, the shared object file (UNIX) or the dll file (Windows) must be in the extensions directory specified in the php.ini file. If it is not, then the loading of the extension will fail and none of the functions will be available to your game.

Let's say you don't know if the server you are hosting your files on is a UNIX or a Windows server. You can create a dynamically loading script that will work on either platform very easily. Just take a look at the following code:

```
<?php  
if(!extension_loaded('gd'))  
{  
    if(strtoupper(substr(PHP_OS, 3)) == "WIN")  
    {  
        dl('php_gd2.dll');
```

```
        }
    else
    {
        dl('gd.so');
    }
}
?>
```

This handy little chunk of code will work for any extension this example just happens to be for the GD library. The first step is to see if the extension is already loaded. If the extension is loaded then why would you want to do anything else? If, however, the extension is not loaded then you will need to determine what platform you are on and load the proper file. To determine what OS the PHP script is living on you will need to use the PHP_OS constant. This returns the information about the current OS. If you are on a Windows platform the first three characters of the string will be "WIN". If you do not find the string "WIN" in the PHP_OS constant it is safe to assume that you are on a UNIX platform. Once you know what platform you are on you can dynamically load the extension that you want by using the `dl()` function.

Now that you have the extension properly loaded it is time to test to see if it is actually there. The easiest way to do this is to use the `phpinfo()` function. Remember when you used this in Chapter 2? You will need to run that same test script again and look for a section in the page that looks like Figure 8.1.

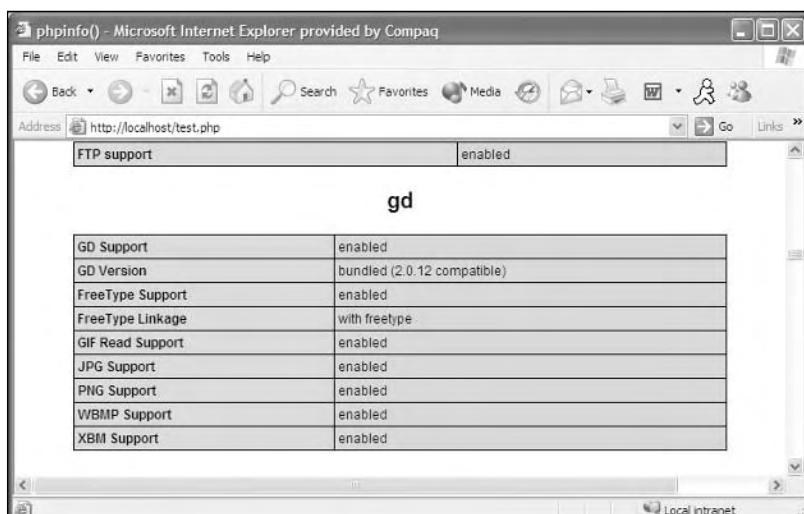


Figure 8.1 Results of the `phpinfo()` function.

Creating and Using a New Image

To create new images, GD offers you two functions. Both of these functions take two arguments, width and height, and they both return a resource to the image.

- `ImageCreate(width, height)`
- `ImageCreateTrueColor(width, height)`

GD also offers functions to create images from existing files, but you will learn about these later in this chapter. First take a look at the `ImageCreate()` function.

Note

A resource to an image is just the chunk of memory in which the image is being stored. In PHP you work in memory to manipulate images instead of a GUI like Paint Shop Pro.

`ImageCreate()` creates a new blank image with width number of pixels across, and height number of pixels tall. The color palette that `ImageCreate()` uses when creating an image is an indexed-color palette, meaning that the number of colors in the palette will be 256.

Remember in Chapter 3 when image types and compressions were discussed? The indexed-color palette uses a color lookup table to determine the specific color. These types of images are really good for images that use flat colors or text and they are especially good with simple shapes. You can save these images as .png files, and if GD ever supports the .gif file format again you will be able to save them as .gif files.

Caution

If you save an 8-bit image (also referred to as an indexed-color palette image) as a .jpeg you will end up with quite large file sizes and blotchy images.

Take a look at an example for creating a blank image:

```
<?php  
if(!extension_loaded('gd'))  
{  
    if(strtoupper(substr(PHP_OS, 3)) == "WIN")  
    {  
        dl('php_gd2.dll');  
    }  
    else  
    {  
        dl('gd.so');
```

```
    }  
}  
$imageResource = ImageCreate(100, 100);  
?>
```

You now have an image in memory that is 100 pixels wide and 100 pixels tall. You can now operate on this image using various functions provided by GD. Before you get into these functions, though, take a very quick look at the `ImageCreateTrueColor()` function.

The `ImageCreateTrueColor()` function also takes a width and height and it also returns a resource to an image in memory. However, it has one key difference: it creates an image with a true-color palette instead of an indexed-color palette. This makes this function ideal for creating complex graphics with lots of different elements in them. You would use this function to create .jpeg images. You can also save the image as a .png, but it will be saved as a true-color .png file instead of an indexed-color .png file.

Now that you know how to create an image resource, you will want to know how to edit this resource.

How to Use Colors

Before you can actually start drawing geometric shapes all over the place, you need the ability to select and use color. There are eight basic functions for color, as follows:

- `ImageColorAllocate(resource image, int red, int green, int blue)`
- `ImageFill(resource image, int x, int y, int color)`
- `ImageColorTransparent(resource image, int color)`
- `ImageTrueColorToPalette(resource image, bool dither, int colors)`
- `ImageColorsTotal(resource image)`
- `ImageColorAt(resource image, int x, int y)`
- `ImageColorsForIndex(resource image, int index)`
- `ImageColorSet(resource image, int index, int red, int green, int blue)`

Each of these functions takes an image resource as its first argument. Now take a more in-depth look at each of these functions to see how you will be able to use them to manipulate the colors on your canvas.

Allocating Colors to an Image

`ImageColorAllocate()` takes four arguments: an image resource, a value for the amount of red that should be in the color, a value for the amount of green that should be in the color,

and a value for the amount of blue that should be in the color. If you have never used RGB (red, green, blue) values before, each element of the color has a range of 0 to 255; 0 being the lowest level and 255 being the highest level. Let's say you wanted to allocate a color that was completely black. You would write a line of code that looks like the following:

```
$black = ImageColorAllocate($someImageResource, 0, 0, 0);
```

What this essentially means is that you will be able to use the variable \$black as a color in the specified image resource. If you were working with two images simultaneously, then you would not be able to use the variable \$black in both images. You can use only the colors that you allocate to a specific image. If this seems a little confusing, take a look at the following code example and hopefully it will make things a little clearer:

```
<?php
// Create two images
$image1 = ImageCreate(100, 100);
$image2 = ImageCreate(100, 200);
// Allocate the colors to be used
$black = ImageColorAllocate($image1, 0, 0, 0);
// This is legal to do
ImageRectangle($image1, 6, 6, 66, 42, $black);
// This is illegal to do because black is not allocated to the image
ImageRectangle($image2, 6, 6, 66, 42, $black);
?>
```

Filling the Image

The `ImageFill()` function also takes four arguments: an image resource, a starting x point, a starting y point, and the color to fill. `ImageFill()` does a flood fill of the entire image starting at the specified coordinates. Since this function fills the entire image there is no need to specify any other coordinate besides (0, 0). Take a look at the following example:

```
<?php
$image = ImageCreate(100, 100);
$red = ImageColorAllocate($image, 255, 0, 0);
ImageFill($image, 0, 0, $red);

// Show our Image
header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

Take a look at Figure 8.2 to see the results of using the `ImageFill()` function.

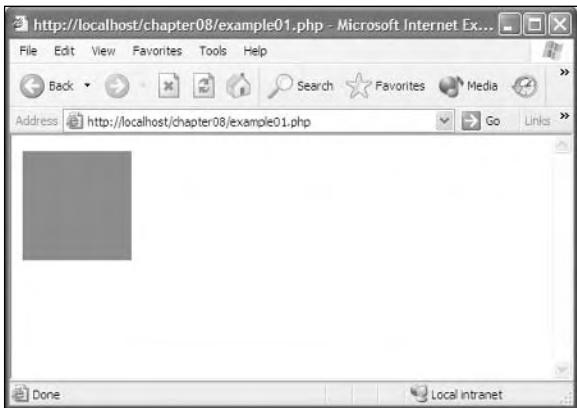


Figure 8.2 Results of using the `ImageFill()` function.

You could have specified any (x, y) coordinate and you would have received the same exact results. Don't worry too much about the display code right now; it will be covered later in this chapter.

Setting Your Transparent Color

If you are using an image format that supports transparencies, such as .png, you can set the color you want to show up as transparent by using the `ImageColorTransparent()` function (Figure 8.3). `ImageColorTransparent()` takes two arguments: the image resource, and the color that you want to make transparent. To allocate a color to be transparent you still use the `ImageColorAllocate()` function. To understand this better, take a look at the code below. In the following example a red circle is drawn over a dark background and then the color red is made transparent and the same image is drawn again.

```
<?php
$image = ImageCreate(128, 128);
$black = ImageColorAllocate($image, 0, 0, 0);
$red = ImageColorAllocate($image, 255, 0, 0);
// Make the background black
ImageFill($image, 0, 0, $black);
// Draw the circle
ImageFilledArc($image, 64, 64, 110, 110, 0, 360, $red, IMG_ARC_PIE);

// Show our Image Filled Image
ImagePng($image, "redcircle.png");

// Make the red transparent
ImageColorTransparent($image, $red);
```

```
// Show our Image Transparent Image  
ImagePng($image, "transparentcircle.png");  
  
ImageDestroy($image);  
?>  
<HTML>  
<HEAD>  
    <TITLE>Image Test</TITLE>  
</HEAD>  
<BODY>  
      
      
</BODY>  
</HTML>
```

Caution

You will need to change the permissions on the folder containing redcircle.png in order to use the example. Simply add read/write permissions to the IUS_(MACHINENAME) user under the security tab for the folder.

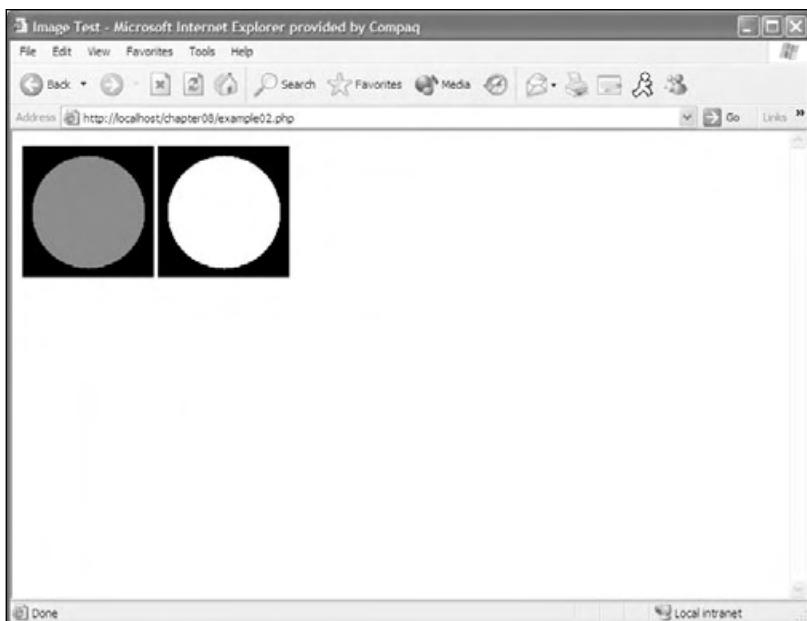


Figure 8.3 Using transparent colors.

How cool is that!? You are quickly becoming a graphic expert in PHP. Again, don't worry about the display code yet; I promise you will get to it later in this chapter.

Converting a True-Color Image to a Palette Image

Converting an image is not as complicated as it sounds. As a matter of fact, this is very easy using the `ImageTrueColorToPalette()` function. `ImageTrueColorToPalette()` takes three arguments, the first of which is the image resource. The second is a Boolean value if it is set to true, then dithering will be used; otherwise dithering will not be used. The third argument is the number of colors that should appear in the color palette of the new image.

You could use this function to convert a .jpeg image into an indexed-color image, but the decline in quality of the image would be very noticeable. The most logical reason to use this function would be if you have a .png image that uses only, say, 16 colors, but it was saved as a true-color image for some odd reason. Take a look at the following example:

```
<?php
// Get the image that needs to be converted
$image = ImageCreateFromPng('truecolorimage.png');
// Now convert the image, since it is only using 16 colors you only need to specify 16
colors
ImageTrueColorToPalette($image, true, 16);
// save the image
ImagePng($image, 'convertedimage.png');
?>
```

See, you converted an existing image from a true-color image to an indexed-color image and saved it as a new image all in three lines of code. I told you it wasn't as complicated as it sounded. Do some experimenting with this function on several different images to see the effects. Convert a .jpeg photo using dithering, and then do the same thing without using dithering to see the dramatic differences.

Counting Colors in an Image

To get the total number of colors in an image, you can use `ImageColorsTotal()`. `ImageColorsTotal()` takes one argument: the image resource for which you want to count the number of colors.

```
<?php
// get an Image
$image = ImageCreateFromPng('somegraphic.png');
// now count the colors in this Image
```

```
$totalColors = ImageColorsTotal($image);
echo("There are " . $totalColors . " colors in this image");
?>
```

Caution

The `ImageColorsTotal()` function works only with indexed-color images.

Retrieving a Color at a Point

The `ImageColorAt()` function takes three arguments: the image resource, an x point, and a y point. `ImageColorAt()` will return the index of the color at the specified (x, y) pixel. Once you have this index you can use the `ImageColorsForIndex()` function to retrieve the red, green, and blue components of the specific color.

```
<?php
$image = ImageCreateFromJpeg('someimage.jpg');
$colorIndex = ImageColorAt($image, 100, 20);
$colorArray = ImageColorsForIndex($image, $colorIndex);
$red = $colorArray['red'];
$green = $colorArray['green'];
$blue = $colorArray['blue'];

echo("Red: " . $red . "<br>Green: " . $green . "<br>Blue:" . $blue);
?>
```

Caution

The `ImageColorAt()` function works only with true-color images.

The `ImageColorsForIndex()` function takes two arguments: an image resource and a color index. It then returns an array with the individual red, green, and blue components in each index. Once you have these components you could use the `ImageColorAllocate()` function to allocate a color for an image, or you could use the `ImageColorSet()` function to change a color in an image.

The `ImageColorSet()` function takes five arguments. The first is the image resource that you want to operate on, the second is the index of the color you want to change, the third argument is the red component, the fourth is the green component, and the fifth and final argument is the blue component.

Drawing Basic Shapes on Your Empty Canvas

You now know how to create an image canvas that you can draw on, and you also know how to allocate, manipulate, and count colors in an image. Now you'll learn how to draw geometric shapes on your empty canvas.

Tip

If you have ever worked with DirectX you can think of the image resource as a buffer that you draw on. When you display the image you can think of that as blitting the buffer to the screen. The only difference is that you don't do this per frame.

To draw anything on your images you need a coordinate system. GD uses a Cartesian coordinate system where the point (0, 0) is in the upper left-hand corner of the image. Moving to the right along the x axis will move you in the positive x direction. Moving down along the y axis will move you in the positive y direction. You will never use a negative x or y coordinate when working with images.

Each of the eight geometric functions available in GD takes one or more sets of coordinates. You will be reviewing each of the eight functions in detail, so dust off all of those old geometry lessons and get ready. Here are the eight geometric functions, in order from easiest to hardest:

- `ImageSetPixel(resource image, int x, int y, int color)`
- `ImageLine(resource image, int x1, int y1, int x2, int y2, int color)`
- `ImageRectangle(resource image, int x1, int y1, int x2, int y2, int color)`
- `ImageFilledRectangle(resource image, int x1, int y1, int x2, int y2, int color)`
- `ImagePolygon(resource image, array points, int numberofPoints, int color)`
- `ImageFilledPolygon(resource image, array points, int numberofPoints, int color)`
- `ImageArc(resource image, int centerXPoint, int centerYPoint, int width, int height, int startDegree, int endDegree, int color)`
- `ImageFilledArc(resource image, int centerXPoint, int centerYPoint, int width, int height, int startDegree, int endDegree, int color, int style)`

Pixels and Lines

`ImageSetPixel()` takes four arguments: the first is the image resource, the second is the x point, the third is the y point, and the fourth argument is the color. The `ImageSetPixel()` function draws a single pixel at point (x, y) in the specified color. Take a look at the following example:

```
<?php  
$image = ImageCreate(320, 200);
```

```
$color = ImageColorAllocate($image, 254, 254, 254);
ImageFill($image, $color);

for($iLoop = 0; $iLoop < 1000; $iLoop++)
{
    $color = ImageColorAllocate($image, rand() % 256, rand() % 256, rand() % 256);
    ImageSetPixel($image, rand() % 320, rand() % 200, $color);
}

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

This example generates a 320×200 image, setting 1000 random pixels in a random color throughout the canvas. You use the random function to generate a random number and then perform a modulus on that number to keep it between the specified limits of the canvas size. Take a look at Figure 8.4 to see the results of the code.

Now that you can draw pixels to the screen, let's deal with a function that is a little more practical. The `ImageLine()` function draws a line to the canvas. It takes six arguments. The first is the resource to the image to which you are drawing the line. The second and third arguments are the starting coordinates of the line. The fourth and fifth arguments are the ending coordinates of the line. The final argument is the color that you want the line to be drawn in.

```
<?php
$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, $white);

ImageLine($image, 5, 5, 40, 100, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

In this example, you first create an image that is 320×200 pixels wide. Then you allocate two colors, white and black. You fill the canvas with the white color so your canvas is now all white. Then you draw a line from point (5, 5) to point (40, 100) in black and output the image to the browser.

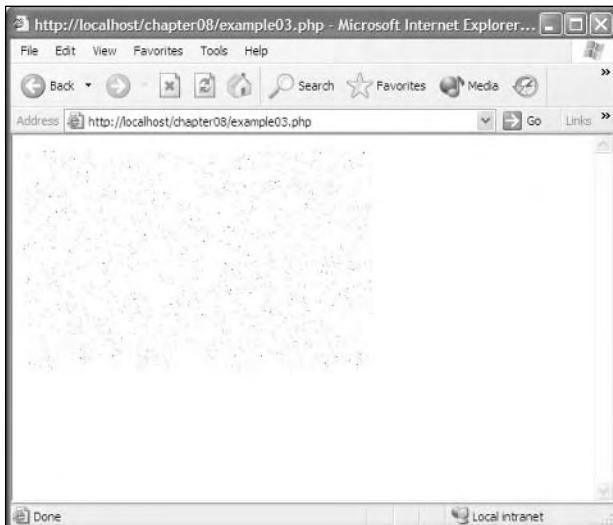


Figure 8.4 Generating 1000 random pixels.

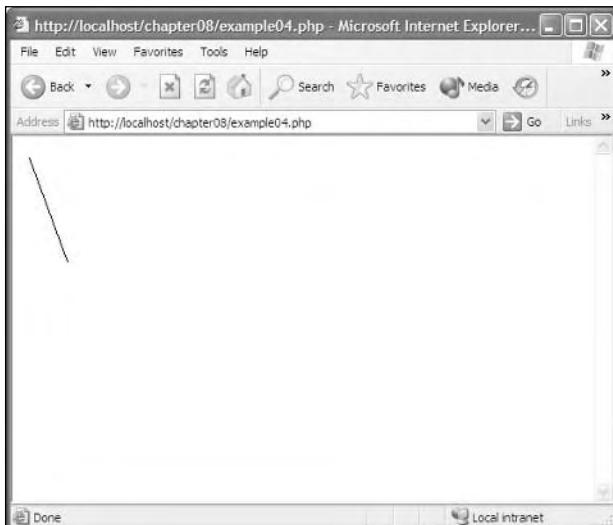


Figure 8.5 Drawing a line.

From Lines to Rectangles

`ImageRectangle()` takes the same four parameters as the `ImageLine()` function, except instead of the starting point of a line and an ending point for a line, it is the upper left-hand corner of the rectangle and the lower right-hand corner of the rectangle.

```
<?php  
$image = ImageCreate(320, 200);
```

```
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, $white);

ImageRectangle($image, 5, 5, 100, 100, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The results of this example are shown in Figure 8.5.

But wouldn't it be better to give the rectangle a starting (x, y) coordinate and a width and a height that you would like the rectangle to be? You can create a function that does exactly that.

```
<?php
function MyRectangle($image, $x1, $y1, $width, $height, $color)
{
    // First you need to calculate the values for x2 and y2
    $x2 = $x1 + $width;
    $y2 = $y1 + $height;

    // Now draw the rectangle
    ImageRectangle($image, $x1, $y1, $x2, $y2, $color);
}

$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, $white);

MyRectangle($image, 5, 5, 95, 95, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The results of this example look like Figure 8.6, but the function is much more intuitive. I recommend that anytime you see a way to make a function simpler by creating your own, then do it. Anytime that you can make code that makes more sense when you read it, you should do it.



Figure 8.6 The `ImageRectangle()` function.

`ImageFilledRectangle()` functions exactly like `ImageRectangle()`, except it creates a filled rectangle. `ImageFilledRectangle()` takes the same six parameters as `ImageRectangle()`. Take a look at the following code example to see the `MyFilledRectangle()` function:

```
<?php
function MyFilledRectangle($image, $x1, $y1, $width, $height, $color)
{
    // First you need to calculate the values for x2 and y2
    $x2 = $x1 + $width;
    $y2 = $y1 + $height;

    // Now draw the rectangle
    ImageFilledRectangle($image, $x1, $y1, $x2, $y2, $color);
}

$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, 0, 0, $white);

MyFilledRectangle($image, 5, 5, 100, 100, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The results of the code above are shown in Figure 8.7.

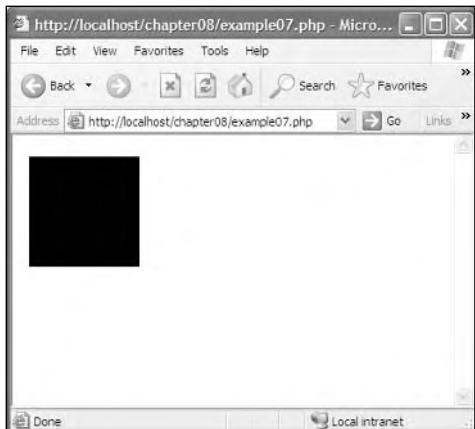


Figure 8.7 The MyFilledRectangle() function.

From Rectangles to Polygons

As mentioned earlier, the concepts of each geometric shape will get progressively more difficult. However, drawing a polygon isn't that complicated. The major difference between drawing a polygon and drawing a rectangle is the number of coordinates the function will take. So far you have dealt only with functions that take one coordinate (x, y). The `ImagePolygon()` function takes multiple coordinates. It also takes the number of points that are in your polygon. When the `ImagePolygon()` function draws your polygon it connects each point with a line.

`ImagePolygon()` takes four arguments. The first is, of course, the resource to the image. The next argument is an array that contains all of your point coordinates for the polygon. The third argument is an integer telling `ImagePolygon()` how many vertices are in your polygon. The fourth and final argument is the color in which you want your polygon to be drawn.

Let's say you wanted to draw a triangle on your canvas. You would need three sets of coordinates. You can store these coordinates in an array, for example:

```
$points = array(50, 10,  
                20, 40,  
                80, 40);
```

This gives you each corner of the triangle. Now you need to calculate how many points there are in the given polygon. With a triangle it is simple; there are obviously three points. But what would you do for a more complex shape? Well, since there are two elements in every point you can divide the size of the array by 2 to get the number of points in the polygon.

```
$vertices = sizeof($points) / 2;
```

Remember in the last section that I said anytime you see a way to make the code a little simpler, do it. Well, what if you created a function that took three arguments instead of four to create a polygon? That would be great, wouldn't it? You could save doing the number of vertices calculation every time you wanted to draw a polygon.

```
<?php
function MyPolygon($image, $points, $color)
{
    // Calculate the number of vertices in the polygon
    $vertices = sizeof($points) / 2;
    // Draw the polygon to the canvas
    ImagePolygon($image, $points, $vertices, $color);
}
?>
```

I know it doesn't seem like much now, but what if you were constantly drawing polygons and you had to redo that stupid division calculation every single time? Take a look at the following example to see how to draw a polygon:

```
<?php
function MyPolygon($image, $points, $color)
{
    // Calculate the number of vertices in the polygon
    $vertices = sizeof($points) / 2;
    // Draw the polygon to the canvas
    ImagePolygon($image, $points, $vertices, $color);
}
$points = array(50, 10,
                20, 40,
                80, 40);
$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, 0, 0, $white);

MyPolygon($image, $points, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The results of this example can be seen in Figure 8.8.

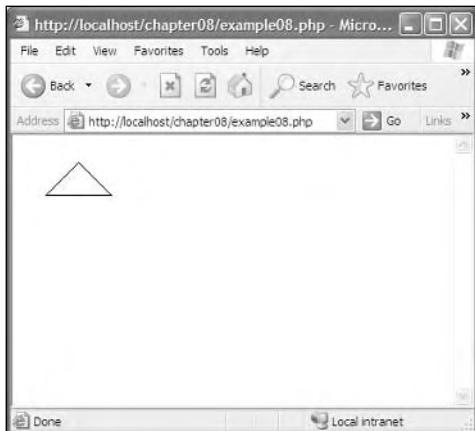


Figure 8.8 Drawing a polygon with the `ImagePolygon()` function.

The `ImageFilledPolygon()` function takes the same parameters as the `ImagePolygon()` function, but instead of drawing an outline of the polygon it draws a filled-in version of the polygon. Here is a modified function for the `ImageFilledPolygon()` function:

```
<?php
function MyFilledPolygon($image, $points, $color)
{
    // Calculate the number of vertices in the polygon
    $vertices = sizeof($points) / 2;
    // Draw the polygon to the canvas
    ImageFilledPolygon($image, $points, $vertices, $color);
}
?>
```

From Polygons to Arcs and Ellipses

Wow, you are quickly becoming a master at creating on-the-fly images with GD. After you learn how to create arcs and ellipses, you will move on to putting dynamic text in your graphics, and finally you will take a more in-depth look at how to display and save your on-the-fly images.

GD provides you with a function called `ImageArc()` to create, well, arcs. This function takes a whopping eight arguments. The first is the resource to the image. The next two arguments are the center x point for the arc and the center y point for the arc. The fourth and fifth arguments are the desired width of the arc, and then the desired height of the arc. The sixth argument is the starting degree for the arc. The seventh argument is the ending degree for the arc. The eighth and final argument is the color that you would like the arc to appear in.

Note

The `ImageArc()` function draws in a clockwise direction.

The `ImageArc()` function puts 0 degrees at three o'clock, 90 degrees at six o'clock, 180 degrees at nine o'clock, and 270 degrees at twelve o'clock. Although documentation on www.php.net says that the `ImageArc()` function draws counter-clockwise, it actually draws clockwise.

Take a look at how you would use the `ImageArc()` function and the results.

```
<?php
$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, 0, 0, $white);

ImageArc($image, 50, 30, 90, 90, 0, 220, $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The code example above draws an arc that is 90 pixels wide and 90 pixels high, with its center point at (50, 30). It starts at the 0 degree mark and ends at the 220 degree mark. Take a look at Figure 8.9 to see the results of the above code.

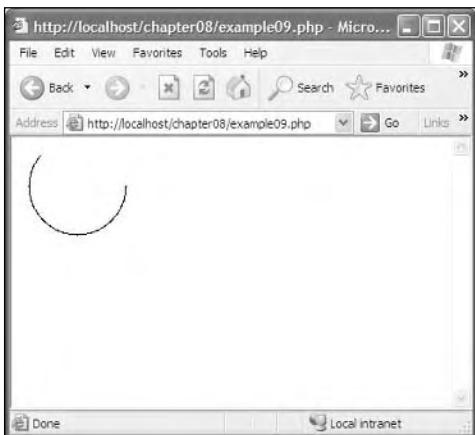


Figure 8.9 The `ImageArc()` function.

The `ImageFilledArc()` function behaves in the same way as the `ImageArc()` function. It also draws its arc starting from three o'clock and going in a clockwise direction. But the `ImageFilledArc()` function takes an additional argument. The ninth argument, after color, is the style in which the arc should be drawn. You can also use a bitwise OR to combine styles. Take a look at what the styles are and how they affect the arc.

- **IMG_ARC_PIE**. Draws a pie chart-styled arc with solid lines connecting the center point to the edges of the arc.
- **IMG_ARC_CHORD**. Draws a triangle that connects the beginning and end points of the arc.
- **IMG_ARC_NOFILL**. If this option is used, it behaves like the `ImageArc()` function.
- **IMG_ARC_EDGED**. Connects the end points of the arc to its center.

Let's use each of these individually in a code example and take a look at the results so you know how each style affects the outcome of your arc.

```
<?php  
$image1 = ImageCreate(320, 200);  
$image2 = ImageCreate(320, 200);  
$image3 = ImageCreate(320, 200);  
$image4 = ImageCreate(320, 200);  
  
$white1 = ImageColorAllocate($image1, 255, 255, 255);  
$white2 = ImageColorAllocate($image2, 255, 255, 255);  
$white3 = ImageColorAllocate($image3, 255, 255, 255);  
$white4 = ImageColorAllocate($image4, 255, 255, 255);  
$black1 = ImageColorAllocate($image1, 0, 0, 0);  
$black2 = ImageColorAllocate($image2, 0, 0, 0);  
$black3 = ImageColorAllocate($image3, 0, 0, 0);  
$black4 = ImageColorAllocate($image4, 0, 0, 0);  
  
ImageFill($image1, $white1);  
ImageFilledArc($image1, 50, 30, 90, 90, 0, 220, $black1, IMG_ARC_PIE);  
ImagePng($image1, "pie.png");  
ImageDestroy($image1);  
  
ImageFill($image2, $white2);  
ImageFilledArc($image2, 50, 30, 90, 90, 0, 220, $black1, IMG_ARC_CHORD);  
ImagePng($image2, "chord.png");  
ImageDestroy($image2);
```

```

ImageFill($image3, $white3);
ImageFilledArc($image3, 50, 30, 90, 90, 0, 220, $black3, IMG_ARC_NOFILL);
ImagePng($image3, "nofill.png");
ImageDestroy($image3);

ImageFill($image4, $white4);
ImageFilledArc($image4, 50, 30, 90, 90, 0, 220, $black4, IMG_ARC_EDGED);
ImagePng($image4, "edged.png");
ImageDestroy($image4);
?>
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
  </tr>
  <tr>
    <td align="left" valign="top">IMG_ARC_PIE</td>
    <td align="left" valign="top">IMG_ARC_CHORD</td>
  </tr>
  <tr>
    <td align="left" valign="top"></td>
    <td align="left" valign="top"></td>
  </tr>
  <tr>
    <td align="left" valign="top">IMG_ARC_NOFILL</td>
    <td align="left" valign="top">IMG_ARC_EDGED</td>
  </tr>
</table>

```

This produces the screen that you see in Figure 8.10.

The one obvious geometric shape missing from GD is a function to create an ellipse. Or is it? You can use the `ImageArc()` and `ImageFilledArc()` functions to create an ellipse. All you have to do is specify a starting degree of 0 and an ending degree of 360. This will create a full ellipse. For example:

```

<?php
$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, $white);

ImageArc($image, 50, 40, 70, 40, 0, 360, $black);

```

```
header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

This generates an ellipse that looks like Figure 8.11.

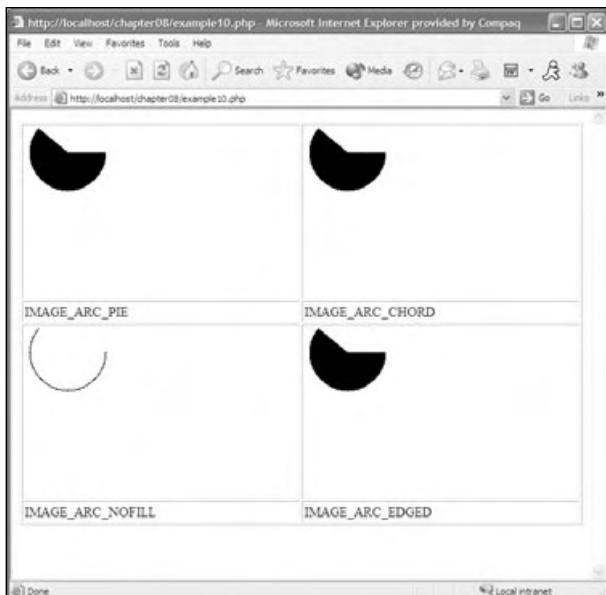


Figure 8.10 The `ImageFilledArc()` function.

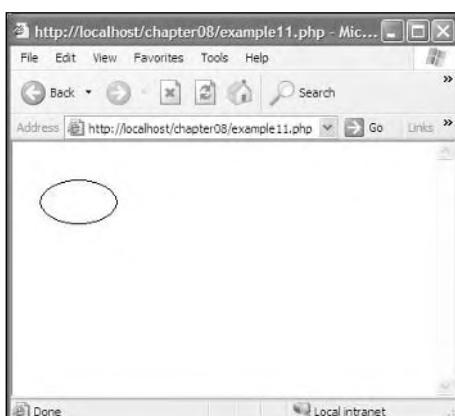


Figure 8.11 Creating an ellipse.

Creating Images with Text

You have seen how GD can handle drawing geometry on the canvas, but what if you have some text that you want to dynamically put into your graphic? GD provides you with three main functions for doing just that. They are:

- `ImageString(resource image, int fontNumber, int x, int y, string text, int color)`
- `ImageTTFText(resource image, int size, int angle, int x, int y, int color, string fontFile, string text)`
- `ImageTTFBBox(int size, int angle, string fontFile, string text);`

The `ImageString()` function is fairly straightforward. It takes six parameters. The first is the resource to the image. The second is a font number, from 1 to 5, that uses a built-in font to write out your text. The third and fourth arguments are the location that you want the text to start at. The fifth argument is the text that you want displayed. The final argument is the color in which you want the text to be displayed. This is the simplest of all three font functions and gives you the least amount of flexibility.

```
<?php
$image = ImageCreate(320, 200);
$white = ImageColorAllocate($image, 255, 255, 255);
$black = ImageColorAllocate($image, 0, 0, 0);
ImageFill($image, $white);

ImageString($image, 1, 10, 10, "Text In Font 1", $black);
ImageString($image, 2, 10, 30, "Text In Font 2", $black);
ImageString($image, 3, 10, 50, "Text In Font 3", $black);
ImageString($image, 4, 10, 70, "Text In Font 4", $black);
ImageString($image, 5, 10, 90, "Text In Font 5", $black);

header("Content-type: image/png");
ImagePng($image);
ImageDestroy($image);
?>
```

The code example above goes through each font size that can be rendered by the `ImageString()` function. Figure 8.12 shows the results of the code. As you can see, it isn't a very pretty font.

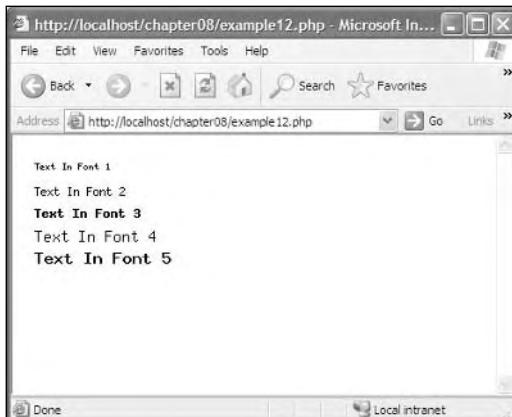


Figure 8.12 Adding text with the `ImageString()` function.

A more useful function for rendering fonts to your canvas is `ImageTTFText()`. The `ImageTTFText()` function takes eight arguments. The first is the resource to the image; the second argument is the size in which you want to render the text. The third argument is the angle at which you want to render the text. This will allow you to render text sideways, diagonally, and even upside down—it really is a fun argument to play with. The fourth and fifth arguments are the (x, y) coordinate that you want the text to start at. The sixth argument is the color you want to render the font in. The seventh argument is the True Type Font you want to use, and the final argument is the text you want to render.

The `ImageTTFText()` function returns an array with eight elements that represent the four points that make the bounding box of the text. Take a look at Table 8.1 to see what index points to which point.

Table 8.1 Bounding Box Coordinates

Index in the Array	Description
0	Lower left x position.
1	Lower left y position.
2	Lower right x position.
3	Lower right y position.
4	Upper right x position.
5	Upper right y position.
6	Upper left x position.
7	Upper left y position.

* You start in the lower left corner and work your way counter-clockwise around the box.

Tip

By default, `ImageTTFText()` renders anti-aliased text. If you would like the font to be rendered as aliased text you need to put a “-” (minus) sign in front of the color.

Now try rendering some text in a cool True Type Font. I am using Matisse ITC for this example.

```
<?php  
$image = ImageCreate(500, 200);  
$white = ImageColorAllocate($image, 255, 255, 255);  
$black = ImageColorAllocate($image, 0, 0, 0);  
ImageFill($image, 0, 0, $white);  
  
ImageTTFText($image, 36, 0, 0, 80, $black, "matisse_.ttf", "Cool Anti-Aliased Text!!!");  
ImageTTFText($image, 36, 0, 0, 150, -$black, "matisse_.ttf", "Cool Aliased Text!!!");  
  
header("Content-type: image/png");  
ImagePng($image);  
ImageDestroy($image);  
?>
```

Caution

To run properly, the font file must be located in the same directory as your PHP file.

In this example, I rendered both anti-aliased text and aliased text to show you the difference. The results are shown in Figure 8.13. As you can see, the aliased text is very jagged around the edges and not very pretty. I imagine there will be very few cases where you'll use the aliased font.

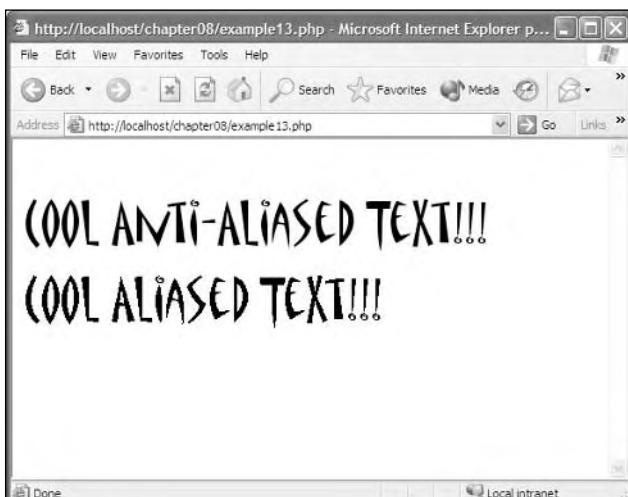


Figure 8.13 Rendering text with the `ImageTTFText()` function.

Caution

There is a bug in the GD library in PHP 4.2. The bug will cause `ImageTTFText()` to throw an error. This shouldn't be a problem since PHP 5 has been released. If you are still using PHP 4.2, upgrade!

I know you are saying, “Cool, but what if I want to do some font effects, like embossing?” You can do that with `ImageTTFText()` too. It just requires some careful placement of colors and text. Take the same example you just did but add a few more lines of code to it.

```
<?php  
$image = ImageCreate(500, 200);  
$white = ImageColorAllocate($image, 255, 255, 255);  
$black = ImageColorAllocate($image, 0, 0, 0);  
$gray = ImageColorAllocate($image, 155, 155, 155);  
ImageFill($image, $white);  
  
ImageTTFText($image, 36, 0, 2, 81, $black, "matisse_.ttf", "Cool Embossed Text!!!");  
ImageTTFText($image, 36, 0, 0, 79, $white, "matisse_.ttf", "Cool Embossed Text!!!");  
ImageTTFText($image, 36, 0, 0, 80, $gray, "matisse_.ttf", "Cool Embossed Text!!!");  
  
header("Content-type: image/png");  
ImagePng($image);  
ImageDestroy($image);  
?>
```

The first step in creating the effect that you see in Figure 8.14 is to render the black border around the text. To do this you just add 1 to the (x, y) coordinate. Next you want to render the text again with white to create a separation between the gray and the black. To do this you subtract 1 from the (x, y) coordinate. The final step is to render the text in gray at the specified (x, y) coordinate.

The final text function to learn is the `ImageTTFBBox()` function. This function takes four arguments. The first is the size of the font, then the angle, followed by the True Type Font file, and, lastly, the text you want to render. `ImageTTFBBox()` returns an array with the bounding box of the text. This is an extremely useful function when you need to make sure that the text you are going to render is within the image, unless you don’t mind cutting your text off in mid-sentence.

The returned array is ordered exactly the same as the `ImageTTFText()` function’s array. It starts at the lower left-hand corner of the box and works its way around in a counter-clockwise fashion.

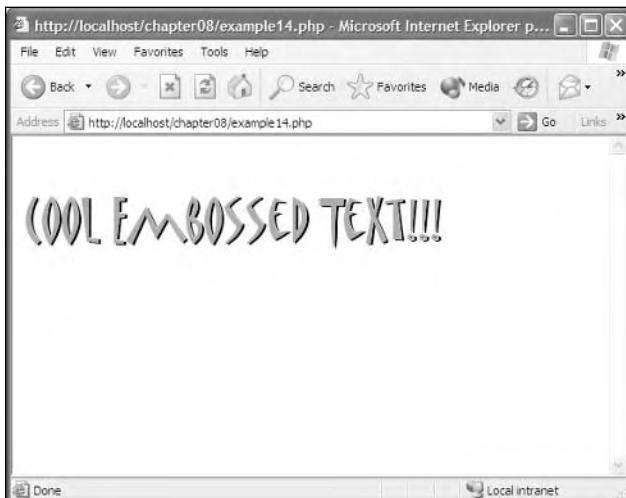


Figure 8.14 Creating cool text effects with the `ImageTTFText()` function.

Now create a function to put in our common.php file that will calculate the width and height of the bounding box for the text. Your function will also need to take four arguments, but you will want to return two integers. So your function will actually need to take six arguments.

```
<?php
function MyTTFBox($size, $angle, $fontfile, $text, &$width, &$height)
{
    // Get the bounding box
    $arrBox = ImageTTFBBox($size, $angle, $fontfile, $text);

    // Now calculate the width and the height of the box
    $width = abs($arrBox[6] - $arrBox[2]);
    $height = abs($arrBox[7] - $arrBox[3]);
}

$myWidth = 0;
$myHeight = 0;

MyTTFBox(36, 0, "matisse_.ttf", "Cool Embossed Text!!!", $myWidth, $myHeight);

echo("The Bounding box is $myWidth pixels by $myHeight pixels.");
?>
```

The `MyTTFBox()` function takes the same four arguments as the `ImageTTFBBox()` function plus two more arguments—the width and the height. The width and the height are passed by reference, meaning that anything that changes these variables in the function will also affect the results of the variables that were passed in. Take a look at Figure 8.15 to see the results of your new function.

Saving Your Images

Throughout the examples you have seen the use of `ImagePng()` and `ImageJpeg()`. These two functions are used to save images to a file and render images to the browser. There is a third function called `ImageWbmp()` that renders and saves WBMP files. All three of these functions take two arguments. The first is required: the resource to the image. The second argument is optional; you can specify a filename that you would like to save the image data to. Just make sure that if you save a file you use the proper extension. Don't try to save a `.png` file while using the `ImageJpeg()` function and vice versa.

```
// This saves a PNG file  
ImagePng($image, "myimage.png");  
// This saves a JPEG file  
ImageJpeg($image, "myimage.jpg");  
// This saves a WBMP file  
ImageWbmp($image, "myimage.wbmp");
```

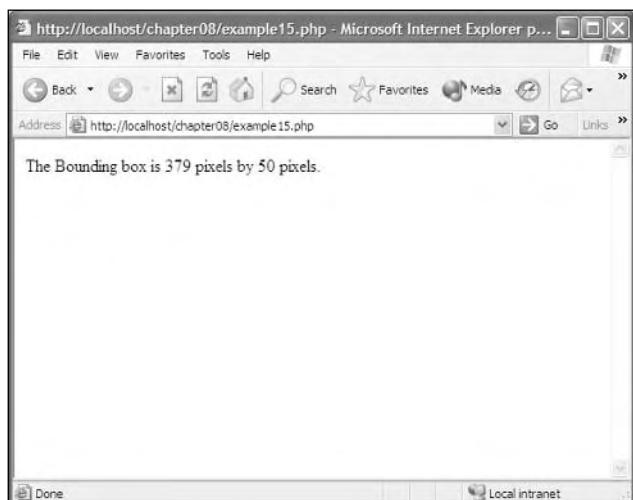


Figure 8.15 The `MyTTFBox()` function.

If you want to render the image directly to the browser you must specify a content type. You can do this by using the built in PHP header() function. To render a .png you have to specify a content type of image/png. To render a .jpeg you have to specify a content type of image/jpg. To render a WBMP you have to specify a content type of image/wbmp.

```
// This renders a PNG to the browser
header("Content-type: image/png");
ImagePng($image);
// This renders a JPEG to the browser
header("Content-type: image/jpg");
ImageJpeg($image);
// This renders a WBMP to the browser
header("Content-type: image/wbmp");
ImageWbmp($image);
```

Using Existing Images

Throughout this chapter you have seen how to create images, use colors, draw geometrical shapes to the canvas, and render text to your images. Now you will learn how to use existing images to create new images. There are six basic functions that you will use. The first two are the basis of using existing images to make new images.

- `ImageCreateFromJpeg(string filename)`
- `ImageCreateFromPng(string filename)`

Each of these functions returns a resource to an image. You cannot create an image from a WBMP, only from .pngs and .jpgs.

```
// Get a resource to an existing image
$image = ImageCreateFromPng("someimage.png");
```

After you have opened a file and retrieved a resource you can copy, resize, resample, or merge the image with another image with the following functions:

- `ImageCopy(resource destinationImage, resource sourceImage, int desinationX, int destinationY, int sourceX, int sourceY, int sourceWidth, int sourceHeight)`
- `ImageCopyResized(resource destinationImage, resource sourceImage, int desinationX, int destinationY, int sourceX, int sourceY, int destinationWidth, int destinationHeight, int sourceWidth, int sourceHeight)`
- `ImageCopyResampled(resource destinationImage, resource sourceImage, int desinationX, int destinationY, int sourceX, int sourceY, int destinationWidth, int destinationHeight, int sourceWidth, int sourceHeight)`

- `ImageCopyMerge(resource destinationImage, resource sourceImage, int destinationX, int destinationY, int sourceX, int sourceY, int sourceWidth, int sourceHeight, int percent)`

I have included two images on the CD called `dragon.jpg` and `frame.jpg`. You will use these images to create a new image using the `ImageCopy()` function. The images appear in Figure 8.16.

Now you will copy `dragon.jpg` onto `frame.jpg` using the `ImageCopy()` function.

```
<?php  
// Get our image resources  
$dragon = ImageCreateFromJpeg("dragon.jpg");  
$frame  = ImageCreateFromJpeg("frame.jpg");  
  
// Copy dragon onto frame  
ImageCopy($frame, $dragon, 21, 31, 0, 0, 477, 462);  
  
// Save the image and show it  
ImageJpeg($frame, "frameDragon.jpg");  
  
echo("<img src=frameDragon.jpg>");  
?>
```

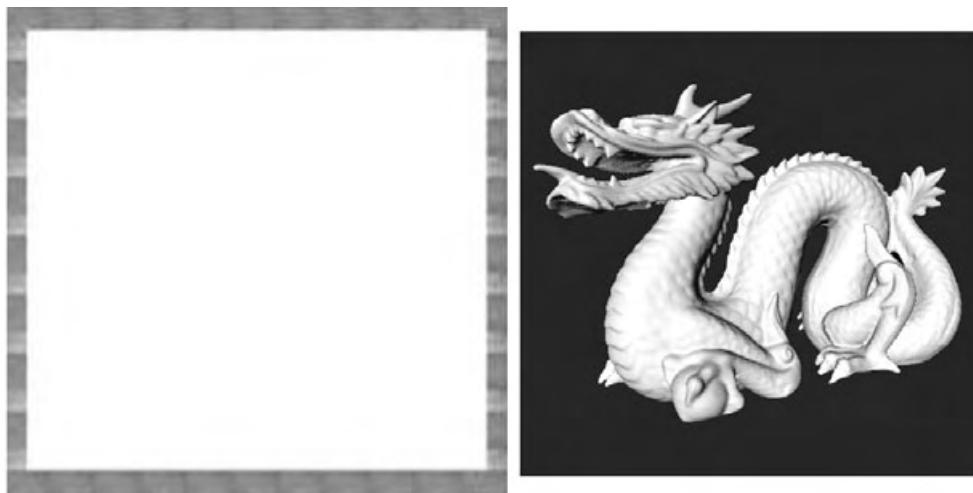


Figure 8.16 Images for use in the `ImageCopy()` function.

I'll break down each element of the `ImageCopy()` function in this example. The first argument is the destination image, which in this case is `$frame`. The second argument is the source image, which is `$dragon`. The next argument is what pixel you want to put the source image in in the destination image. Since the frame ends at the coordinate (21, 31), that is where you want to put the source image. The next argument is what pixel you want to grab the image at. Since the dragon image fits perfectly inside the frame, you just grab it starting at the coordinate (0, 0). The final arguments are the width and height of the source image. Since the dragon does fit perfectly within the frame, you can specify the actual width and height of the image. But what if the dragon was the same size as the frame?

If both the images were the same size, you would have to do a little math to get the same results as Figure 8.17. You would want to first grab the source image from the same point you were placing the image. In the example above, it would be (0, 0). Then you would want to subtract that amount from the overall width and height of the image. In the example above, it would be (456, 431). So the new `ImageCopy()` line would look like this:

```
ImageCopy($frame, $dragon, 21, 31, 0, 0, 456, 431);
```



Figure 8.17 Copying an image with the `ImageCopy()` function.

So now you know how to copy an image that is either the same size or smaller than the destination image. But what if you want to copy an image that is larger than the destination image? You would use the `ImageCopyResized()` or the `ImageCopyResampled()` functions. A good example would be to take the `framedDragon.jpg` image that you just created and create a 100×100 thumbnail of the image.

```
<?php
// Get our image resources
$image = ImageCreateFromJpeg("frameDragon.jpg");

// Create a true color image 100x100
$thumb = ImageCreateTrueColor(100, 100);

// Copy dragon onto frame
ImageCopyResized($thumb, $image, 0, 0, 0, 0, 100, 100, ImagesX($image),
ImagesY($image));

// Save the image and show it
ImageJpeg($thumb, "thumbFrameDragon.jpg");

echo("<img src=thumbFrameDragon.jpg>");
?>
```

This starts off the same way as the last example, by getting the resource to an existing image. Then you create a new true-color image that is 100×100 pixels. Now you copy the image `$image` to the destination image `$thumb`. The seventh and eighth arguments are the destination image's width and height. The ninth and tenth arguments are the source image's width and height. In this example you use the `ImagesX()` and `ImagesY()` functions that return the width and height of the image resource. A full listing of all image functions can be found in Appendix D. Take a look at Figure 8.18 to see the results of the example above.



Figure 8.18 Using the `ImageCopyResized()` function.

I mentioned earlier in this chapter that the `ImageCopyResized()` function and the `ImageCopyResampled()` function do the same thing. Well, technically they do the same thing, but with one minor difference. After the `ImageCopyResampled()` function copies and resizes the image, it also resamples it. The results are a crisper, cleaner image. In the upcoming code example you will resize the image using `ImageCopyResized()` and then `ImageCopyResampled()`. After they are resized you will save them and compare the results.

```
<?php
// Get our image resources
$image = ImageCreateFromJpeg("frameDragon.jpg");

// Create a true color image 100x100
$thumb = ImageCreateTrueColor(250, 250);
$thumb2 = ImageCreateTrueColor(250, 250);

// Copy dragon onto frame
ImageCopyResized($thumb, $image, 0, 0, 0, 0, 250, 250, ImagesX($image),
ImagesY($image));
ImageCopyResampled($thumb2, $image, 0, 0, 0, 0, 250, 250, ImagesX($image),
ImagesY($image));

// Save the image and show it
ImageJpeg($thumb, "thumbFrameDragon1.jpg");
ImageJpeg($thumb2, "thumbFrameDragon2.jpg");

echo("<img src=thumbFrameDragon1.jpg>&nbsp;&nbsp;<img src=thumbFrameDragon2.jpg>");
?>
```

The results of the code example are shown in Figure 8.19.

The difference is noticeable. The top image is the one made with `ImageCopyResized()`. It has quite a bit of artifacting and looks quite grainy compared to the bottom image, which was created with `ImageCopyResampled()`.

So why would you ever use `ImageCopyResized()`? Well, the only reason I can think of is speed. But since you aren't batching a ton of images at once, you might as well use the `ImageCopyResampled()` function whenever speed isn't a concern. After all, you do want to get the best results you possibly can.

The final function to look at is the `ImageCopyMerge()` function. This function behaves exactly like the `ImageCopy()` function with one very key difference. The extra parameter (int percent) tells the image how opaque the image should be on the destination image. This allows you to create some cool backgrounds or translucent images.

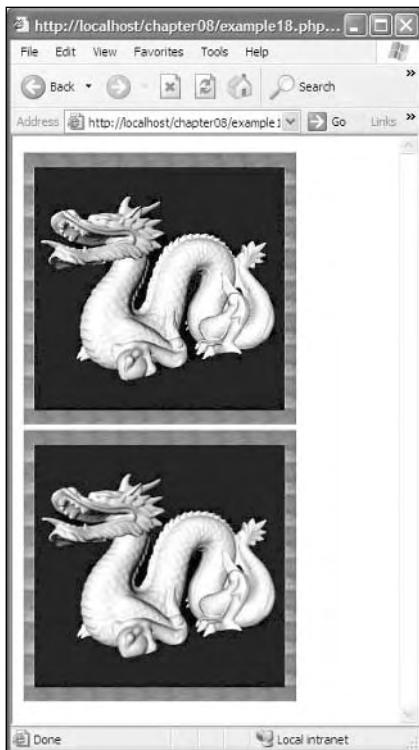


Figure 8.19 Results of the `ImageCopyResized()` [top] and `ImageCopyResampled()` [bottom] functions.

Let's merge our dragon image onto a white image with 20% opacity. This will allow most of the white to show through while displaying a faint image of the dragon.

```
<?php
// Get our image resources
$dragon = ImageCreateFromJpeg("dragon.jpg");

// Create a true color white image
$whiteImage = ImageCreateTrueColor(512, 512);
$white = ImageColorAllocate($whiteImage, 255, 255, 255);
$black = ImageColorAllocate($whiteImage, 0, 0, 0);
$gray = ImageColorAllocate($whiteImage, 155, 155, 155);
ImageFill($whiteImage, 0, 0, $white);
```

```
// Copy dragon onto frame
ImageCopyMerge($whiteImage, $dragon, 0, 0, 0, 0, ImagesX($dragon), ImagesY($dragon),
20);

ImageTTFFText($whiteImage, 36, 45, 201, 257, $black, "matisse_.ttf", "Dragon!!!");
ImageTTFFText($whiteImage, 36, 45, 199, 254, $white, "matisse_.ttf", "Dragon!!!");
ImageTTFFText($whiteImage, 36, 45, 200, 256, $gray, "matisse_.ttf", "Dragon!!!");

// Show the new image
header("Content-type: image/jpeg");
ImageJpeg($whiteImage);
ImageDestroy($whiteImage);
?>
```

The results of all of your hard work are shown in Figure 8.20.

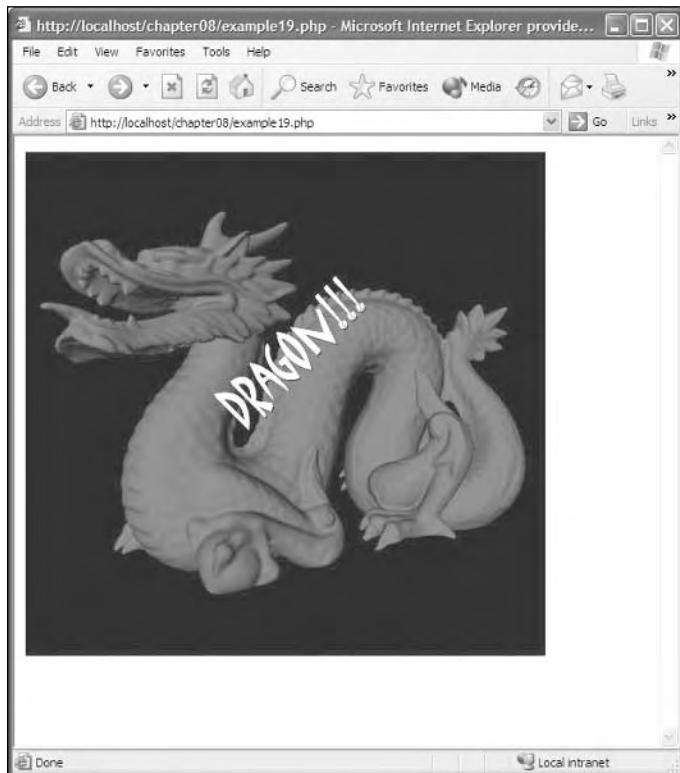


Figure 8.20 The results of your hard work!

Conclusion

That's it! You are now officially a dynamic image master in PHP. You learned how to create indexed and true-color images. You also learned how to allocate colors to an image. You became a god at drawing dynamic shapes onto your empty canvases. You also learned how to generate text from a True Type Font and use it in your image. Finally, you learned how to use existing images to your advantage to create some really cool-looking graphics.

Next up: creating a game called Battle Tank and creating dynamic terrain for your new tank game!

CHAPTER 9

CREATING BATTLE TA USING DYNAMIC TERRAIN



- Planning Battle Tank
- Creating the Graphics
- Creating the Game Logic
- Creating Dynamic Terrain

You have now become a master of using graphics. You have also become quite good at using HTML in conjunction with PHP. In this chapter you will create a Battle Tank game. Remember in the early 90s when an awesome game called Scorched Earth was released? Well, Battle Tank will be your rendition of the game. Are you ready?

Planning Battle Tank

Before you just jump in and start coding you need to plan out how you want your game to look and work. Since you want to create a Scorched Earth remake, you will need two tank graphics. You will need some sort of playing field. You will need to be able to input an angle and velocity so you can attempt to kill your opponent. All of that is just what the user will see. It has nothing to do with the logic of the game.

So what will you need for the logic of the game? Well, you will need some sort of calculation for your bullets. You will need to detect if the opponent has been hit. If the opponent has been hit you need to tell the user if he won or lost. You will also need to be able to control the computer. If the computer has not been hit then the computer gets to calculate a move and fire at the opponent.

You will also need to determine what game states you will have. A good start is assuming that you will need three game states. The three default game states you should have are:

- Game Starting
- Game Running
- Game Over

If you need more game states you can add them later, but for now let's assume those are the only three you will need. This seems like a good start.

Now you must determine what graphics you will need. Before you get to creating dynamic terrain you should create the game with all static images. You will want to define an image for your tanks, an image for your terrain, and an image for the explosion that will occur when a shot detonates. Just to make sure this is clear, you will need to define the following:

- Image for Tank #1
- Image for Tank #2
- Image for the Terrain
- Image for the explosion

When you create the explosion you will want to create an animated .gif. That way you have a pseudo effect of real-time animation. You will learn how to do this in the next section of this chapter, "Creating the Graphics."

Since you will be doing calculations that deal with angle and velocity you will need some sort of constant to pull down the bullet. That's right, dust off all those old physics books you will need to define gravity. If you need any other constants you can add them later just like with the game states.

So what are the objectives of the game? You want to generate a turn-based game that takes angle and velocity as the user input. Once you have the angle and velocity you calculate where the explosion will occur. If you hit the opponent, you win; if the opponent hits you, then you lose. After you have a working version of the game you'll want to go back and add dynamic terrain with all of your newfound graphics knowledge. Based on this description you will need six game states. You will use the three that you have already determined plus three more.

- Game Starting
- Game Menu
- Game Init
- Game Play
- Game Win
- Game Over

The game starting state tells the game to start the session then go to the menu. While you are on the menu you are in the game menu state. Once you click to start a new game, that takes you to the game init state. This isn't a big deal for the first version of the game, but it will come in handy when you start generating the terrain. Once you leave the game init state you enter the game play state. This is where you are physically playing that game. If you hit the opponent, then you enter the game win state, where you can display a cool "you win" graphic. If you get hit by the opponent, then you enter the game over state where you will put up a graphic telling the player that he has lost. Pretty simple, huh? Take a look at Figure 9.1 to see a screen shot of what the final product should look like when you are all through.

Creating the Graphics

For the most part I have created all the graphics for you and put them on the CD. You have two tank graphics: one for the left tank and one for the right tank. They are called tankLeft.gif and tankRight.gif on the CD. The tank graphics are the same tank, but one of them is a 180 degree mirror of the other. They are both .gif files because they have a transparent background; you could use .pngs if you like. The two tank graphics will be placed on the screen using some cascading style sheets and a `<div></div>` layer. Don't worry, I will explain all of that when you get to coding.

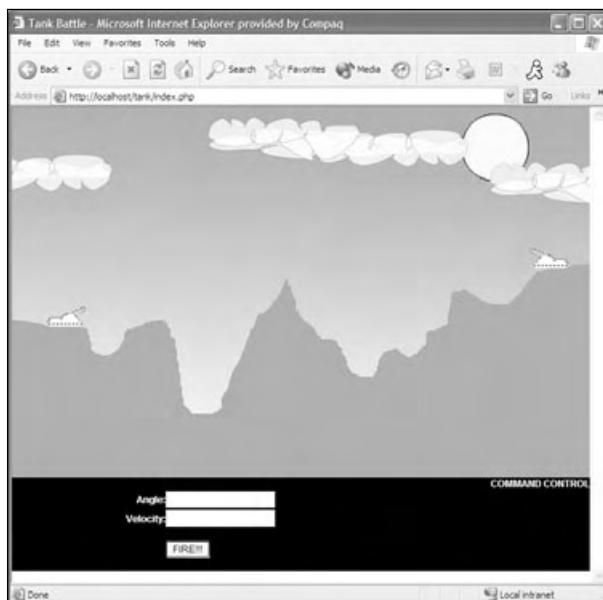


Figure 9.1 A screen shot of Battle Tank.

The next part of the graphic you will be using is the sky background. The sky background is called sky.jpg on the CD. The sky graphic is simply a gradient-blue background with some cool clouds and a sun on it. The clouds are partially opaque to give them the transparent look.

For this first version of the game I have created a graphic called terrain.jpg that is also included on the CD. Terrain.jpg contains the sky background with the pre-drawn ground over the background. That's essentially all the graphics that you will work with for this first version of the game.

Creating the Game Logic

Here is where you get down and dirty. This is where you take your plan and all your graphics and put them to work; this is where you actually code the game. The first step to take is to decide how you want to lay out your screen. Remember, you don't have to do this exactly like I do it. For this version I will have the game screen at the top of my table and the user input at the bottom of my table. This seems like a logical layout to me, but you can feel free to play with it however you like.

I am going to take this one step further and break out my rendering function into three functions: one for rendering the terrain, one for rendering the tanks, and one for rendering the actual user interface. I will call them `RenderTerrain()`, `RenderTanks()`, and `RenderInterface()`.

Let's start off with the `RenderTerrain()` function.

```
function RenderTerrain()
{
    printf("<table border=\"0\" cellpadding=\"0\" cellspacing=\"0\" width=\"700\""
height=\"450\" bgcolor=\"#000000\"");
    printf("<tr>");
    printf("<td align=\"left\" valign=\"top\">");
    printf("<img src=\"" . TERRAIN_IMAGE . "\" border=\"0\">");
    printf("</td>");
    printf("</tr>");
    printf("</table>");
}
```

All that the `RenderTerrain()` function does is create a table that is 700 pixels wide and 450 pixels high and displays the terrain graphic in the middle of the table. Notice that I have the terrain graphic defined as a constant. You will get to that shortly.

Take a look at the `RenderTanks()` function. The whole purpose of this function is to position the tanks over the background to make them look like they are in the scene. To do

this you will use a `<div></div>` tag for each tank and specify the pixels from the top and from the left where the tank should appear.

```
function RenderTanks()
{
    global $gLeftTankLocation;
    global $gRightTankLocation;

    // Get locations
    $gLeftTankLocation = $_SESSION['gLeftTankLocation'];
    $gRightTankLocation = $_SESSION['gRightTankLocation'];

    // Left tank
    printf("<div id=\"leftTank\" style=\"position: absolute; left: " .
        $gLeftTankLocation["x"] . "px; top: " . $gLeftTankLocation["y"] . "px;\">>");
    printf("<img src=\"" . LEFT_TANK_IMAGE . "\" border=\"0\\"");
    printf("</div>");

    // Right tank
    printf("<div id=\"rightTank\" style=\"position: absolute; left: " .
        $gRightTankLocation["x"] . "px; top: " . $gRightTankLocation["y"] . "px;\">>");
    printf("<img src=\"" . RIGHT_TANK_IMAGE . "\" border=\"0\\"");
    printf("</div>");
}
```

The first thing you do in this function is retrieve the location array from the session for each tank. Next, you print the actual HTML tag to position the tank. To position the tank absolutely you use a cascading style sheet. The cascading style sheet is defined with the `style` attribute. You want to position the tanks absolutely using the `position` element. Next, you will need to specify the number of pixels from the left of the browser and from the top of the browser. You can do this with the `left` and `top` element.

```
style="position: absolute; left: 200px; top: 200px;"
```

Now you can render the terrain and the tanks to the screen, but you still have no way to get the user input. That is where the `RenderInterface()` function comes in. The `RenderInterface()` function will render the form to the browser underneath the game.

```
function RenderInterface()
{
    printf("<table border=\"0\" cellpadding=\"0\" cellspacing=\"0\" width=\"700\" "
        "bgcolor=\"#000000\">>");

    printf("<tr>");
    printf("<td align=\"right\" valign=\"top\" colspan=\"2\">>";
```

```

printf("<b class=\"white\">COMMAND CONTROL</b>");
printf("</td>");
printf("</tr>");

printf("<tr>");
printf("<td align=\"right\" valign=\"middle\">");
printf("<b class=\"white\">Angle:</b>");
printf("</td>");
printf("<td align=\"left\" valign=\"top\">");
printf("<input type=\"text\" width=\"80\" name=\"angle\">");
printf("</td>");
printf("</tr>");

printf("<tr>");
printf("<td align=\"right\" valign=\"middle\">");
printf("<b class=\"white\">Velocity:</b>");
printf("</td>");
printf("<td align=\"left\" valign=\"top\">");
printf("<input type=\"text\" width=\"80\" name=\"velocity\">");
printf("</td>");
printf("</tr>");

printf("<tr>");
printf("<td align=\"left\" valign=\"top\">");
printf("&nbs");
printf("</td>");
printf("<td align=\"left\" valign=\"middle\"><br>");
printf("<input type=\"submit\" name=\"btnFire\" value=\"FIRE!!!\">");
printf("<br><br></td>");
printf("</tr>");

printf("</table>");
}

```

This creates a table that is 700 pixels wide with a black background. It puts three form elements in the table. The first form element is a text box to retrieve the angle. The second form element is a text box to retrieve the velocity. The third and final form element is a Submit button so you can tell your tank to fire.

That does it for the rendering functions for the game. Now take a look at the defines and the globals that the game will use.

```
<?php
// Game States
```

```
define("GAME_START", 0);
define("GAME_MENU", 1);
define("GAME_INIT", 2);
define("GAME_PLAY", 3);
define("GAME_WIN", 4);
define("GAME_OVER", 5);

// Constants for calculations
define("GRAVITY", -9.8); // m/s/s
define("PI", 3.14159);

// Images
define("TERRAIN_IMAGE", "images/terrain.jpg");
define("LEFT_TANK_IMAGE", "images/tankLeft.gif");
define("RIGHT_TANK_IMAGE", "images/tankRight.gif");
define("EXPLOSION_IMAGE", "images/explosion.gif");

// Globals
global $gGameState;
global $gDifficulty;
global $gLeftTankLocation;
global $gRightTankLocation;
?>
```

First you define the game states, then the constant for gravity. This will be used in your calculations for the bullet. Then you define all the images that you will use. For this first iteration of the game you will have only one level of difficulty, but you should add the global anyway so that when you do add more then one level of difficulty, the framework is already there.

Now that all of the globals and constants are declared you can move on to the `StartGame()` and `EndGame()` functions. The `StartGame()` function will start the session and initialize all the variables for the session. The `EndGame()` function will unset all the variables and end the session.

```
function StartGame()
{
    global $gGameState;
    global $gDifficulty;

    if($gGameState == GAME_START)
    {
        $gGameState = GAME_MENU;
    }
```

```

// Manage out session
session_start();
$bSession = $_SESSION['bSession'];
if(!isset($bSession))
{
    $bSession = 1;
    $_SESSION['bSession'] = $bSession;
    $_SESSION['gDifficulty'] = $gDifficulty;
}
else
{
    // Get the current game state
    $gGameState = $_SESSION['gGameState'];
}
}

function EndGame()
{
    global $gGameState;
    global $gDifficulty;
    global $gLeftTankLocation;
    global $gRightTankLocation;

    unset($gGameState);
    unset($gDifficulty);
    unset($gLeftTankLocation);
    unset($gRightTankLocation);
    unset($turn);
    session_destroy();
}

```

The first thing that occurs in the `StartGame()` function is that the game state is switched from `GAME_START` to `GAME_MENU`. This will take the user to the menu after the function has completed. Then the session is started. After the session is started, the `StartGame()` function checks to see if this is a new session or an existing session. If the session is new, it sets some variables. If the session already exists, it retrieves the game state.

The `EndGame()` function unsets all the global variables that were used throughout the game and destroys the current session. This is a handy function because this allows the user to start a completely new game without having to close the browser and open it again.

Once the player is dropped to the menu he can choose to start a new game. When a new game is started it switches the state to `GAME_INIT`, where the `GameInit()` function is

called. This function is used primarily to place the tanks in their starting positions and to switch the game state to GAME_PLAY.

```
function GameInit()
{
    global $gGameState;
    global $gLeftTankLocation;
    global $gRightTankLocation;

    // Set the tank locations
    $gLeftTankLocation = array("x" => 42, "y" => 243);
    $gRightTankLocation = array("x" => 628, "y" => 172);

    $_SESSION['gLeftTankLocation'] = $gLeftTankLocation;
    $_SESSION['gRightTankLocation'] = $gRightTankLocation;

    $gGameState = GAME_PLAY;
}
```

The exact location of the tanks must be determined based on how you created your terrain. For the preset terrain that I am using in this version of the game, the left tank is 42 pixels from the left of the browser and 243 pixels from the top of the browser. The right tank is 628 pixels from the left of the browser and 172 pixels from the top of the browser. These locations are important because they will be used in determining if a tank has been hit by a bullet or not.

Now take a look at the main game loop. This function is called every single time the page is loaded. It uses the current game state to determine what the game should do.

```
function Render()
{
    global $gGameState;
    global $gDifficulty;
    global $gLeftTankLocation;
    global $gRightTankLocation;

    // Get locations
    $gLeftTankLocation = $_SESSION['gLeftTankLocation'];
    $gRightTankLocation = $_SESSION['gRightTankLocation'];

    switch($gGameState)
    {
        case GAME_MENU:
```

```
{  
    // Display the menu  
    RenderMenu();  
  
    break;  
}  
case GAME_INIT:  
{  
    // Init the Game  
    GameInit();  
  
    // Update Screen  
    Render();  
  
    break;  
}  
case GAME_PLAY:  
{  
    // Get the input and calculate the hit point  
    if($_POST['btnFire'] != "")  
    {  
        $explosionCoords = CalculateFire($_POST['angle'], $_POST['velocity']);  
  
        // Check to see if there was a hit  
        if($explosionCoords["x"] >= $gRightTankLocation["x"] &&  
        $explosionCoords["x"] <= $gRightTankLocation["x"] + 47 && $explosionCoords["x"] >=  
        $gRightTankLocation["y"] && $explosionCoords["y"] <= $gRightTankLocation["x"] + 24)  
        {  
            // Hit the other tank  
            $gGameState = GAME_WIN;  
            Render();  
            return;  
        }  
  
        RenderExplosion($explosionCoords);  
    }  
  
    // Make the computer shoot  
    $explosionCoords = CalculateFire(-rand(30,50), -rand(70, 100));  
    if($explosionCoords["x"] >= $gLeftTankLocation["x"] &&  
    $explosionCoords["x"] <= $gRightTankLocation["x"] + 47 && $explosionCoords["x"] >=  
    $gLeftTankLocation["y"] && $explosionCoords["y"] <= $gLeftTankLocation["x"] + 24)
```

```
{  
    // Hit the other tank  
    $gGameState = GAME_OVER;  
    Render();  
    return;  
}  
RenderExplosion($explosionCoords);  
  
// Render the terrain  
RenderTerrain();  
  
// Render the tanks  
RenderTanks();  
  
// Render the interface  
RenderInterface();  
  
    break;  
}  
case GAME_WIN:  
{  
    // Let the player know they won  
    printf("WIN!!!");  
    break;  
}  
case GAME_OVER:  
{  
    // Let the player know the game is over  
    printf("GAME OVER");  
    break;  
}  
}  
  
// Update our game state  
$_SESSION['gGameState'] = $gGameState;  
}
```

This function is doing quite a bit. The first thing it does is retrieve the locations of the two tanks. After the locations of the tanks have been retrieved, it determines what state the game is currently in. If the game is in the GAME_MENU state, the menu is rendered to the browser. If the game is in the GAME_INIT state, the game is initialized by calling the GameInit() function that you saw earlier in this chapter.

The GAME_PLAY state is where the whole game takes place. First it takes the user's input. It knows that the user has clicked the Fire button in the command center and it retrieves the angle and velocity that the user entered. After that, it calculates where the explosion should occur. If the user has hit the tank on the other end of the field, the game state is switched to GAME_WIN, the screen is updated, and the function returns without processing anything else.

However, if the user misses, the computer now gets to shoot. The computer randomly calculates a coordinate that is within the playing field and calculates where its shot should explode. If the shot hits the user, then the game state is set to GAME_OVER, the screen is updated, and the function returns. If neither player was killed, the game returns control to the user and awaits another angle and velocity to be entered. How cool is that?!

To actually calculate the distance the bullet will travel you need to use a little projectile physics. First off, the velocity needs to be broken into velocity in the x direction and velocity in the y direction. To do this you multiply by the cosine and sine of the angle.

```
$vx = $velocity * cos($angle);
$vy = $velocity * sin($angle);
```

Next, you calculate the hang time of the projectile. (In this case the projectile is the shell that is fired from the cannon.) You do this by multiplying 0.204 by the y component of the velocity.

```
$time = 0.204 * $vy;
```

Now that you know the time the bullet is in the air you can calculate the maximum distance the bullet can go. To do this you multiply the x component of the velocity by the total time to get the maximum range.

```
$xMax = $vx * $time;
```

The final y coordinate can be calculated by multiplying the y component of the velocity by time and subtracting half of gravity multiplied by times squared.

```
$yMax = $vy * time - 0.5 * GRAVITY * ($time * $time);
```

That's it. That is how you calculate where the explosion should occur. Now that you have an x and y coordinate you need to just check to see if it is in the same range as the playing field. If it is not, you need to put it there. Now you can render the explosion using a cascading style sheet x pixels from the left of the browser and y pixels from the top of the browser.

Figure 9.2 shows a shot of the game in action (well, as much action as can be portrayed in a still frame).

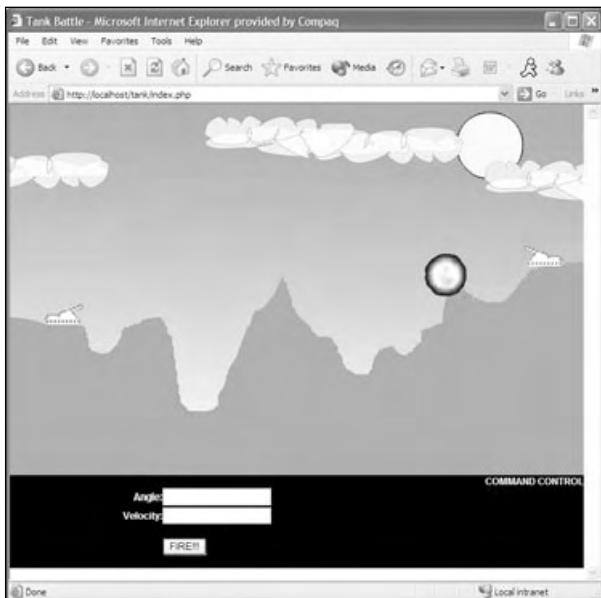


Figure 9.2 Battle Tank in action.

Caution

You will want to edit your php.ini file to turn off notices. You can do this by adding "& ~E_NOTICE" to the error_reporting line.

Creating Dynamic Terrain

Creating the dynamic terrain is actually quite easy. I have included the background as sky.jpg on the CD. You can use this to create your image. Once you have created your image you just need to set the first two points to 0 and 450. This will put the first coordinate at the bottom left of the graphic.

Once you have placed the very first coordinate at the bottom left of the graphic you need to start at the next pixel you would like to put your next point on and traverse the entire width of the image. While you are traversing the entire width of the image you will be generating two points. One is the x portion that should just be your loop counter, and the other is a randomly generated y coordinate.

After you have traversed the whole width of the image you will want to set the last two points. The last two points need to be the lower right corner of the image. In this case it is (700, 450). Now you are all set to generate a filled polygon with all of the points that you have just generated.

All of these procedures should go into the GameInit() function. Take a look at the following code example to see the new GameInit() function:

```
function GameInit()
{
    global $gGameState;
    global $gLeftTankLocation;
    global $gRightTankLocation;

    $terrain = ImageCreateFromJpeg("images/sky.jpg");

    $points[0] = 0;
    $points[1] = 450;
    $i = 2;
    for($x = 10; $x < 710; $x = $x + 20)
    {
        $points[$i] = $x;
        $points[$i+1] = rand(150, 350);

        $i = $i + 2;
    }

    $points[count($points)-1] = 700;
    $points[count($points)] = 450;

    $brown = ImageColorAllocate($terrain, 139, 164, 125);

    ImageFilledPolygon($terrain, $points, sizeof($points)/2, $brown);

    ImageJpeg($terrain, "images/temp.jpg");
    ImageDestroy($terrain);

    $leftY = (($points[3] + $points[5])/2) - ((($points[2] + $points[4])/2) - 10;
    $rightY = (($points[count($points)-4] + $points[count($points)-3])/2) -
              $points[count($points)-4];

    // Set the tank locations
    $gLeftTankLocation = array("x" => 42, "y" => $leftY);
```

```

$gRightTankLocation = array("x" => 628, "y" => $rightY);

$_SESSION['gLeftTankLocation'] = $gLeftTankLocation;
$_SESSION['gRightTankLocation'] = $gRightTankLocation;

$gGameState = GAME_PLAY;
}

```

The very first event that occurs in the GameInit() function is a new image is generated from the existing sky background by using:

```
$terrain = ImageCreateFromJpeg("images/sky.jpg");
```

Next you set the first set of coordinates.

```
$points[0] = 0;
$points[1] = 450;
```

Now that you are starting in the lower left corner of the image you need to start 10 pixels into the width of the image and traverse all the way across the image to 700.

```
$i = 2;
for($x = 10; $x < 710; $x = $x + 20)
{
    $points[$i] = $x;
    $points[$i+1] = rand(150, 350);
    $i = $i + 2;
}
```

This loop generates a random number every 20 pixels between 150 pixels and 350 pixels. This is what gives you the x and y points for the next logical coordinate.

Now that you have traversed all the way across the image you need to set the final point. Since you want the polygon to fill all the way to the bottom, you need to set the final point to (700, 450).

```
$points[count($points)-1] = 700;
$points[count($points)] = 450;
```

Now you simply draw your polygon onto the canvas by using the ImageFilledPolygon() function. After you have drawn the polygon you need to save the image to disk. The reason you save the image to disk is because you only want to generate new terrain when you first start a game. You don't want the terrain to be changing every time someone fires his cannon.

```

$brown = ImageColorAllocate($terrain, 139, 164, 125);

ImageFilledPolygon($terrain, $points, sizeof($points)/2, $brown);

ImageJpeg($terrain, "images/temp.jpg");
ImageDestroy($terrain);

```

Now all that is left is to position the tanks. In this example, the tank's x coordinate is always the same and the y coordinate is calculated with the average of two of the y points. This is not a totally accurate way to position the tank, but it does get the tank close enough. If you want a more accurate method you need to take three or four points and reposition the tank based on the x coordinate also.

```
$leftY = ((points[3] + points[5])/2) - ((points[2] + points[4])/2) - 10;  
$rightY = ((points[count($points)-4] + points[count($points)-3])/2) -  
    points[count($points)-4];  
  
// Set the tank locations  
$gLeftTankLocation = array("x" => 42, "y" => $leftY);  
$gRightTankLocation = array("x" => 628, "y" => $rightY);
```

Now that you've calculated the x and y coordinates for your tank positions you need to save that information to the session.

```
$_SESSION['gLeftTankLocation'] = $gLeftTankLocation;  
$_SESSION['gRightTankLocation'] = $gRightTankLocation;
```

After the coordinates are saved to the session all that is left to do is to start the game by switching the game state.

```
$gGameState = GAME_PLAY;
```

Figure 9.3 shows a screen shot of the game with some dynamic terrain being generated.

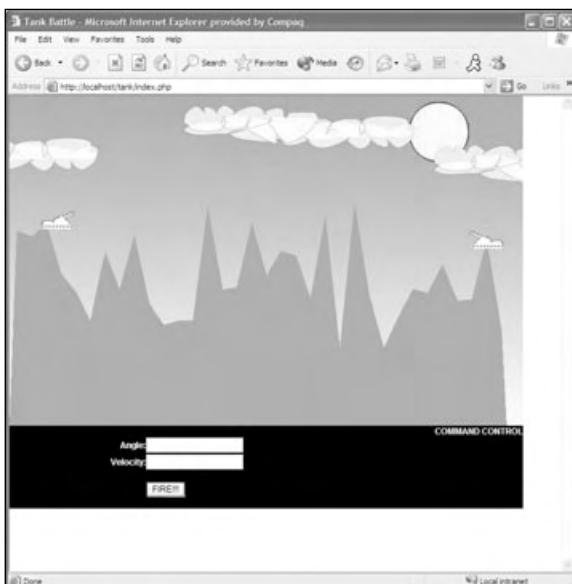


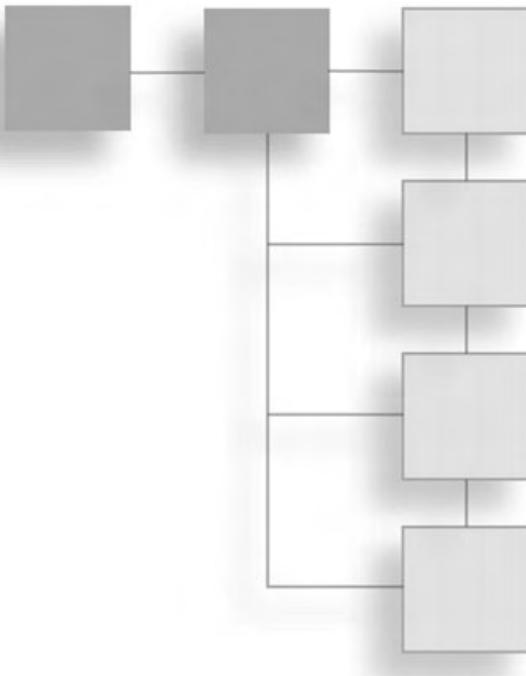
Figure 9.3 Generating dynamic terrain.

Conclusion

You have successfully created a fully dynamic PHP game that is a pretty good clone of the classic Scorched Earth. There are, however, several things that you can do to make this game a whole lot better. The first thing you can do is get the dynamic positioning of the tanks more accurate. The second thing you can do is add some collision detection to see if the bullet has hit the terrain. The third step you could take to make the game better is to dynamically remove chunks of terrain hit by the bullet. You could probably figure out something with an arc to do this.

You are getting extremely good at this PHP stuff, and you should be proud of yourself. Give yourself a big pat on the back. In the upcoming chapters you will perform some odds and ends. For example, in the next chapter you will look at how to use sockets in PHP. In Chapter 11 you will start to learn about MMORPG games and begin to create your own mini game. Sound like fun? Good, then let's get to it.

This page intentionally left blank



PART IV

EXTRAS AND PROJECTS

CHAPTER 10

PHP and Sockets	213
-----------------------	-----

CHAPTER 11

Kiddy Cartel—Creating Your Own MMO	229
--	-----

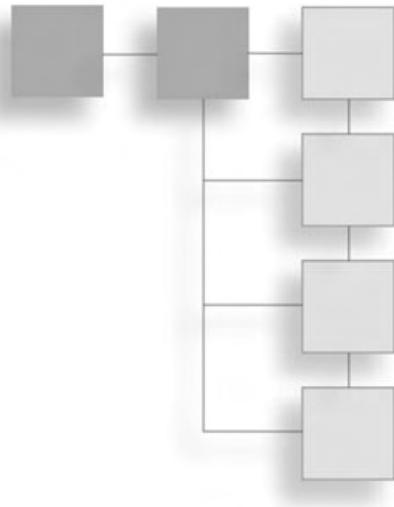
CHAPTER 12

Building Your PHP Skills	257
--------------------------------	-----

This page intentionally left blank

CHAPTER 10

PHP AND SOCKETS



- Socket Basics
- Creating a Server
- Creating a Client

In this chapter you will explore the fascinating and sometimes confusing world of sockets. Sockets must be one of the most underused components in PHP. Today you'll take a look at how to create a server to which a client can connect, how to connect to a server from a client using sockets, and how to process information on the server and send it to the destination client.

Believe it or not, you have been using sockets the entire time you've been programming in PHP. The server is the HTTP server that you connect to, and the client is the Web browser you are using to connect to the server. This is a single client/server relationship.

Socket Basics

PHP uses the Berkley sockets library to make its connections. You can think of a socket as nothing more than a data structure. You use this socket data structure to start a conversation between a client and a server. The server is always listening to open a new conversation. When a client wants to talk to the server, it opens a conversation through a specific port on which the server is listening. When the server receives the client's request it completes the connection, thus completing the cycle. Now the client can send information to the server and the server can send information to the client.

To create a socket you will need three variables: a protocol, a socket type, and a common protocol type. There are three protocols that you can choose from when creating a socket. Take a look at Table 10.1 for the names of the protocols and a description of what each protocol does.

When you create your own server you will use the AF_INET protocol. There are five socket types to choose from in the Berkley socket library. Please refer to Table 10.2 for the constant and a description of the socket type.

The final element to creating a socket is to define the type of common protocol that the connection should use. Table 10.3 lists the name and the description of the three common protocol types.

Table 10.1 Protocols

Name/Constant	Description
AF_INET	The most common of the protocols that are used when creating sockets. AF_INET uses TCP or UDP and an IPv4 address.
AF_INET6	Similar to the AF_INET but uses an IPv6 address instead of an IPv4 address.
AF_UNIX	A local communication protocol. This is specific to UNIX and Linux, and it uses the file system to define its socket connections. This protocol is rarely used. When it is used, the client and server are usually on the same machine.

Table 10.2 Socket Types

Name/Constant	Description
SOCK_STREAM	This socket type provides sequenced, reliable, full-duplex connections based on byte streams. This is the most commonly used socket type. This is also the socket type that the TCP common protocol uses.
SOCK_DGRAM	This socket type provides connectionless, fixed-length transmissions called <i>datagrams</i> . This socket type is fairly unreliable. UDP uses this socket type for its connections.
SOCK_SEQPACKET	This socket type provides a two-way, reliable connection for sending fixed-length transmissions. The receiver is required to read the entire packet for every read call made when using this socket type.
SOCK_RAW	This socket type provides raw network protocol access. The ICMP common protocol (ping, traceroute, and so on) uses this socket type.
SOCK_RDM	This socket type is rarely used and is not implemented on most operating systems. This provides a datagram layer that does not guarantee the ordering of your packets.

Table 10.3 Common Protocols

Name/Constant	Description
ICMP	The Internet Control Message Protocol. It is used mostly by gateways and hosts to report errors in communication.
UDP	The User Datagram Protocol. As mentioned previously, this is a connectionless, unreliable way to transmit data.
TCP	The Transmission Control Protocol. This is the most common and most reliable of the common protocols. TCP guarantees that its packets will arrive to the recipient. If there were errors during transmission, TCP re-broadcasts the packets to make sure they show up error free.

Now that you know the three elements for creating a socket, take a look at the `socket_create()` function that PHP uses to create a socket. The `socket_create()` function takes three parameters: a protocol, a socket type, and a common protocol. The `socket_create()` function returns a resource to the socket if it was created successfully, or it returns false if the socket was not created successfully.

```
resource socket_create(int protocol, int socketType, int commonProtocol);
```

Now that you can create a socket, how can you use it? PHP provides several functions to handle sockets. You can bind sockets to an IP, listen for communication on a socket, accept a socket; the list just goes on and on. Start by taking a look at an example to see what functions you will need to create, accept, and listen to a socket.

```
<?php
$commonProtocol = getprotobynumber("tcp");

$socket = socket_create(AF_INET, SOCK_STREAM, $commonProtocol);

socket_bind($socket, 'localhost', 1337);

socket_listen($socket);

// More socket functionality to come
?>
```

The example above is a start to creating your own server. The first line of the example,

```
$commonProtocol = getprotobynumber("tcp");
```

gets the protocol type that you are going to use by name. In this case you want to use the TCP common protocol. If you wanted to use UDP or ICMP you would pass “udp”

or “icmp” as the parameter to the `getprotobynumber()` function. An alternative to using the `getprotobynumber()` function is specifying either `SOL_TCP` or `SOL_UDP` as the final argument to the `socket_create()` function.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

The second line of the example creates the socket and returns the instance of the socket resource. After you have the instance of the socket resource, you need to bind the socket to a certain port and IP address.

```
socket_bind($socket, 'localhost', 1337);
```

In this case you are binding the socket to your local computer (127.0.0.1) and you are binding the socket to port 1337. After everything is created and bound you must listen for a connection to come in to the socket.

```
socket_listen($socket);
```

These are just four of the functions used in the wonderful world of sockets. Take a look at Table 10.4 to see all the functions that sockets use.

Table 10.4 Socket Functions

Function Name	Returns	Description
<code>socket_accept(resource socket)</code>	resource	After you have created a socket, bound it, and started listening on that socket, this function will accept incoming connections.
<code>socket_bind(resource socket, string address, int port)</code>	bool	This binds the socket resource to the address and port specified. <code>socket_bind()</code> will return TRUE if it succeeds and FALSE if it fails.
<code>socket_clear_error([resource socket])</code>	void	This will clear the errors on the specified socket. If a socket is not specified then it clears the global last socket error.
<code>socket_close(resource socket)</code>	void	Closes the created socket.
<code>socket_connect(resource socket, string address, [int port])</code>	bool	Attempts to connect to the socket resource. Returns TRUE if connection succeeded and FALSE if the connection failed.
<code>socket_create_listen(int port, [int backlog])</code>	resource	Creates a new AF_INET socket that listens on the specified port. Backlog defines the length of the queue of pending connections.

Table 10.4 Socket Functions (*continued*)

Function Name	Returns	Description
socket_create_pair(int protocol, int socketType, int commonProtocol, array &fd)	bool	This creates a pair of sockets that are stored in the array fd. There is no way to distinguish between the two sockets.
socket_create(int protocol, int socketType, int commonProtocol)	resource	Creates a new socket.
socket_get_option(resource socket, int level, int optname)	mixed	This function retrieves the options for a socket.
socket_getpeername(resource socket, string &address, [int &port])	bool	This function returns the remote IP address of the connecting peer computer.
socket_getsockname(resource socket, string &address, [int &port])	bool	This function returns the local IP address of the socket.
socket_iovec_add(resource iovec, int iov_len)	bool	This function adds a new vector to the scatter/gather array.
socket_iovec_alloc(int num_vectors)	resource	This function builds a iovec structure for use with sendmsg, recvmsg, writev, and readv.
socket_iovec_delete(resource iovec, int iov_pos)	bool	Deletes the allocated iovec.
socket_iovec_fetch(resource iovec, int iovec_pos)	string	Returns the data held in the iovec_pos in the specified resource.
socket_iovec_free(resource iovec)	bool	Frees the iovec resource.
socket_iovec_set(resource iovec, int iovec_position, string new_val)	bool	Sets the data at iovec_position to the new value.
socket_last_error([resource socket])	int	Retrieves the last error code that occurred on any socket. If a socket is specified, it returns the last error that occurred for that socket.
socket_listen(resource socket, [int backlog])	bool	Listens for a connection to the specified socket. Backlog defines the length of the queue for pending connections.
socket_read(resource socket, int length, [int type])	string	This reads length bytes from the specified socket. Type can be PHP_BINARY_READ or PHP_NORMAL_READ. If PHP_BINARY_READ is used, the string is a binary string; otherwise the string is a normal string with normal escape characters.
socket_readv(resource socket, resource iovec)	bool	Reads from the fd array using the scatter/gather array.

Table 10.4 Socket Functions (*continued*)

Function Name	Returns	Description
socket_recv(resource socket, string &buffer, int length, int flags)	int	Receives data into buffer on a connected socket.
socket_recvfrom(resource socket, string &buffer, int length, int flags, string &name, [int &port])	int	Receives data from a socket whether or not the socket is currently connected.
socket_recvmsg(resource socket, resource iovec, array &control, int &controllLength, int &flags, string & address. [int &port])	bool	This function is used to receive messages on a socket that uses iovectors.
socket_select(array &read, array &write, array & except, int tv_sec, [int tv_usec])	int	This function accepts an array of sockets to watch. The sockets that are passed in the <code>read</code> array are watched for characters that become available for reading. The <code>write</code> array is watched for blocks that are written to. The <code>except</code> arrays are watched for exceptions.
socket_send(resource socket, string buffer, int length, int flags)	int	This function sends data to a connected socket.
socket_sendmsg(resource socket, resource iovec, int flags, string address, [int port])	bool	Sends a message to a socket.
socket_sendto(resource socket, string buffer, int length, int flags, string address, [int port])	int	This function sends length of the buffer through the socket to the specified address.
socket_set_block(resource socket)	bool	Sets the blocking mode on a socket.
socket_set_nonblock(resource socket)	bool	Sets non-blocking mode on a socket.
socket_set_option(resource socket, int level, int optname, mixed optval)	bool	Sets the socket options for a socket.
socket_shutdown(resource socket, [int how])	bool	This function allows you to shut down reading, writing, or both for the specified socket. <code>how</code> can be 0, 1, or 2.
socket_strerror(int errorNumber)	string	Returns a description of the specified error number.
socket_write(resource socket, string buffer, [int length])	int	Writes the buffer to the socket.
socket_writev(resource socket, resource iovec)	bool	Writes to the <code>fd</code> array using the scatter/gather array.

* At the time of this writing, www.php.net did not have definitions for every socket function.

Those are all the functions that PHP offers for use with sockets. You should already have sockets enabled, but if you don't, then edit the php.ini file and uncomment the line that says:

```
extension=php_sockets.dll
```

If you don't uncomment this line you could load the extension dynamically using the following code:

```
<?php
if(!extension_loaded('sockets'))
{
    if(strtoupper(substr(PHP_OS, 3)) == "WIN")
    {
        dl('php_sockets.dll');
    }
    else
    {
        dl('sockets.so');
    }
}
?>
```

If you don't know whether sockets are enabled you can always use the `phpinfo()` function to determine if sockets are enabled. There should be a section that looks like Figure 10.1.

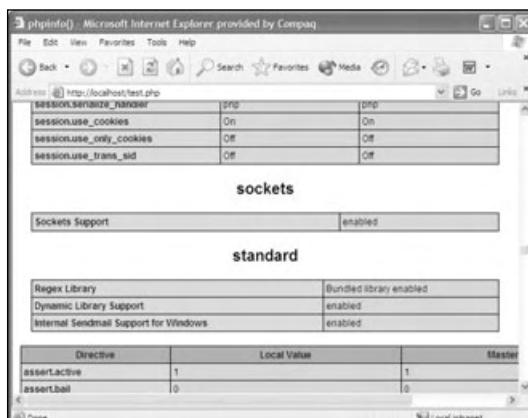


Figure 10.1 Viewing the `phpinfo()` for sockets.

Creating a Server

Now go back to the earlier example and complete it. To do this you simply need to listen to a particular socket and then process the user(s) connecting to it.

```
<?php  
$commonProtocol = getprotobynumber("tcp");  
  
$socket = socket_create(AF_INET, SOCK_STREAM, $commonProtocol);  
  
socket_bind($socket, 'localhost', 1337);  
  
socket_listen($socket);  
  
// Accept any incoming connections to the server  
$connection = socket_accept($socket);  
if($connection)  
{  
    socket_write($connection, "You have connected to the socket...\\n\\r");  
}  
?>
```

You will want to run this example from a command prompt. The reason for this is that you are creating a server, not a Web page. If you try to run this script from a Web browser, the script will most likely time out after 30 seconds. You could set an infinite time out by using the following line of code, but it is better to run the server from a command prompt:

```
set_time_limit(0);
```

To run your scripts from a command line you simply type:

```
php.exe example01_server.php
```

If you have not mapped the path to the PHP interpreter you will need to specify the path before php.exe. Once you have started the server you can test the connection by telneting to port 1337. You should see results that resemble Figure 10.2

The problem with this server is threefold: One, it doesn't accept multiple connections. Two, it performs only one command that is very useful. Finally, you cannot yet connect to this server through your Web browser.

The first problem is easy to solve if you were using an application that didn't have to reconnect to the server every time you clicked something. But since you are using a Web page to connect to the server, this poses a very large problem. You have to make your server accept a connection, write data to the client (if there is any to write), close the connection, and wait for another connection.

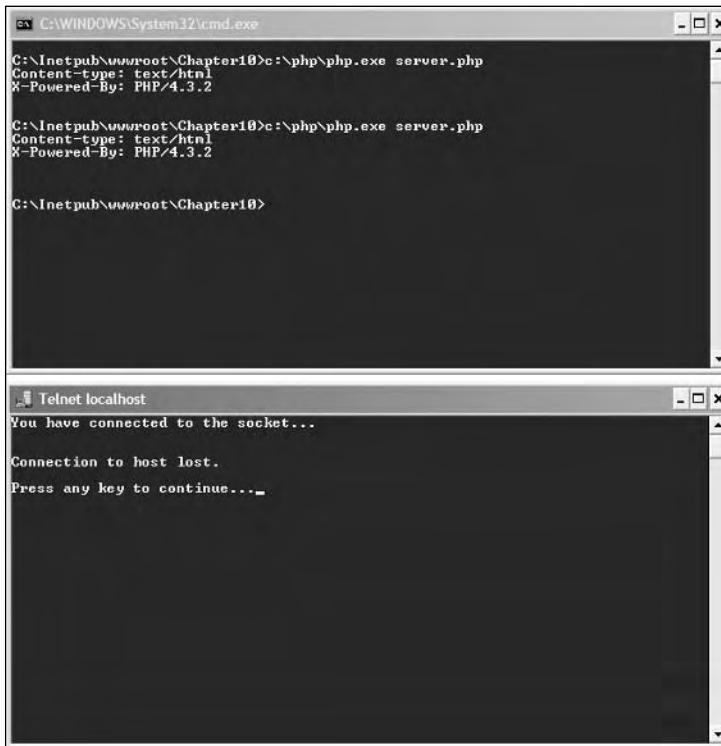


Figure 10.2 Your first server.

Let's do exactly that. Take a look at the following code example to see the new server:

```
<?php
// Set up our socket
$commonProtocol = getprotobynumber("tcp");

$socket = socket_create(AF_INET, SOCK_STREAM, $commonProtocol);

socket_bind($socket, 'localhost', 1337);

socket_listen($socket);

// Initialize the buffer
$buffer = "NO DATA";

while(true)
{
    // Accept any connections coming in on this socket
```

```
$connection = socket_accept($socket);
printf("Socket connected\r\n");

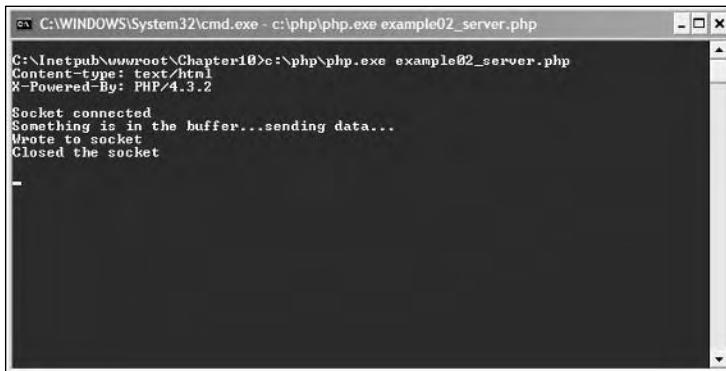
// Check to see if there is anything in the buffer
if($buffer != "")
{
    printf("Something is in the buffer...sending data...\r\n");
    socket_write($connection, $buffer . "\r\n");
    printf("Wrote to socket\r\n");
}
else
{
    printf("No Data in the buffer\r\n");
}

// Get the input
while($data = socket_read($connection, 1024, PHP_NORMAL_READ))
{
    $buffer = $data;
    socket_write($connection, "Information Received\r\n");
    printf("Buffer: " . $buffer . "\r\n");
}

socket_close($connection);
printf("Closed the socket\r\n\r\n");
}
?>
```

This is what the server does. It initializes the socket and the buffer that you use to receive and send data. Then it waits for a connection. Once a connection is created it prints “Socket connected” to the screen the server is running on. The server then checks to see if there is anything in the buffer; if there is, it sends the data to the connected computer. After it sends the data it waits to receive information. Once it receives information it stores it in the data, lets the connected computer know that it has received the information, and then closes the connection. After the connection is closed, the server starts the whole process again.

Take a look at Figure 10.3 to see the results of the server.



The screenshot shows a Windows Command Prompt window titled 'C:\WINDOWS\System32\cmd.exe - c:\php\php.exe example02_server.php'. The window displays the following text:
C:\Inetpub\wwwroot\Chapter10>c:\php\php.exe example02_server.php
Content-type: text/html
X-Powered-By: PHP/4.3.2
Socket connected
Something is in the buffer...sending data...
Wrote to socket
Closed the socket

Figure 10.3 Results of the new server.

Creating the Client

To solve the second problem is very easy. You need to create a PHP page that connects to a socket, receive any data that is in the buffer, and process it. After you have processed the data in the buffer you can send your data to the server. When another client connects, it will process the data you sent and the client will send more data back to the server.

Note

This is simply an example to demonstrate a use of sockets. This is by no means production-level code to be used on a public game.

```
<?php
// Create the socket and connect
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
$connection = socket_connect($socket,'localhost', 1337);

while($buffer = socket_read($socket, 1024, PHP_NORMAL_READ))
{
    if($buffer == "NO DATA")
    {
        echo("<p>NO DATA</p>");
        break;
    }
    else
    {
```

```
// Do something with the data in the buffer
echo("<p>Buffer Data: " . $buffer . "</p>");
}

}

echo("<p>Writing to Socket</p>");
// Write some test data to our socket
if(!socket_write($socket, "SOME DATA\r\n"))
{
    echo("<p>Write failed</p>");
}

// Read any response from the socket
while($buffer = socket_read($socket, 1024, PHP_NORMAL_READ))
{
    echo("<p>Data sent was: SOME DATA<br> Response was:" . $buffer . "</p>");
}
echo("<p>Done Reading from Socket</p>");
?>
```

This example client connects to the server using all the code that you have seen before. The client reads the data. If this is the first time through the loop on the first connection, then the server will send “NO DATA” back to the client. If this occurs, the client continues on. The client sends its data to the server. After the data has been sent to the server the client waits for a response. Once it receives a response it writes the response to the screen.

Caution

You will want to edit your php.ini file to turn off notices. You can do this by adding “& ~E_NOTICE” to the error_reporting line.

Tip

When using PHP_NORMAL_READ the line of data is considered done once the \r\n escape characters have been sent.

Integrating Sockets with Battle Tank

Now that you have all this cool code that deals with sockets, you can integrate it into the Battle Tank game you created in the previous chapter. To do this is very simple. Obviously, the first thing you will want to do is connect to the socket every time you click the Fire

button or start a new game. If you click the Fire button you will want to send your data to the server.

Once your data has reached the server you will want to query the server for the next coordinates of the bullet, and then render the game with the new coordinates. Here is the new Render() function for Battle Tank:

```
function Render()
{
    // Create the socket and connect
    $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
    $connection = socket_connect($socket,'localhost', 1337);

    global $gGameState;
    global $gDifficulty;
    global $gLeftTankLocation;
    global $gRightTankLocation;

    // Get locations
    $gLeftTankLocation = $_SESSION['gLeftTankLocation'];
    $gRightTankLocation = $_SESSION['gRightTankLocation'];

    switch($gGameState)
    {
        case GAME_MENU:
        {
            // Display the menu
            RenderMenu();
            break;
        }
        case GAME_INIT:
        {
            // Init the Game
            GameInit();

            // Update Screen
            Render();

            break;
        }
        case GAME_PLAY:
        {
```

```
//printf(session_id());  
  
// Get the input and calculate the hit point  
if($_POST['btnFire'] != "")  
{  
    $explosionCoords = CalculateFire($_POST['angle'], $_POST['velocity']);  
  
    if(!socket_write($socket, $explosionCoord[0] . "," .  
$explosionCoords[1] . "\r\n"))  
    {  
        echo("<p>Write failed</p>");  
    }  
  
    // Check to see if there was a hit  
    if($explosionCoords["x"] >= $gRightTankLocation["x"] &&  
$explosionCoords["x"] <= $gRightTankLocation["x"] + 47 && $explosionCoords["x"] >=  
$gRightTankLocation["y"] && $explosionCoords["y"] <= $gRightTankLocation["x"] + 24)  
    {  
        // Hit the other tank  
        $gGameState = GAME_WIN;  
        Render();  
        return;  
    }  
  
    RenderExplosion($explosionCoords);  
}  
  
while($buffer = socket_read($socket, 1024, PHP_NORMAL_READ))  
{  
    if($buffer == "NO DATA")  
    {  
        echo("<p>NO DATA</p>");  
        break;  
    }  
    else  
    {  
        // Do something with the data in the buffer  
        $explosionCoords = split($buffer, ",");  
    }  
}
```

```
// Make the other players shot
if($explosionCoords[0] >= $gLeftTankLocation["x"] && $explosionCoords[0]
<= $gRightTankLocation["x"] + 47 && $explosionCoords[1] >= $gLeftTankLocation["y"] &&
$explosionCoords[1] <= $gLeftTankLocation["x"] + 24)
{
    // Hit the other tank
    $gGameState = GAME_OVER;
    Render();
    return;
}
RenderExplosion($explosionCoords);

// Render the terrain
RenderTerrain();

// Render the tanks
RenderTanks();

// Render the interface
RenderInterface();

break;
}
case GAME_WIN:
{
    // Let the player know he won
    printf("WIN!!!!");
    break;
}
case GAME_OVER:
{
    // Let the player know the game is over
    printf("GAME OVER");
    break;
}
}

// Update the game state
$_SESSION['gGameState'] = $gGameState;
}
```

That's it! As I said before, this is not the best way to do this, but it works for demonstrating PHP's socket capabilities. There are some cool things you could do by creating sockets that talk to Flash pieces through your own PHP server, which is a more common way to use sockets.

Conclusion

Whether or not you realize it, you have done a tremendous amount of learning in this chapter. If you would like to learn more about sockets I suggest picking up a more in-depth book on sockets. You can also refer to www.php.net, where you'll find more in-depth explanations of all the functions you've seen in this chapter.

In the next chapter you will be creating your own MMORPG game. Sound like fun? Well then, turn the page and get moving!

CHAPTER 11

KIDDY CARTEL—CREATING YOUR OWN MMO



- Installing mySQL
- Relational Databases: A Quick Rundown
- Kiddy Cartel: The Rules and Specifications
- Creating Your Base Actions
- Creating a Command with Sub-Commands
- Creating a Command without Sub-Commands
- Look at All of the Commands...Now What?

In this chapter you will learn what goes into a massively-multiplayer online (MMO) game. At the end of this chapter you will have created an awesome MMO game called Kiddy Cartel. The object of this game is to take on the persona of a kid and build an empire through purchasing lemonade stands, mowing lawns, and attacking competing neighborhoods. Sound like fun? Let's get started.

First, a few notes about Kiddy Cartel. This game uses turns. That means that every time you perform an action it takes n number of turns. It is also a text-based game, so if you want graphics you will need to add them. Each account you create will control only one neighborhood, so if you want to control multiple neighborhoods you'll need to create multiple accounts.

Also, this game will need to use a relational database, so what better relational database than mySQL? It is freely available on www.mysql.com, and mySQL Version 4 is included on the CD that came with this book. You will also need a library call PEAR. PEAR stands for PHP Extension and Application Repository.

If you installed PHP from the CD, then PEAR and its default libraries and manager are already installed on your machine if you are running a UNIX platform. If you're running a Windows platform you will need to install the PEAR manager.

To install the PEAR manager, simply run the go-pear.bat file in the setup directory after you have installed PHP. If you did not install PHP from the CD included in this book, then I suggest you go to <http://pear.php.net> and view the installation section of the documentation to install PEAR.

Installing mySQL

Installing mySQL on Windows is a very simple task. All you need to do is to run the setup.exe program that comes with the Windows distribution. Once you have installed mySQL you will need to run the WinMySQLAdmin executable to set up the my.ini file.

To install mySQL on Linux, the easiest way is to use an RPM package. For the purpose of this book you will only need to use the server and client RPM packages. If you would like to install other packages you can download them at www.mysql.com.

To install the client and server as a standard install you should use the following line:

```
%> rpm -i MySQL-server-VERSION.i386.rpm MySQL-client-VERSION.i386.rpm
```

Once the package is finished installing you should find a new directory under /var/lib/ called mysql. The mySQL install also adds the proper lines to the /etc/init.d/ file so the mySQL server will start on boot.

Note

If you are upgrading from a previous version of mySQL you will need to stop the current running service in order to install the new mySQL.

Once mySQL is installed you need to grant permission to a user and to tables. If you skip this step you will only be able to connect to your databases as root. You can add a user later but I recommend that you do this now. On Windows you need to type the following at the command prompt:

```
C:\> cd\mysql\scripts  
C:\> mysql_install_db  
C:\> cd\mysql\bin  
C:\> mysqld_safe --user=mysql &
```

Where --user=mysql you can put any user you want. For example, --user=mmodb would set up a user called mmodb.

If you are installing mySQL on a Linux machine you will need to type the following at a command prompt:

```
%> ./scripts/mysql_install_db  
%> cd mysql_installation_directory  
%> ./bin/mysqld_safe --user=mysql &
```

If you have problems creating your user and granting your user permissions to the database you should look at the documentation at www.mysql.com to see suggestions for troubleshooting.

Now go ahead and create the database that Kiddy Cartel will use to store its data. Go to a command prompt and navigate to the directory where you installed mySQL. Once you get in that directory, type the following:

```
./bin/mysqladmin create kiddycartel
```

This will create a new database called kiddycartel. Now you need to add a table to the database. To do that, first take a quick look at the T-SQL language.

Relational Databases: A Quick Rundown

Relational databases are databases that know of other tables. You absolutely need to use a relational database to make a MMO game. The reason for this is that MMO games quickly get complicated and if you are always trying to loop through files to find a record, your game will suffer in performance and I guarantee that you will lose track of what data you are trying to store and retrieve.

With that said, relational databases use a language called T-SQL to operate on databases. T-SQL allows you to create tables in your databases, and to insert, update, and delete records, just to mention the most common functions. With these four options you can get started on creating the Kiddy Cartel game, although these four operations will not make you an expert in relational databases.

Note

This is not a complete tutorial of T-SQL. If you would like to learn more about relational databases you should explore some relational database books and online tutorials.

To create a table using T-SQL you need to use the CREATE TABLE function. The CREATE TABLE function takes the column definitions as parameters and a table name. The easiest way to explain how to create a table is to show you. Open up a command prompt and navigate to the mysql\bin directory. Once you are in that directory, type the following:

```
mysql
```

This should start the mySQL command prompt. Once you are in the mySQL command prompt you need to connect to a database, so type the following:

```
connect kiddycartel
```

This will connect you to the Kiddy Cartel database. Now that you are in the database you can add the necessary tables to the database. You will need to enter the following lines into the command prompt to create the neighborhoods table:

```
CREATE TABLE neighborhoods
(
    PlayerName          CHAR(64) NOT NULL PRIMARY KEY,
    NeighborhoodName    CHAR(64) NOT NULL,
    Password            CHAR(64) NOT NULL,
    Bullies              INT    DEFAULT 0,
    PaperBoys            INT    DEFAULT 0,
    LawnMowers           INT    DEFAULT 1,
    LemonadeStands       INT    DEFAULT 0,
    Bakeries             INT    DEFAULT 0,
    Money                INT    DEFAULT 300,
    Lemonade             INT    DEFAULT 100,
    Cookies              INT    DEFAULT 50,
    WaterBalloons        INT    DEFAULT 0,
    PlasticBats          INT    DEFAULT 0,
    SlingShots            INT    DEFAULT 0,
    LastTurnCredited     DATETIME NOT NULL,
    Turns                INT    DEFAULT 50
);
```

This will create a 16-column table called neighborhoods. The first column in the database stores the player name and is called PlayerName. You specify the data type of the column after you define the name of the column. The PRIMARY KEY keywords tell mySQL that this is a unique column and that is what should be used to identify the records in this table.

Note

The Kiddy Cartel database consists of only one table. This game is for example purposes only and not to show you proper relational database design.

Now that you have created the database and a table, you can insert records into the table. To insert a record into the table you will use the `INSERT INTO` function.

```
INSERT INTO neighborhoods(PlayerName, NeighborhoodName, Password, LastTurnCredited)
VALUES('MyPlayer', 'MyTown', 'mmorocks', GetDate())
```

This will insert a record that has the values MyPlayer as the player name, MyTown as the neighborhood name, mmorocks as the password, and the current date and time for the last turn credited column. The rest of the columns will contain the default values specified in the table definition.

Let's say you want to update the MyPlayer record and change its password. To do that you will need to use the `UPDATE` function. The `UPDATE` function takes the table name and then the column you want to update. For example:

```
UPDATE neighborhoods SET Password = 'newPassword' WHERE PlayerName = 'MyPlayer'
```

This will update the record associated with MyPlayer. Deleting a record is just as simple. You use the `DELETE FROM` function. The `DELETE FROM` function takes a table name as its main parameter.

```
DELETE FROM neighborhoods WHERE PlayerName = 'MyPlayer'
```

The line above will delete the MyPlayer record. If you did not specify a `WHERE` clause in the `DELETE FROM` function it would delete all the records in the specified table. So make sure you always specify a `WHERE` clause unless you don't mind losing all of your data.

That is a very quick rundown of some basic T-SQL functions. By no means is this a complete how-to on relational databases. But now that you have a general idea about what T-SQL is, and now that you have a database created for Kiddy Cartel, let's take a more in-depth look at the specifications for Kiddy Cartel.

Kiddy Cartel: The Rules and Specifications

Since Kiddy Cartel is a massively-multiplayer game, each player will be controlled by a human; there will be no computer players. Once you have created a neighborhood to control you will be given a stock of lemonade and a stock of cookies. You will also be given one lawn mower so you can generate more money. You will start off with \$300 and 50 turns.

The object of the game is to get as many resources and as much money as possible. To do this you will need to create lemonade stands, bake cookies, mow lawns, spy, and take over other neighborhoods. There are three main rules that apply to the coding of this game.

1. Turns – A certain number of turns will be generated for a player each hour. Players can stockpile turns for as long as they want.

2. Resource Generators – You will need to generate resources. Each resource generator is able to generate a specified amount of resources.
3. Actions – This is the heart of the game. Each action will take a number of turns. An action can require a number of resources or a number of minions.

To play this game you will need some sort of interface. You will be using forms to execute all the actions in the game. You can see the interface that will be used for the game in Figure 11.1. Take a look at the actions the player can perform during the course of the game.

- Purchase a Lemonade Stand
- Purchase a Simple Bake Oven
- Purchase a Lawn Mower
- Purchase Water Balloons
- Purchase Plastic Bats
- Purchase a Slingshot
- Create a Bully
- Create a Paper Boy
- Mow Lawns
- Mix Lemonade



Figure 11.1 The Kiddy Cartel game interface.

- Bake Cookies
- Destroy a Property
- Scare a Kid
- Steal Property
- Spy on a Kid
- Sabotage a Property
- Send a Bribe

Creating Your Base Actions

Each action that you can take requires some command to be executed. Since each command will need the same base variables you will create a command class from which each subsequent command can inherit.

Since each command can have a parent command, you will need a variable to store the parent. You will also need a way to execute the commands. It would make sense to make each command execute its own functions instead of having the `CCommand()` class execute the command functions. Each command could possibly have sub-commands, so you also need the ability to store those. Since you will store sub-commands you need to keep track of which command you are currently on. If anything goes wrong with the command, you need to roll back anything that you have done. After all, you don't want partial data floating around during the game.

Now that you know what each command needs to do, you can start to plan how you will code the class. The way I see it, each command can have four possible states. The command could error, it could be waiting for input, it could have more sub-commands to process, or it could be done processing. So let's define some constants.

```
// define some standard return values for commands.
// Specific commands can return whatever they like.
// Commands that are executed by CGame must return one of these four values.
define("CMD_ERROR", 1);      // an error occurred
define("CMD_CONTINUED",2);   // the command has more steps to execute before it finishes
define("CMD_FINISHED", 3);   // the command has finished executing
define("CMD_NEEDS_INPUT", 4); // the command needs human input before it can continue
```

In the comments you see a reference to a class called `CGame()`. The `CGame()` class will handle running the game and will be covered later in this chapter.

Next let's create the `CCommand()` class. You know that you have to track the parent object, the current command, the sub-commands of the parent object, and whether the command needs to be rolled back.

```
class CCommand
{
    // this is a reference to a parent object
    var $m_pParent = null;

    // the index of the current sub-command that is being executed
    var $m_nCurrentCommand;

    // if true, the command needs to be rolled back.  if false,
    // the command does not
    var $m_bNeedRollBack;

    // an array to hold the sub-commands
    var $m_arraySubCommands;

    function CCommand( &$parent )
    {
        // set the reference to our parent
        $this->m_pParent = &$parent;

        // set the current command to the first one
        $this->m_nCurrentCommand = 0;

        // initially, commands don't need to be rolled back
        $this->m_bNeedRollBack = false;
    }

    function Execute( $args )
    {
        // each command must deal with handling its own execution
        return CMD_FINISHED;
    }

    /*
    This function handles the return value from a sub-command.
    It is responsible for incrementing the current command index
    and for determining whether this command has anything
    left to execute.
}
```

```
/*
function HandleSubCommand( $result )
{
    // did it finish?
    if ($result == CMD_FINISHED)
    {
        // yes, so go to the next one
        $this->m_nCurrentCommand++;
    }
    // was there an error?
    else if ($result == CMD_ERROR)
    {
        // yes, so propagate the error upwards
        return CMD_ERROR;
    }
    // does the command need input?
    else if ($result == CMD_NEEDS_INPUT)
    {
        // yes, so just propagate the need for input upwards
        return CMD_NEEDS_INPUT;
    }

    // have we already executed everything
    // we need to for this command?
    if ($this->m_nCurrentCommand >=
count( $this->m_arraySubCommands ))
    {
        return CMD_FINISHED;
    }
    // we still have more to do!
    else
    {
        return CMD_CONTINUED;
    }
}

function OnError( $sError )
{
    // if we have a parent, let him deal with it
    if ($this->m_pParent != null )
```

```
{  
    $this->m_pParent->OnError( $sError );  
}  
// otherwise, we *are* the parent, so we have to deal with it  
else  
{  
    // print out the error message  
    echo "An error occurred: $sError<BR>";  
  
    /*  
    We roll us back, which, assuming all our child classes  
    implemented OnRollBack correctly, will undo everything.  
    */  
    $this->OnRollBack();  
}  
}  
  
// the default implementation of the OnRollBack command  
// assumes that the command  
// performs no processing that affects the roll back.  
// if this does not hold, a  
// command must override this method in order to  
// take the processing into account  
function OnRollBack()  
{  
    // since this command doesn't perform any processing  
    // outside of subcommand,  
    // we can simply call rollback on each of its  
    // sub-commands in reverse order  
    for($i = $this->m_nCurrentCommand; $i >= 0; $i--)  
    {  
        // unroll the sub-command if it's set.  
        // the isset check is  
        // to make sure that we don't try to  
        // roll back commands that  
        // don't have sub-commands but don't override OnRollBack.  
        if (isset($this->m_arraySubCommands[$i]))  
        {  
            $this->m_arraySubCommands[$i]->OnRollBack();  
        }  
    }  
}
```

```
// reset the current command pointer  
$this->m_nCurrentCommand = 0;  
}  
  
}
```

When a command object is created, it calls the `CCommand()` constructor. This constructor initializes all the member variables of the class. In this case the parent object gets set, the number of sub-commands is set to 0, and the rollback required variable is set to false.

Earlier I mentioned that each parent command would handle its own execution. So the only thing that the `Execute()` function needs to do is to tell the game that it is finished executing.

Now you need a way to handle the sub-commands that need processing. What better name for a function than `HandleSubCommand()`? The `HandleSubCommand()` class takes a single argument that tells the handler what state the command is executing in. First you see if the command has finished. If the command finished successfully then you can move on to the next sub-command in the tree. If the command did not finish then it must be in another state. So you check to see if an error in execution has occurred. If an error has occurred, then you need to tell the game to stop its execution. If an error has not occurred, then you need to check to see if the command is awaiting input. If the command is waiting for input, then you need to tell the game to hold up and wait for the user.

If you get past all of that checking you must then check to see if there are more commands to execute. If there are more commands to execute, then you need to tell the game that it needs to continue to the next command. If you have executed all of the possible sub-commands then you are finished executing.

What happens if an error occurs during execution? If an error does occur during the execution of a command, you need to call the `OnError()` function. There are two states where an error can occur: 1) if the command has a parent, and 2) if the command doesn't have a parent. If the executing command has a parent command, then the parent command should handle the error. But if the executing command has no parent, then the `CCommand()` class needs to clean itself up by calling the `OnRollBack()` function. All the `OnRollBack()` function does is clear any sub-commands that there might be and then resets the count.

You might be thinking that looks like an awful lot of code for a class that really doesn't do anything. To tell you the truth, this class will allow you to keep consistency between all of your commands. Each time you create a command, that command will inherit all the functions in this class.

Creating a Command with Sub-Commands

So how do you create a new command? Creating a new command is as simple as creating a new class that inherits from the `CCommand()` class and executing the logic of the `Execute()` and `OnRollBack()` member functions. Some commands, such as the create login command, will not implement the `OnRollBack()` function because it uses sub-commands. This means that the `CCommand()` class will handle rolling back all of the sub-commands. Take a look at the `CCreateAccountCmd()` class.

```
<?
require_once( "CCommand.php" );
require_once( "CRenderCmd.php" );
require_once( "CLockTableCmd.php" );
require_once( "CUnlockTableCmd.php" );
require_once( "CVerifyAccountInfoCmd.php" );
require_once( "CCreateNeighborhoodCmd.php" );

class CCreateAccountCmd extends CCommand
{
    function CCreateAccountCmd( &$pParent )
    {
        parent::CCommand( $pParent );

        $this->m_arraySubCommands[0] = new CRenderCmd($this);
        $this->m_arraySubCommands[1] = new CLockTableCmd($this);
        $this->m_arraySubCommands[2] = new CVerifyAccountInfoCmd($this);
        $this->m_arraySubCommands[3] = new CCreateNeighborhoodCmd($this);
        $this->m_arraySubCommands[4] = new CUnlockTableCmd($this);
        $this->m_arraySubCommands[5] = new CRenderCmd($this);

    }

    function Execute( $args )
    {
        $result = null;

        // is it ok for us to continue on to actually creating the account?
        if ($this->m_nCurrentCommand == 0 && isset($args['proceed']))
        {
            // we got our input, so let's move to the next command
            $this->m_nCurrentCommand++;

            // we want the next "proceed" we see to be a new one.
        }
    }
}
```

```
// so get rid of the one we have now
unset( $args['proceed'] );
}

do
{
    // jump to the current command and execute it
    switch ($this->m_nCurrentCommand)
    {
        case 0:
            // render the account creation page
            $result = $this->m_arraySubCommands[0]
                ->Execute( "CreateAccount.html", $args );
            // if it was successful, then we know
            // that we need to wait for input
            if ($result == CMD_FINISHED)
            {
                $result = CMD_NEEDS_INPUT;
            }
            break;
        case 1:
            // lock the database table
            $result = $this->m_arraySubCommands[1]
                ->Execute( $GLOBALS['g_DBTable'] );
            break;
        case 2:
            // verify the account information
            $result = $this->m_arraySubCommands[2]
                ->Execute($args['sPlayerName'],
                           $args['sPassword'],
                           $args['sPasswordRetype'],
                           $args['sNeighborhoodName'] );
            // was there an error?
            if ($result == CMD_ERROR)
            {
                // let the regular error
                // handler deal with these errors
                break;
            }
            // any other error?
            else if ($result != CMD_FINISHED)
            {
```

```
// undo whatever we've done so far
$this->OnRollBack();

// make a note of the error,
// so the create
// account page can render
// it to the user
$args['sError'] = $result;

// restart this command's execution
$this->m_nCurrentCommand = 0;
return $this->Execute( $args );
}

break;
case 3:
// create the neighborhood
$result = $this->m_arraySubCommands[3]
->Execute($args['sPlayerName'],
$args['sPassword'],
$args['sNeighborhoodName'] );
break;
case 4:
// unlock all locked database tables
$result = $this->m_arraySubCommands[4]
->Execute( );
break;
case 5:
// render a "successful creation" page
$result = $this->m_arraySubCommands[5]
->Execute( "AccountCreationSuccess.html", $args );
break;
}

// handle the result from our sub-command
$result = $this->HandleSubCommand($result);

// while there are still commands to be
// executed that don't need human input
} while( $result == CMD_CONTINUED ) ;
```

```

        return $result;
    }
}

?>
```

You'll notice that the first six lines of the code all require other files. These other files are simply other classes that the game uses. Since this is a command, you need to require the CCommand.php file so you can inherit from it. The CRenderCmd() allows you to pass the execute function an html file to render to the browser. This command handles everything that the user sees. The CLockTableCmd() and the CUnlockTableCmd() locks and unlocks a specified table in the Kiddy Cartel database. Without these two classes there would be a chance that a user could be updating the database at the same time that another user is updating the database, and that is not good. The CVerifyAccountInfoCmd() looks in the database to make sure that the user isn't trying to create a duplicate account. If they are, then execution needs to stop and the user needs to know that the account already exists. The last required file is the file that actually creates the default neighborhood for the user.

I am not going to explain each one of these classes. There are just too many classes in this game to explain each in great detail. Please look at the CD and take a look at each class. You'll notice that each command is structured very similarly. I will explain the CCreateAccountCmd() class and another class that doesn't have sub-commands to execute so you can see the difference in how to code them.

All the CCreateAccountCmd() class implements is a constructor and the Execute() function. The constructor tells the CCommand() class that the CCreateAccountCmd() is the parent class. Then the class proceeds to set up an array of sub-commands. Each of these sub-commands will have its own logic in the Execute() function.

The Execute() function uses a do...while loop and a switch statement to determine which command it should execute. Each case in the switch statement corresponds to a sub-command. The sub-command looks like this:

```

$this->m_arraySubCommands[0] = new CRenderCmd($this);
$this->m_arraySubCommands[1] = new CLockTableCmd($this);
$this->m_arraySubCommands[2] = new CVerifyAccountInfoCmd($this);
$this->m_arraySubCommands[3] = new CCreateNeighborhoodCmd($this);
$this->m_arraySubCommands[4] = new CUnlockTableCmd($this);
$this->m_arraySubCommands[5] = new CRenderCmd($this);
```

So that means the first case statement will contain the logic for the CRenderCmd(). The second case statement will contain the logic for the CLockTableCmd(), and so on. Once one command has finished executing, the loop starts over and moves on to the next command by incrementing the \$m_nCurrentCommand variable.

The first case statement uses the CRenderCmd() class. The Execute() function of the CRenderCmd() class takes an HTML page for an argument. This will render the HTML page to the browser. Once the page is rendered to the browser, the game state switches to waiting for input. Once the user fills out the form on the CreateAccount.html page and then clicks Submit, the command executes and locks the table. After the table is locked the loop executes again, but this time it falls to the third case. This is where you verify that this user doesn't exist. If the user does exist, then the class needs to roll back whatever you have done up to this point.

Once you have verified that the account does not exist in the database you create the neighborhood and unlock the table. Once the table is unlocked you render a message to the browser telling the user that the account has been created successfully and that they can log in and start playing. Take a look at Figures 11.2 and 11.3 to see the Create Account page and the Success page.

It's that easy to create a command with sub-commands. Don't worry about how you integrate the command into the game just yet. That will be covered later in this chapter. For now, take a look at how to create a command with no sub-commands.

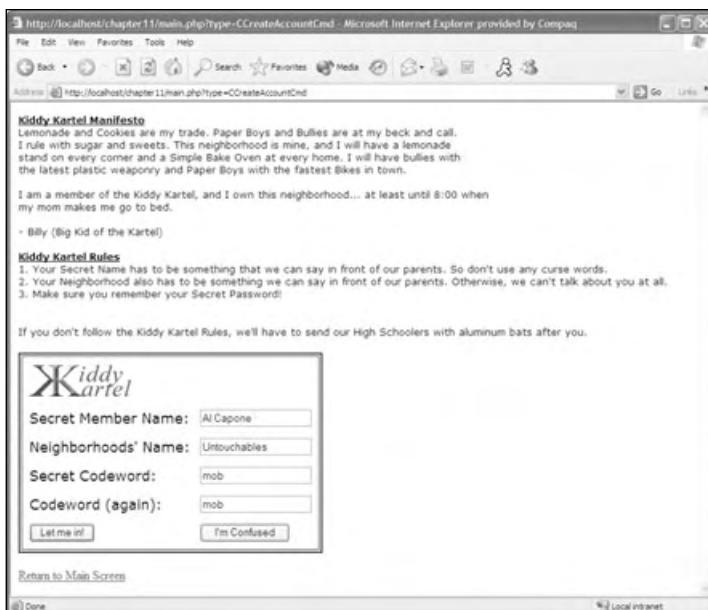


Figure 11.2 The Create Account page.



Figure 11.3 Successful account creation.

Creating a Command without Sub-Commands

To create a command without sub-commands is much simpler than creating commands with sub-commands. Instead of implementing a constructor and the `Execute()` function, you implement the `Execute()` function and the `OnRollBack()` function.

Since you don't have to execute the logic for each sub-command, the only logic that needs to be in the `Execute()` function is the logic for the task you are trying to complete. I'll use the `CAddWeaponsCmd()` class for an example.

```
<?
require_once( "DB.php" );
require_once( "CCommand.php" );

require_once( "settings.php" );

class CAddWeaponsCmd extends CCommand
{
    // information needed to roll back
    var $m_sPlayerName;
    var $m_nWaterBalloons;
```

```
var $m_nPlasticBats;
var $m_nSlingshots;

function Execute( $sPlayerName, $nWaterBalloons,
    $nPlasticBats, $nSlingshots )
{
    // make sure that the input are integer values
    $nWaterBalloons = (int)$nWaterBalloons;
    $nPlasticBats = (int)$nPlasticBats;
    $nSlingshots = (int)$nSlingshots;

    // connect to the db
    $db = DB::connect( $GLOBALS['g_PearDBDSN'] );
    if (DB::isError( $db ))
    {
        // let everyone know that there has been a problem
        $this->OnError("Failed to connect to the
database using " . $GLOBALS['g_PearDBDSN']);

        // return failure
        return CMD_ERROR;
    }

$query = "UPDATE " . $GLOBALS['g_DBTable'] . " SET WaterBalloons
    = WaterBalloons + $nWaterBalloons, PlasticBats = PlasticBats +
    $nPlasticBats, Slingshots = Slingshots + $nSlingshots WHERE
    PlayerName = '$sPlayerName'";
$result = $db->Query( $query );
if (DB::isError( $result ))
{
    // let everyone know there is a problem
    $this->OnError("Failed to update ".$GLOBALS['g_DBTable']);

    // return failure
    return CMD_ERROR;
} else {

    // set up member variables
    $this->m_sPlayerName = $sPlayerName;
    $this->m_nWaterBalloons = $nWaterBalloons;
```

```
$this->m_nPlasticBats = $nPlasticBats;
$this->m_nSlingshots = $nSlingshots;

return CMD_FINISHED;
}

}

function OnRollBack()
{
    // since there is no command to deduct weapons,
    // this rollback must be implemented here
    // connect to the db
    $db = DB::connect( $GLOBALS['g_PearDBDSN'] );
    if (DB::isError( $db ))
    {
        // let everyone know that there has been a problem
        $this->OnError("Failed to connect to the database using "
            . $GLOBALS['g_PearDBDSN']);

        // return failure
        return CMD_ERROR;
    }

$result = $db->Query( "UPDATE ".$GLOBALS['g_DBTable']."'." SET " .
    "WaterBalloons = WaterBalloons + $this->m_nWaterBalloons, " .
    "PlasticBats = PlasticBats + $this->m_nPlasticBats, " .
    "Slingshots = Slingshots + $this->m_nSlingshots " .
    "WHERE PlayerName = '$this->m_sPlayerName'" );
if (DB::isError( $result ))
{
    // there is nothing we can do if the
    // rollback failed, so just let it go
}

return;
}
?>
```

This command is much more straightforward than a class with sub-commands. The Execute() function takes the arguments that you want to update. In this case, since you are adding weapons to your player, it needs to take a player name, the number of water balloons to add, the number of plastic bats to add, and the number of slingshots to add.

The Execute() function starts off by making sure that the variables are integers. Then it tries to connect to the database. If the Execute() function cannot connect to the database then an error has occurred and you need to let the user know what has happened. If the Execute() function connected to the database successfully then you can update the record. The \$db variable is using the PEAR API to connect to the database. The PEAR API contains a member function called Query() that will execute the query you specified. If an error occurs during execution you will need to call the OnRollBack() function. Otherwise you update the players' stats and return control back to the game so it can re-render the page.

The OnRollBack() function attempts to connect to the database. After the OnRollBack() function has successfully connected to the database it attempts to update the database back to the original values that the player had before the query failed. If anything in the OnRollBack() function fails there isn't too much you can do.

Look at All the Commands... Now What?

Now that you know how to create your own commands, you need a way to access them through the game structure. You will need to create something called a *command factory*. The command factory class contains a single function called CreateCommand(). The command factory class also has a require_once() reference to every single command class that you have created. The command factory class will enable you to create a command-based on the query string. When you click on a link that triggers an action and you need to tell the game what type of action you will be starting, you do this through the query string. Take a look at the CommandFactory() class.

```
<?
```

```
/*
```

```
You should require_once() all source files containing  
commands you want instantiated using the factory. The  
CGame object uses the factory to handle creation of the  
command objects in order to hide the need for a parent  
from the CGame object.
```

```
*/
```

```
require_once( "CLockTableCmd.php" );
require_once( "CUnlockTableCmd.php" );

require_once( "CRenderCmd.php" );

require_once( "CManageAccountCmd.php" );
require_once( "CCreateAccountCmd.php" );
require_once( "CEditAccountCmd.php" );
require_once( "CMoveOutOfNeighborhoodCmd.php" );

require_once( "CLogoutCmd.php" );
require_once( "CLoginCmd.php" );

require_once( "CGameMenuCmd.php" );

require_once( "CAssetAcquisitionCmd.php" );
require_once( "CBuyWeaponsCmd.php" );
require_once( "CBuyEquipmentCmd.php" );
require_once( "CRecruitCmd.php" );

require_once( "CBuildResourceCmd.php" );
require_once( "CMowLawnCmd.php" );
require_once( "CMakeLemonadeCmd.php" );
require_once( "CBakeCookiesCmd.php" );

require_once( "CShowNeighborhoodStatsCmd.php" );
require_once( "CViewTopNeighborhoodsCmd.php" );

require_once( "COrderMinionsCmd.php" );
require_once( "COrderPaperBoysCmd.php" );
require_once( "COrderBulliesCmd.php" );

require_once( "CRazeCmd.php" );
require_once( "CScareCmd.php" );
require_once( "CTakeOverCmd.php" );

require_once( "CSpyCmd.php" );
require_once( "CSabotageCmd.php" );
require_once( "CBribeCmd.php" );
```

```

class CCommandFactory
{
    // This exists in order to allow the parent
    // reference in the commands to be set to null.
    var $m_NullVariable = null;

    function CreateCommand( $sCmdName = "" )
    {
        // if it's a known class, create it
        if (class_exists( $sCmdName ))
        {
            return new $sCmdName( $m_NullVariable );
        }
        // are we trying to create a class that doesn't exist?
        else if ( $sCmdName != "" && !class_exists( $sCmdName ) )
        {
            // this should never happen, so warn the programmer
            // (they probably forgot to require_once the command)
            echo "UNKNOWN COMMAND!<BR>";

            // hard exit, since there is nothing else we can really do
            exit(1);
        }
        // it's undefined, so go with the default
        else
        {
            // render the main game command by default
            return new CGameMenuCmd($this->m_NullVariable);
        }
    }
}

?>

```

If you create your own commands and forget to add the `require_once()` reference to the file then the game will let you know that it is trying to execute an unknown command. So remember to add your command to the command factory.

As you can see, all that the command factory does is check to see if the command that you are trying to create exists. If the command does exist then it creates it; if the command does not exist then it lets you know. If there is no command to execute then it assumes that you are on the game menu and that it should render the menu to the browser.

So how in the world do you put all of this together? Well, that is where the `CGame()` class comes into play. Everything that happens in the game occurs through the `main.php` page. The `main.php` page calls the `CGame()` class. Take a look at the `main.php` page.

```
<?

require_once( "CGame.php" );
require_once( "CNeighborhood.php" );
require_once( "CCommand.php" );
require_once( "CRenderCmd.php" );

function main()
{
    // seed the random number generator once for all commands
    srand( time() );

    // get the session up and running
    session_start();

    // if there isn't a game objection in the session, make one
    if ( !isset($_SESSION['game']) )
    {
        $_SESSION['game'] = new CGame;
    }

    // execute the current game command
    $_SESSION['game']->ExecuteCommand( $_REQUEST );

}

main();

?>
```

The first event that occurs when you hit this page is that it checks to see if you currently have a session. If you do have a session, then it executes any requests that have come through the page by using the global `$_REQUEST` variable. However, if you do not have a session it starts a new game by calling the `CGame()` class. Figure 11.4 shows the login screen of the game, which is rendered when a session is started.

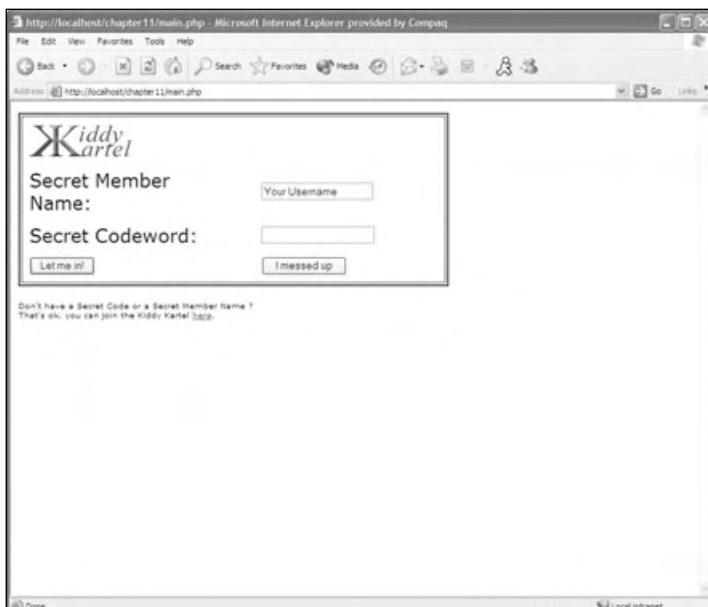


Figure 11.4 The login screen displays when a new game is started.

Once the main.php file constructs the CGame() class, the CGame() class constructs a new command factory.

```
function CGame()
{
    // create the factory that we will be using to get our commands
    $this->m_CmdFactory = new CCommandFactory;
}
```

Once the command factory is loaded, the main.php file calls the ExecuteCommand() function for the CGame() class. This function checks the query string to see if there are any commands posted to it. If there are no commands posted to the \$_REQUEST object then the command type is set to a blank string. This blank string tells the game to render the game menu. Remember that the \$_REQUEST object sends the type of command to the command factory and if there aren't any commands, it assumes that you are on the menu.

After the CGame() class checks to see if the \$_REQUEST object is populated it credits the player with the turns he deserves. Next it checks to see if a command is currently executing. If a command is not executing then it attempts to create a new command.

If a command is executing then the CGame() class checks to see if the executing command is the same as the command that was requested. If it is not the same then something weird

has happened and the executing command needs to be rolled back and the new requested command needs to be created.

Once the command is created, it runs the command. After the command is run, it checks the results that you defined earlier in the `Command()` class. If the result is equal to `CMD_FINISHED` it means that the command has completed successfully. Otherwise it checks to see if an error has occurred. If an error has occurred the command has already rolled back what it has attempted to do, so you just need to let the user know what happened. The following code listing shows the `CGame()` class in all its fantastic glory:

```
<?

require_once( "CCommandFactory.php" );
require_once( "CNeighborhood.php" );
require_once( "CCreditTurns.php" );

class CGame
{

    // the player's neighborhood
    var $m_Neighborhood;

    // the currently executing command
    var $m_CurrentCmd;

    // the command factory from which to get all of our commands
    var $m_CmdFactory;

    function CGame()
    {
        // create the factory that we will be using to
        // get our commands from
        $this->m_CmdFactory = new CCommandFactory;
    }

    // executes the next step in the game
    function ExecuteCommand( $args )
    {
        // if a type is not set, change it to a blank
        if ( !isset($args['type']) )
        {
            $args['type'] = "";
        }
    }
}
```

```
}

// if the player is logged in, credit him the
// turns that he deserves
if ( isset( $_SESSION['neighborhood'] ) )
{
    $turn_update = new CCreditTurns( $null );
    $turn_update->Execute();
}

// is there no currently executing command?
if ($this->m_CurrentCmd == NULL)
{
    // create the new command
    $this->m_CurrentCmd = $this->m_CmdFactory->
        CreateCommand( $args['type'] );
}

// did the command type change on us?
else if ( strtolower(get_class($this->m_CurrentCmd)) != 
    strtolower($args['type']) )
{
    // we interrupted a command, so undo whatever we've done
    $this->m_CurrentCmd->OnRollBack();

    // create the new command
    $this->m_CurrentCmd = $this->m_CmdFactory->
        CreateCommand( $args['type'] );
}

// if there is something to execute (and there should be!), do it
if ($this->m_CurrentCmd != NULL)
{
    // execute the command
    $result = $this->m_CurrentCmd->Execute( $args );

    // is it finished?
    if ($result == CMD_FINISHED)
    {
        $this->m_CurrentCmd = NULL;
    }
}
```

```

        else if ( $result == CMD_ERROR )
        {
            echo "An error occured while executing " .
get_class($this->m_CurrentCmd) . "!<BR>";
        }
    }

}

?>

```

Once you have everything put together in the command factory and all the settings tweaked in the settings.php file, you can run the game. When you log in to the game you will see the menu that looks like Figure 11.5. When you roll over the links you will see a query string. This query string is letting the game know what command to execute. The link in the status bar of your browser should look like this:

<http://localhost/chapter11/main.php?type=CBuyEquipmentCmd>



Figure 11.5 The game in action.

Conclusion

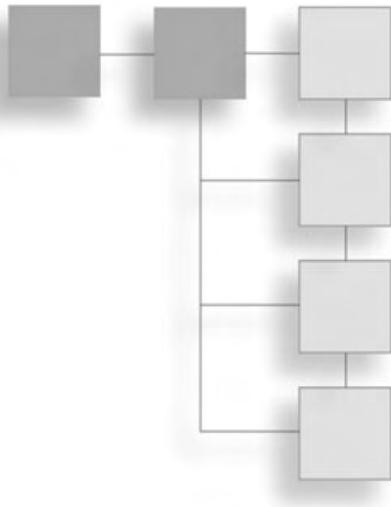
You have successfully created one of the coolest MMO games currently out there. You should give yourself a huge pat on the back because you have come leaps and bounds from the beginning of this book. In the next chapter you will explore how to create your own dynamic Flash pieces with PHP. You will also learn how to add ActionScript and animation to these dynamic Flash pieces.

I have to quickly thank Dragon Fly Game Design (www.dragonflygamedesign.com/) for creating this awesome idea for this MMO. Go check them out; they do some awesome work.

With that said, let's move on to creating some dynamic Flash pieces!

CHAPTER 12

BUILDING YOUR PHP SKILLS



- PHP and Ming
- How to Create a Flash Movie
- Drawing to Your Flash Movie
- Filling with Ming
- Adding Animation to the Flash Piece
- Adding ActionScript to Your Flash Piece

Congratulations, you have made it to the end of this book, and you have accomplished quite a bit. You have learned all about the server environment on which PHP runs. You have installed your own Web server. You have installed the PHP interpreter. You learned all about HTML, and then you conquered the basics of PHP. You have learned about arrays and you've created a tic-tac-toe game. You then dominated non-relational databases and created a basic chess game. After that, you remade a Web-based version of the classic Scorched Earth, called Battle Tank. You even created your own MMO game.

You have done all of this with PHP alone. Imagine what else you can do with PHP. In this final chapter, you'll take a look at some of the other cool things you can do with PHP.

PHP and Ming

What is Ming? Ming is a library that allows you to create your own dynamic SWF Flash movies. That's right; you can create on-the-fly Flash movies. Imagine the endless

applications you could use this for. Dynamic tickers, an online chat application using a PHP engine, creating random challenge games; the list goes on and on. Up to this point, all of your games have been turn-based with minimal user interaction with the page itself.

With PHP combined with Flash you could add some really cool interactions to your game. Besides adding interaction you can create some real animations. You can even add Flash elements into your game.

Now you are probably chomping at the bit to install Ming. Before I get to that I just want to say that this is not a complete “all you can do with Ming” chapter. This is simply a look at what you can do to build your PHP skills. Now, with that said, you install Ming like any other extension. Open up the php.ini file and uncomment the following line:

```
extension=php_ming.dll
```

If you are the type that would rather load the extension dynamically, you can always use the following code to do just that:

```
<?php
if(!extension_loaded('ming'))
{
    if(strtoupper(substr(PHP_OS, 0, 3)) == 'WIN')
    {
        dl('php_ming.dll');
    }
    else
    {
        dl('php_ming.so');
    }
}
?>
```

Now that you have officially enabled Ming you can start creating dynamic Flash pieces. Ming comes with 13 classes that allow you to perform various tasks in Flash. Take a look at Table 12.1 for a description of these 13 classes for creating and manipulating your dynamic Flash pieces.

Note

Ming does not create ActionScript for you. However, it does give you an interface to add your own ActionScript to your dynamic Flash pieces.

Table 12.1 Ming Objects

Class	Description
SWFMovie	Creates a Flash movie.
SWFShape	Creates geometric shapes for your Flash movie.
SWFFill	Provides methods to fill your objects.
SWFGradient	Creates a gradient object so you can use it as a fill for your geometry.
SWFBitmap	Allows you to include .jpg and .png images in your Flash movie.
SWFFont	Allows you to create a font for your Flash piece.
SWFText	Allows you to output text to your Flash piece.
SWFTextField	Allows you to create a text input area in your Flash piece.
SWFDisplayItem	Allows you to access other Flash objects.
SWFSprite	Allows you to create a movie clip for your Flash piece.
SWFMorph	Allows you to create shape tweens between two objects.
SWFButton	Allows you to create a button in your Flash piece.
SWFAction	Creates an ActionScript object so you can add ActionScript to your Flash piece.

How to Create a Flash Movie

To create a new Flash movie you need to invoke an instance of the `SWFMovie` class. Take a look at the following line of code to see how to invoke the `SWFMovie` class:

```
$myMovie = new SWFMovie();
```

The variable `$myMovie` now contains an instance of the `SWFMovie` class. Now that you have an instance of `SWFMovie` you can specify the fundamentals of your movie, such as movie dimensions, frame rate, and background color. After you have specified these fundamentals of your movie you could output it to the browser.

To set the dimensions of your movie, the `SWFMovie` class contains a member function called `setDimension()`. The `setDimension()` function takes two arguments: the width and the height (in pixels) you want your movie to be. Take a look at the following line of code to see how to set the dimension of your dynamic Flash movie:

```
$myMovie->setDimension(400, 300);
```

This sets the `$myMovie` Flash piece to a width of 400 pixels and a height of 300 pixels. Now that you have set the dimensions of your Flash piece you need to specify its frame rate. Your frame rate specifies the number of frames per second that will be displayed. A Flash

movie is just a series of frames, sort of like a cartoon. So to display an animation you would flip through different frames.

To set the frame rate you use the `setRate()` function. This function takes one parameter: an integer that specifies the number of frames per second that will be displayed.

```
$myMovie->setRate(30);
```

The above code snippet sets the movie to flip through its frames at 30 frames per second. Now that you have specified the dimensions of your movie and the rate at which your movie will play you need to set a background color.

To set the background color of your movie, the `SWFMovie` class contains a member function called `setBackground()`. The `setBackground()` function takes three arguments, all integers. The first argument is the red component of the color you wish to use. The second argument is the green component, and the third and final argument is the blue component of the color you wish to use. Take a look at the following code snippet. It sets the background color of your Flash piece to black.

```
$myMovie->setBackground(0, 0, 0);
```

Now that you have created a basic Flash piece, and I mean *basic*, you will want some way to display it to the screen. It's a good thing that the `SWFMovie` class contains a member function called `output()` that writes the output to the browser. To use it, you need to specify the content type just like you did when you were creating your own dynamic graphics. Take a look at the following lines of code:

```
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
```

Let's put all this together and take a look at it to see the results of all this code.

```
<?php
$myMovie = new SWFMovie();
$myMovie->setDimension(400, 300);
$myMovie->setRate(30);
$myMovie->setBackground(0, 0, 0);
```

```
// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Hmmm . . . notice how the whole Flash piece fills the browser no matter how large it is even though you specified the dimensions of the Flash movie? It does this because a Flash piece can scale itself dynamically because it uses vector graphics. So in order to keep the

Flash piece in the dimensions you specified, you need to embed it into an `<object>` tag and save the Flash movie to disk. Take a look at the following example to see how to do that:

```
<html>
  <body>
    <OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
      swflash.cab#version=6,0,0,0"
      WIDTH="400" HEIGHT="300" id="flashTest">
      <PARAM NAME=movie VALUE="example01.php">
      <EMBED src="example01.php" WIDTH="400" HEIGHT="300"
        TYPE="application/x-shockwave-flash"
        PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer"></EMBED>
    </OBJECT>
  </body>
</html>
```

Take a look at Figure 12.1 to see the results of embedding the dynamic Flash piece in an HTML page.

It isn't much yet, but now I'll show you how to draw onto your Flash movie canvas.

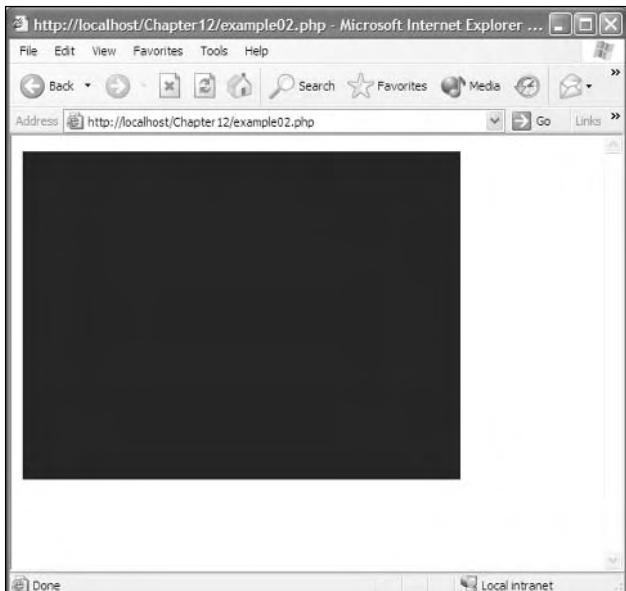


Figure 12.1 A basic dynamic Flash movie embedded in an HTML page.

Drawing to Your Flash Movie

To draw to your Flash piece you need to create an instance of the `SWFShape` class. You do this the same way you created an instance of the `SWFMovie` class. Once an instance of `SWFShape` is created you have access to several functions to draw lines and curves.

```
$shape = new SWFShape();
```

Now that you have an instance of the `SWFShape` class, you can draw to the movie that you created earlier. To draw a line, `SWFShape` gives you two member functions to actually draw the line and one member function to define the style of the line.

To set the style of the line you can use the `setLine()` function. The `setLine()` function takes five arguments. The first is the width of the line in pixels; the next three arguments are the RGB values for the color of the line; the fifth and final argument is the alpha value for the line. Let's say you wanted to draw a 5-pixel wide, yellow line onto your canvas. You would set the style like this:

```
$shape->setLine(5, 255, 255, 0, 255);
```

The two functions to draw a line to the canvas are `drawLine()` and `drawLineTo()`. Each of these functions take two arguments: an x coordinate and a y coordinate. The `drawLine()` function draws a line from the current pen position to a point (x, y) pixels away from the current position. The `drawLineTo()` function draws a line from the current pen position to the (x, y) coordinate specified.

The pen position is where the tip of the pen is currently on the canvas. You can move the position of the pen by calling the `movePenTo()` member function. The `movePenTo()` member function takes two arguments: an x coordinate and a y coordinate. Take a look at the following code example to see the difference between the two draw line functions:

```
<?php  
$myMovie = new SWFMovie();  
$myMovie->setDimension(400, 300);  
$myMovie->setRate(30);  
$myMovie->setBackground(200, 200, 200);  
  
// Draw a line to the canvas using drawLine()  
$line1 = new SWFShape();  
$line1->setLine(5, 0, 0, 0, 255);  
$line1->movePenTo(40, 20);  
$line1->drawLine(100, 100);  
  
$line2 = new SWFShape();  
$line2->setLine(5, 0, 0, 0, 255);
```

```
$line2->movePenTo(80, 20);
$line2->drawLineTo(200, 100);

// Now add the shapes to the movie
$myMovie->add($line1);
$myMovie->add($line2);

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

First you create a movie that is 400×300 pixels with a light gray background. After you have initialized the movie you create two lines, both 5 pixels wide with a color of black. Then you move the pen to a specific point on the canvas and draw a line. After you have drawn a line to both of your SWFShape objects you need to add them to the movie. To do this you use the `add()` member function of the `SWFMovie` class. The `add()` function takes one argument of a mixed type. The results of the above example look like Figure 12.2.

Now that you know how to draw lines to the stage of your Flash piece, take a look at how you draw curves to the stage. Ming provides two functions to draw curves just like it provides you with two functions to draw lines. The first of the two functions is `drawCurve()`. The `drawCurve()` function draws a curve relative to the current pen position. The `drawCurve()` function takes four arguments. The first two arguments are the x and y coordinates of the control point of the curve. The last two arguments are the x and y coordinates of what is called the *anchor point*.

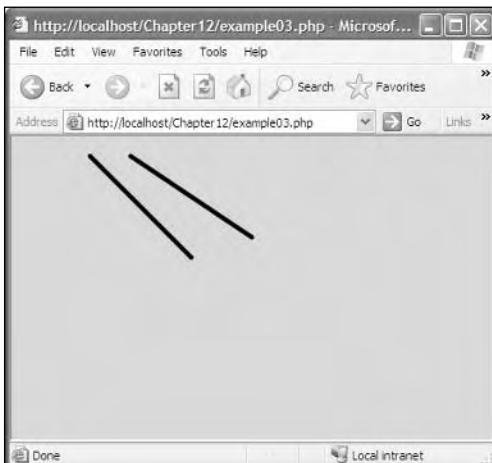


Figure 12.2 Drawing lines with SWFShape.

When Ming draws a curve, it starts at the current position of the pen, which is called the *source point*. It then draws a curve to the anchor point, first passing through the control point. Take a look at Figure 12.3 to see a visual representation of this.

The second function used to draw a curve is called `drawCurveTo()`. Just like the `drawLineTo()` function, it starts at the current pen position and draws its point to the anchor point, using the control point to define the severity of the curve. The `drawCurve()` function also takes four arguments. The first two arguments are the x and y coordinates of the control point. The final two arguments are the x and y coordinates of the anchor point.

To actually draw a curve you follow the same logic you would to draw a line. You need to create a Flash movie, and initialize its dimensions, rate, and background color. Then you need to create a new `SWFShape()` object and set the style of the line you are going to use by using the `setLine()` function. Take a look at the following function to see how to draw a curve using the `drawCurveTo()` function:

```
<?php
$myMovie = new SWFMovie();
$myMovie->setDimension(400, 300);
$myMovie->setRate(30);
$myMovie->setBackground(200, 200, 200);

// Draw a curve
$curve = new SWFShape();
$curve ->setLine(5, 0, 0, 0, 255);
$curve ->movePenTo(40, 20);
$curve ->drawCurveTo(100, 100, 100, 20);

// Now add the shapes to the movie
$myMovie->add($curve);

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Take a look at the results in Figure 12.4

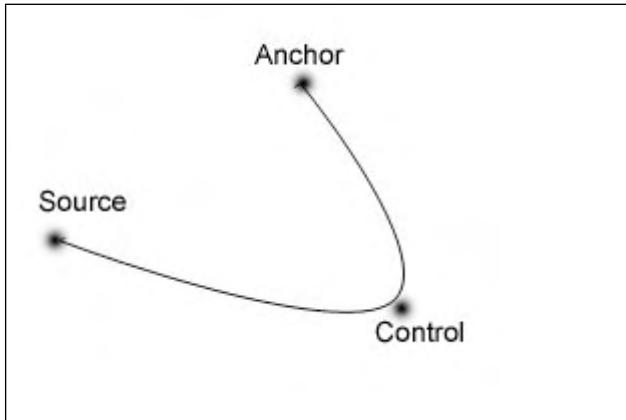


Figure 12.3 The Ming curve drawing method.



Figure 12.4 Results of the DrawCurveTo() function.

Filling Objects with Ming

Ming provides you with three ways to fill an object. You can fill an object with a color, you can fill an object using a gradient, and you can even fill an object by using an image. The `SWFShape()` class provides you with two functions to fill in an image. The `setLeftFill()` function and the `setRightFill()` function both take a `SWFFill()` object as their only argument. You can retrieve an `SWFFill()` object by using the `addFill()` member function that

the SWFShape() class provides. The addFill() function takes three arguments. The first argument is the red component of the color you wish to use to fill. The second argument is the green component of the color, and the third argument is the blue component of the color.

So which fill function do you use? Well, it all depends on how you are drawing your object. If you are drawing your object in a clockwise direction, then you need to use the setRightFill() function. If you are drawing your shape in a counter-clockwise direction, then you need to use the setLeftFill() function.

In the following example you will draw a square with a black 5-pixel border that is filled with green:

```
<?php
$myMovie = new SWFMovie();
$myMovie->setDimension(400, 300);
$myMovie->setRate(30);
$myMovie->setBackground(200, 200, 200);

// Create a new shape and add a fill object
$square = new SWFShape();
$square->setLine(5, 0, 0, 0, 255);
$fill = $square->addFill(0, 255, 0);
$square->setRightFill($fill);

// Draw a square
$square->movePenTo(40, 20);
$square->drawLineTo(140, 20);
$square->drawLineTo(140, 120);
$square->drawLineTo(40, 120);
$square->drawLineTo(40, 20);

// Now add the shapes to the movie
$myMovie->add($square);

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Since I am drawing the square in a clockwise direction I use the setRightFill() function. If I were drawing the square in a counter-clockwise direction I would have used setLeftFill(). Take a look at Figure 12.5 to see the results of the above example.

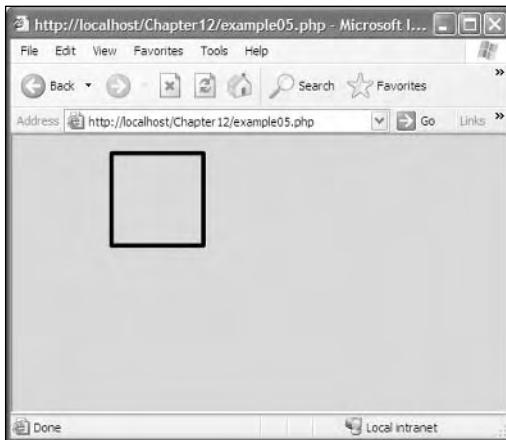


Figure 12.5 Filling a shape.

The second way to fill an object in Flash is to use the `SWFGradient()` class. This class allows you to fill an area with a gradient fill. Once you create an instance of the `SWFGradient()` class you can add a fill entry to it by using the `addEntry()` function.

The `addEntry()` function takes five arguments. The first argument is a ratio value that can be between 0.0 and 1.0. This ratio specifies where in the gradient the color should occur. Usually when you are adding a gradient fill, you will add two entries to the object. The first color will be the 0.0 ratio and the second color will be the 1.0 value for the ratio.

Can you guess what the next three arguments are? That's right, they are the red, green, and blue components of the color. The fifth argument is an optional argument where you can specify the alpha channel for the color.

Now take the previous example used to fill a square and change the regular fill to a gradient fill going from red to white and have the gradient start halfway across the square.

```
<?php  
$myMovie = new SWFMovie();  
$myMovie->setDimension(400, 300);  
$myMovie->setRate(30);  
$myMovie->setBackground(200, 200, 200);  
  
// Create a new shape and set the line style  
$square = new SWFShape();  
$square->setLine(5, 0, 0, 0, 255);
```

```
// Create a new gradient fill
$gradient = new SWFGradient();
$gradient->addEntry(0.0, 255, 0, 0);
$gradient->addEntry(0.5, 255, 255, 255);

// Add the fill to the shape
$fill = $square->addFill($gradient, SWFFILL_RADIAL_GRADIENT);
$square->setRightFill($fill);

// Draw a square
$square->movePenTo(40, 20);
$square->drawLineTo(340, 20);
$square->drawLineTo(340, 220);
$square->drawLineTo(40, 220);
$square->drawLineTo(40, 20);

// Now add the shapes to the movie
$myMovie->add($square);

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Figure 12.6 shows the results of the above example. Pretty cool, huh? But there is one thing I need to mention so as not to lose you. There are two ways to fill a gradient object: the way that you did it with the SWFFILL_RADIAL_GRADIENT flag or you can use the SWFFILL_LINEAR_GRADIENT flag.

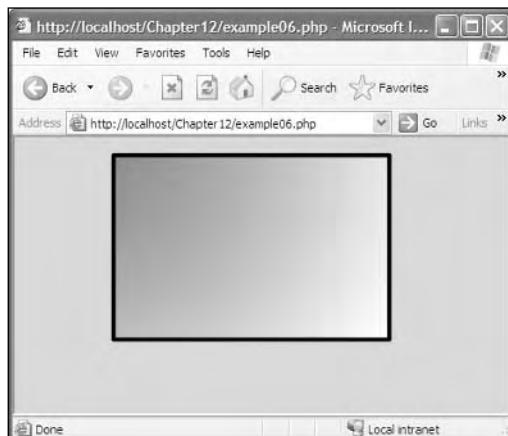


Figure 12.6 Applying a gradient fill to a shape.

Now that you know how to flood-fill objects and how to gradient-fill objects, you can learn how to fill objects with bitmaps. This is a fun way to fill objects. You can sort of compare it to texture mapping in 3D game programming, but it's just not as complex.

To fill an object with bitmaps you need to create an instance of the `SWFBitmap()` class. The `SWFBitmap()` class doesn't expose any fill functions but it does allow you to create a bitmap. The `addFill()` member function of the `SWFShape()` class is what allows you to fill a shape.

The `SWFBitmap()` class allows you to create a bitmap from two types of files. The first is a `.dbl` (Define Bits Lossless) file which can be created from a `.gif` or a `.png` file. Ming provides a tool called `gif2dbl` that converts a `.gif` to a `.dbl` file. The second graphic type from which you can create a bitmap is a non-progressive `.jpg`. You can use the evaluation edition of Paint Shop Pro included on the CD to create non-progressive `.jpg` images.

To create a bitmap from another file, you need to read in the data into a buffer using the file functions that PHP provides. To read in a file you will need to open the file with the `fopen()` function and read the file with the `fread()` function. Take a look at the following code snippet for a quick example:

```
$fp = fopen("somefile.jpg", "rb");
$data = fread($fp, filesize("somefile.jpg"));
```

Note

For those of you accustomed to opening and reading files in C, PHP uses the same parameters as C.

After you have read the data into a buffer, you pass the buffer into the `SWFBitmap()` constructor to create a new bitmap image.

```
$bitmap = SWFBitmap($data);
```

Then, after you have a bitmap image, you can pass it to the `addFill()` member function of the `SWFShape()` class and specify one of the following flags:

- `SWFFILL_CLIPPED_BITMAP`. This flag will display just one instance of the bitmap.
- `SWFFILL_TILED_BITMAP`. This flag will tile the bitmap across your image.

Take a look at the following example to see how to tile a bitmap across a `SWFShape()`:

```
<?php
$myMovie = new SWFMovie();
$myMovie->setDimension(400, 300);
$myMovie->setRate(30);
$myMovie->setBackground(200, 200, 200);

// Create a new shape and set the line style
```

```
$square = new SWFShape();
$square->setLine(5, 0, 0, 0, 255);

// Open a graphic and read the data into a buffer
$fp = fopen("tessellation.jpg", "rb");
$data = fread($fp, filesize("tessellation.jpg"));

// Create a new bitmap image
$bitmap = new SWFBitmap($data);

// Add the fill to the shape
$fill = $square->addFill($bitmap, SWFFILL_TILED_BITMAP);
$square->setRightFill($fill);

// Draw a square
$square->movePenTo(40, 20);
$square->drawLineTo(340, 20);
$square->drawLineTo(340, 220);
$square->drawLineTo(40, 220);
$square->drawLineTo(40, 20);

// Now add the shapes to the movie
$myMovie->add($square);

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Figure 12.7 shows the results of the above example.



Figure 12.7 Filling a shape with a tiled bitmap image.

Adding Animation to Your Flash Movie

Adding animation to your Flash movie really isn't that difficult. Up to this point you have used the `add()` member function of the `SWFMovie()` class assuming that it does not return anything. But as a matter of fact it does return something. It returns a `SWFDisplayItem()` object. You can use this returned object to manipulate the shape that you add to your Flash movie.

```
// How you have been using the add() function  
$movie->add($shape);  
// The way you will be using the add() function  
$shapeHandle = $movie->add($shape);
```

With this new `SWFDisplayItem()` handle you can move the object using one of two functions. The first is the `move()` function. It takes two arguments. The first argument is the x coordinate and the second argument is the y coordinate. The second function is the `moveTo()` function, and it also takes an x coordinate and a y coordinate.

Not only can you move a shape with the `move()` or `moveTo()` functions, you can also rotate a shape by using the `rotate()` function. The `rotate()` function takes a single argument specifying the number of degrees you wish to rotate the object.

Let's take the square example again and draw the square in the upper left-hand corner of the Flash piece. Then move it to a point and rotate the square by 45 degrees.

```
<?php  
$myMovie = new SWFMovie();  
$myMovie->setDimension(400, 300);  
$myMovie->setRate(30);  
$myMovie->setBackground(200, 200, 200);  
  
// Create a new shape and set the line style  
$square = new SWFShape();  
$square->setLine(5, 0, 0, 0, 255);  
  
// Draw a square  
$square->movePenTo(1, 1);  
$square->drawLineTo(61, 1);  
$square->drawLineTo(61, 61);  
$square->drawLineTo(1, 61);  
$square->drawLineTo(1, 1);  
  
// Now add the shapes to the movie  
$squareHandle = $myMovie->add($square);
```

```
// Move the shape a bit  
$squareHandle->moveTo(30, 100);  
  
// Rotate the shape 45 degrees  
$squareHandle->rotate(45);  
  
// Now output the movie  
header("Content-type:application/x-shockwave-flash");  
$myMovie->output();  
?>
```

The above example, with results shown in Figure 12.8, simply moves the object down and to the right a bit, and then it rotates the object 45 degrees. This only affects the first frame of the Flash piece. In order for this to become an animation you need to add multiple frames to your Flash piece. To do this you can use the `nextFrame()` member function of the `SWFMovie()` class.

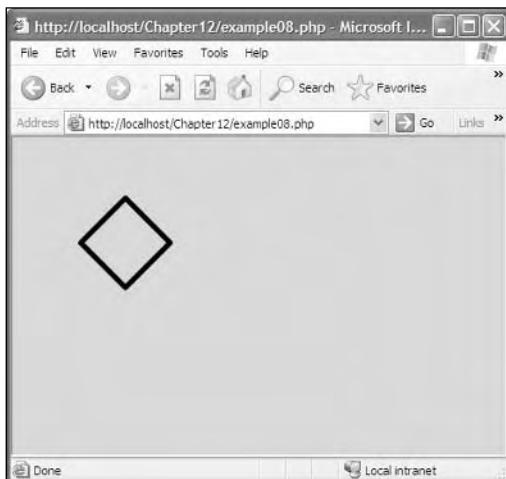


Figure 12.8 Moving and rotating a shape.

Now take the example you just did and make it an animation. The goal of this animation will be to simply move the square some number of pixels and rotate it each frame. I won't be able to show you a screen shot of the animation because it wouldn't make any sense, but the example is on the CD provided with this book.

```
<?php  
$myMovie = new SWFMovie();  
$myMovie->setDimension(400, 300);  
$myMovie->setRate(30);
```

```
$myMovie->setBackground(200, 200, 200);

// Create a new shape and set the line style
$square = new SWFShape();
$square->setLine(5, 0, 0, 0, 255);

// Draw a square
$square->movePenTo(1, 1);
$square->drawLineTo(61, 1);
$square->drawLineTo(61, 61);
$square->drawLineTo(1, 61);
$square->drawLineTo(1, 1);

// Now add the shapes to the movie
$squareHandle = $myMovie->add($square);

$squareHandle->moveTo(30, 100);
$myMovie->nextFrame();
$squareHandle->rotate(15);
$myMovie->nextFrame();
$squareHandle->moveTo(80, 200);
$myMovie->nextFrame();
$squareHandle->rotate(15);
$myMovie->nextFrame();
$squareHandle->moveTo(130, 280);
$myMovie->nextFrame();
$squareHandle->rotate(15);
$myMovie->nextFrame();
$squareHandle->moveTo(180, 180);
$myMovie->nextFrame();
$squareHandle->rotate(15);
$myMovie->nextFrame();
$squareHandle->moveTo(130, 80);
$myMovie->nextFrame();
$squareHandle->rotate(15);
$myMovie->nextFrame();

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

If you take a look at the example on the CD you'll notice that the square bounces around the screen. If you think it is going too fast you should adjust the frame rate on your Flash piece or add more frames to the rotation. This is a simple example of the animation but it should give you the idea.

Now take a look at the `SWFMorph()`, which will allow you to create a shape tween between two shapes. The `SWFMorph()` class has two member functions: `getShape1()` and `getShape2()`. These two functions don't take any arguments at all. They allow you to set a start shape and an end shape to morph between.

```
$morph = new SWFMorph();
$shape1 = $morph->getShape1();
$shape2 = $morph->getShape2();
```

Let's morph the square into a rectangle to see how to use the `SWFMorph()` class.

```
<?php
$myMovie = new SWFMovie();
$myMovie->setDimension(400, 300);
$myMovie->setRate(5);
$myMovie->setBackground(200, 200, 200);

// Create a new morph object
$morph = new SWFMorph();

// Create a new shape and set the line style
$square = new SWFShape();
$rectangle = new SWFShape();

$square = $morph->getShape1();
$square->setLine(5, 0, 0, 0, 255);

$rectangle = $morph->getShape2();
$rectangle->setLine(5, 0, 0, 0, 255);

// Draw a square
$square->movePenTo(1, 1);
$square->drawLineTo(61, 1);
$square->drawLineTo(61, 61);
$square->drawLineTo(1, 61);
$square->drawLineTo(1, 1);
```

```
// Draw a rectangle
$rectangle->movePenTo(1, 1);
$rectangle->drawLineTo(161, 1);
$rectangle->drawLineTo(161, 61);
$rectangle->drawLineTo(1, 61);
$rectangle->drawLineTo(1, 1);

$morphHandle = $myMovie->add($morph);
$morphHandle->moveTo(100, 100);

for($i = 0.0; $i < 1.0; $i += 0.1)
{
    $morphHandle->setRatio($i);
    $myMovie->nextFrame();
}

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

First you set the attributes of your movie. (You'll notice in this example that the frame rate is only 5 frames per second. The reason for this is so you can see the morph more clearly.) Then you create a morph object. Once you have created a morph object you need to create two shape objects and set which one will be the starting shape and which one will be the ending shape.

After you have set the start and end shapes you can draw them. Then you need to add the morph object to the movie and get the new object handle. Once you have the object handle you can actually create the tween. In this case the tween occurs over 11 frames. It starts at 0.0 and adds 0.1 to \$i. Each time \$i is incremented you update the ratio at which the tween should occur and then add a frame to the movie. Take a look at the example included on the CD to see the results of the above example.

Adding ActionScript to Your Flash Piece

ActionScript is an object-oriented programming language used specifically to create interactivity and animations in Flash. With ActionScript you can do everything from loading in dynamic sprites to dynamically moving and resizing movie clips.

Note

ActionScript's syntax is a lot like JavaScript. It uses a DOM to access its elements—e.g., `_root.clientinfo.ip`.

ActionScript uses data types to define its variables. It used to be in Flash 4 you did not need to declare the data type of your variable. But since the release of Flash 5 you need to specify a data type. There are five basic data types that Flash uses:

- Boolean
- String
- Number
- Movie Clip
- Object

These data types are used in many languages and should be nothing new to you by now.

So how do you add ActionScript to your dynamic Flash pieces? You use the `SWFAction()` class. The constructor for this class takes a single argument. That argument is an ActionScript command. For example:

```
$action = new SWFAction("stop();");
```

The easiest way to see how this all works is to see an example, so take a look at the following example, in which a text box will display the position of a shape on the stage:

```
<?php  
$myMovie = new SWFMovie();  
$myMovie->setDimension(400, 300);  
$myMovie->setRate(5);  
$myMovie->setBackground(200, 200, 200);  
  
// Create a new morph object  
$morph = new SWFMorph();  
  
// Create a new shape and set the line style  
$square = new SWFShape();  
$square->setLine(5, 0, 0, 0, 255);  
  
// Draw the square  
$square->movePenTo(1, 1);  
$square->drawLineTo(61, 1);  
$square->drawLineTo(61, 61);
```

```
$square->drawLineTo(1, 61);
$square->drawLineTo(1, 1);

// Add the shape to a movie clip so you can access some properties later
$shape = new SWFSprite();
$shape->add($square);
$shape->nextFrame();

// Create a text box to display some text
$textbox = new SWFTextField(SWFTEXTFIELD_DRAWBOX);
$textbox->setBounds(100, 100);
$textbox->setFont(new SWFFont("_sans"));
$textbox->setColor(255, 0, 0);

// Now give the text box a name so you can reference it
$textbox->setName("myTextBox");

// To move the text box you have to add it to a sprite
$container = new SWFSprite();
$container->add($textbox);

// Move to the next frame in the sprite so it will show up
$container->nextFrame();

// Add the square to the movie clip
$shapeHandle = $myMovie->add($shape);
$shapeHandle->moveTo(10, 10);
$shapeHandle->setName("shape");

$containerHandle = $myMovie->add($container);
$containerHandle->moveTo(200, 100);
$containerHandle->setName("container");

$myMovie->nextFrame();
$myMovie->add(new SWFAction("_root.container.myTextBox = _root.shape._x +
', ' + _root.shape._y;"));

// Now output the movie
header("Content-type:application/x-shockwave-flash");
$myMovie->output();
?>
```

Let's break down the above example. First you create your master Flash movie and set its properties just like you have been doing. Then you create a new shape, in this case a square, and you draw it. Next you do something a little different. You add the shape to its own movie sprite or movie clip. This will allow several things but most importantly it will allow you to access some predefined properties that a movie contains. The next step is very important; you must move the sprite to the next frame in order for anything to show up in your master movie.

After you have created the movie clip that contains the shape you want to access, you create a new text box. This will allow you to enter text, although in the example you are using it to display text. After the text box is created, you add it to its own movie clip. Remember to move the movie clip to the next frame.

Caution

Make sure you move sprites to the next frame. Otherwise you will never see the results on the Flash piece that is sent to the browser.

Now you essentially have three movies: the master movie, a movie clip containing your shape, and a movie clip containing your text box. To have a complete Flash movie you need to combine these into one piece. So you add the shape movie clip to your master movie and then you set a name. Setting the name of the movie clip will allow you to access it with ActionScript. Now you add the second movie clip to the master movie and also set the name of the movie clip.

Now you have one movie with two movie clips in it. These movie clips and their elements can now be accessed and manipulated through ActionScript because you set the names of the clips. To access the clips, you start at the root of the movie, which is called _root. Then you go down one more level to the movie clip. Once there, you can manipulate the movie clip, get its properties, or access elements that are nested further into the clip.

Remember that an ActionScript requires an action to trigger it, so you need to add a frame to the master movie and then add the script. Once the movie moves to the next frame it will execute the script on the frame. In this case, it will put the coordinates of the movie clip into the text box.

Take a look at Figure 12.9 to see the results of the above example.

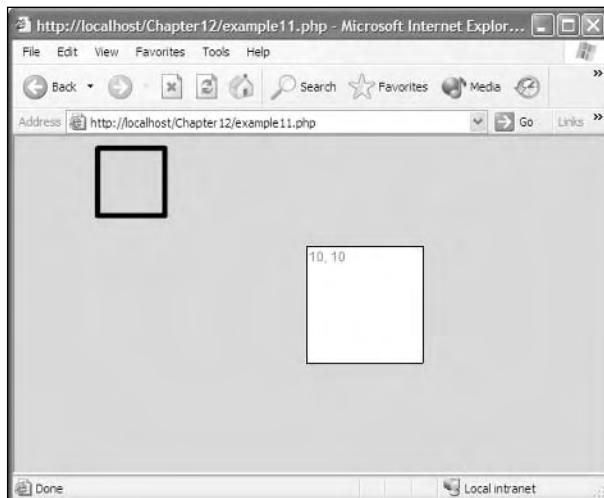


Figure 12.9 An example of ActionScript.

Conclusion

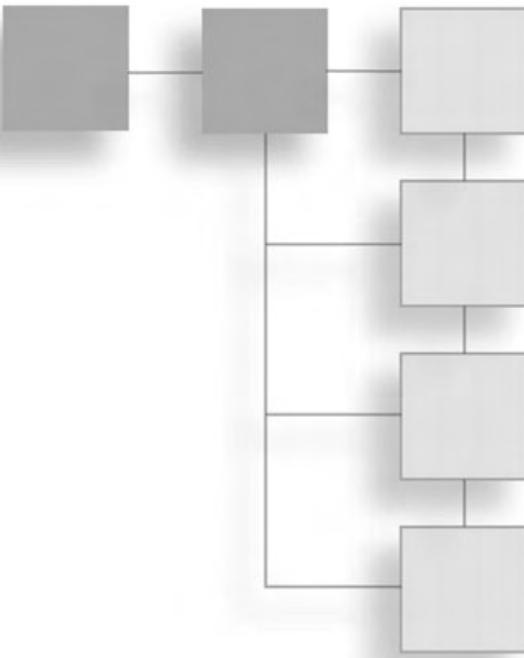
This was a quick overview of some of the more advanced uses of PHP. You can go much more in depth than I have here. I suggest that you use all the resources available to you to check out the extensive functionality that PHP offers you.

PHP is a great language that has many uses. Take a look at the following list to give you just a few ideas:

- Create your own browsers.
- Create parsing and management backends for your games.
- Create extensible, fast, reliable applications and systems.
- Manage and manipulate graphics.
- Create your dynamic Flash pieces all with code.
- Create sockets to Flash pieces to create a multiplayer experience.
- Quickly create chat systems for your games or Web sites.

These are just a few of the applications for which you can use PHP. Let your mind wander and I am sure you will come up with some great and fantastic ideas for applying all of your newfound knowledge.

With the Web your options are endless. Have fun with it. Now go create the next generation of online games!



PART V

APPENDICES

APPENDIX A

HTML Language Reference	283
-------------------------------	-----

APPENDIX B

PHP Language Reference	309
------------------------------	-----

APPENDIX C

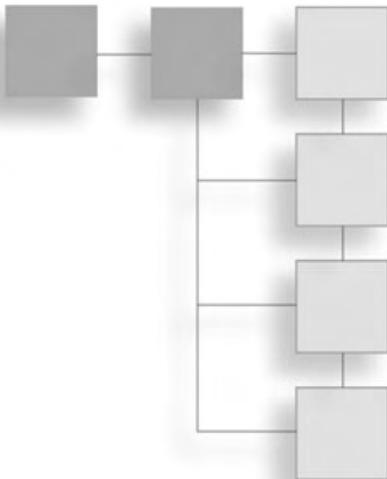
Support—Debugging Applications	329
--------------------------------------	-----

APPENDIX D

GD SDK Language Reference	341
---------------------------------	-----

This page intentionally left blank

APPENDIX A



HTML LANGUAGE REFERENCE

This chapter is simply your guide to all the HTML functions available to you. Each element is listed followed by a description of what the element does, and, if there are any, a list of all the attributes the element can use.

<!-- and <--> Comment

Denotes a comment in HTML.

!DOCTYPE

Declares the type of content and format of the document.

A

Defines a hypertext link. The `HREF` or `NAME` attribute must be specified.

A Attributes

Attribute	Description
ACCESSKEY	Sets the accessibility key for this element
CHARSET	Sets the character set used to encode this element
CLASS	Sets the style sheet class this element should use
COORDS	Sets the coordinates for this element
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding for this element
DIR	Sets the reading direction for this element
HREF	Sets the destination of this element
HREFLANG	Sets the language code of the resource pointed to by <code>HREF</code>
ID	Sets the ID of this element
LANG	Sets the language of this element

Attribute	Description
LANGUAGE	Sets the scripting language this element should use
METHODS	Sets the HTTP methods supported by this element
NAME	Sets the name of this element
REL	Sets the relationship between this element and the destination element
SHAPE	Sets the shape of the element
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tabbing order for this element
TARGET	Sets the destination to where the content should go
TITLE	Sets a Tool Tip for this element
TYPE	Sets the MIME type of this element
URN	Sets a Uniform Resource Name for this element

ADDRESS

Specifies information such as an address, a signature, and ownership.

ADDRESS Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading direction for this element
ID	Sets the ID for this element
LANG	Sets the language for this element
LANGUAGE	Sets the scripting language this element should use
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element

APPLET

Places a Java Applet into the page.

APPLET Attributes

Attribute	Description
ALIGN	Sets the alignment of the element
ALT	Sets the alternative text for this element
ARCHIVE	Sets a character string that can be used for your own archive functionality
BORDER	Sets the amount of border this element should have
CLASS	Sets the style sheet class this element should use
CODE	Sets the URL pointing to the compiled Java Class
CODEBASE	Sets the URL of the component
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding
HEIGHT	Sets the height of the element
HSPACE	Sets the horizontal margin for the element

Attribute	Description
ID	Sets the ID for this element
NAME	Sets the name for this element
SRC	Sets a URL to be loaded by this element
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element
VSPACE	Sets the vertical margin for this element
WIDTH	Sets the width of the element

AREA

Specifies the shape of a hot spot for a client-side image map.

AREA Attributes

Attribute	Description
ALT	Sets the alternate text for this element
CLASS	Sets the style sheet class this element should use
COORDS	Sets the coordinates for this hot spot
DIR	Sets the reading order of this element
HREF	Sets a destination URL for this element
ID	Sets the ID of the element
LANG	Sets the language of this element
LANGUAGE	Sets the client-side scripting language this element should use
NAME	Sets the name of this element
NOHREF	Sets whether a click in the region should cause an action
SHAPE	Sets the shape of this element
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tab order for this item
TARGET	Sets the destination of the content
TITLE	Sets a Tool Tip for this element

B

Renders the specified text in boldface type.

B Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading order of this element
ID	Sets the ID of the element
LANG	Sets the language of the element
LANGUAGE	Sets the client-side scripting language this element should use
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element

BASE

Specifies the document's base URL.

BASE Attributes

Attribute	Description
HREF	Sets the baseline URL on which all the links in the page will be based
TARGET	Sets the destination of the content

BASEFONT

Sets the base font that the page will use for rendering text.

BASEFONT Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
COLOR	Sets the color the font should be rendered in
FACE	Sets the font the element should use
ID	Sets the ID of the element
LANG	Sets the language of the element
LANGUAGE	Sets the client-side scripting language this element should use
SIZE	Specifies the size of the font

BDO

Turns off the bidirectional rendering algorithm for the specified fragment of text.

BDO Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading order of this element
ID	Sets the ID of this element
LANG	Sets the language of this element
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element

BGSOUND

Specifies a background sound to be played while the page is loaded.

BGSOUND Attributes

Attribute	Description
BALANCE	Sets how the background sound should be distributed between the left and the right speakers
ID	Sets the ID of the element

Attribute	Description
LOOP	Sets the number of times the sound should loop when played
SRC	Sets the URL of a sound to be played
VOLUME	Sets the volume for the sound

BIG

Renders text in a larger font than the current font specified.

BIG Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading order of this element
ID	Sets the ID of the element
LANGUAGE	Sets the client-side scripting language this element should use
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element

BLINK

Causes the specified text to blink on and off in the page.

BLINK Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

BLOCKQUOTE

Tabs a specified set of text in, and denotes a quotation in the text.

BLOCKQUOTE Attributes

Attribute	Description
CITE	Sets reference information about the text
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading order of the element
ID	Sets the ID of the element
LANG	Sets the language of the element
LANGUAGE	Sets the client-side scripting language this element should use
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element

BODY

Defines the beginning of the body of the HTML document.

BODY Attributes

Attribute	Description
ALINK	Sets the color of the active state of a link
BACKGROUND	Sets the background image of the body
BGCOLOR	Sets the background color of the body of the page
BOTTOMMARGIN	Sets the amount of margin on the bottom of the page
CLASS	Sets the style sheet class this element should use
DIR	Sets the reading order of the element
ID	Sets the ID of the element
LANG	Sets the language of the element
LANGUAGE	Sets the client-side scripting language this element should use
LEFTMARGIN	Sets the amount of margin on the left of the page
LINK	Sets the color of links in the page
RIGHTMARGIN	Sets the amount of margin on the right of the page
SCROLL	Sets whether the scroll bars are turned on or off
STYLE	Sets style-sheet-specific information for this element
TEXT	Sets the color that body text should be
TOPMARGIN	Sets the amount of margin on the top of the document
VLINK	Sets the color of visited links

BR

Inserts a line break in the page.

BR Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
CLEAR	Sets the side where the next line of content will appear after the line break, relative to a floating object
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

BUTTON

Renders an HTML button with the specified text.

BUTTON Attributes

Attribute	Description
ACCESSKEY	Sets the accessibility key for this element
CLASS	Sets the style sheet class this element should use

Attribute	Description
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding
DIR	Sets the reading order of this element
DISABLED	Sets the status of the element
ID	Sets the ID of the element
LANG	Sets the language for this element
LANGUAGE	Sets the client-side scripting language that this element should use
NAME	Sets the name of the element
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tab order of this element
TITLE	Sets a Tool Tip for this element
TYPE	Specifies the type of button this element is
VALUE	Sets the default selected value of this element

CAPTION

Specifies a caption to be placed next to a table.

CAPTION Attributes

Attribute	Description
ALIGN	Sets the position of the element relative to the table
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

CITE

Renders text in italics.

CITE Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

CODE

Renders text as a code sample using a fixed-width font.

CODE Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

COL

Specifies column-based defaults for a table.

COL Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
SPAN	Sets the number of columns in the group
STYLE	Sets style-sheet-specific information for this element
VALIGN	Sets the vertical alignment of the element
WIDTH	Sets the width of the element

COLGROUP

Sets a container for a group of columns.

COLGROUP Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
SPAN	Sets the number of columns in the group
STYLE	Sets style-sheet-specific information for this element
VALIGN	Sets the vertical alignment of the element
WIDTH	Sets the default width for each column in the group

DD

This specifies the definition of an item in a definition list.

DD Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

DFN

Defines an instance of a term.

DFN Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

DIR

Renders text so it appears like a directory listing.

DIR Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
COMPACT	Sets a Boolean value specifying whether the list should be compacted by removing white space between the items
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

DIV

Defines a container section or a new layer within a page.

DIV Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or text
DATASRC	Sets the source of the data for binding for this element
DIR	Sets the reading direction for this element
ID	Sets the ID of the element
LANG	Sets the language of the element
LANGUAGE	Sets the client-side scripting language this element should use
NOWRAP	Sets the browser to not automatically wrap text
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tab order for this element
TITLE	Sets a Tool Tip for this element

DL

Starts a definition list.

DL Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
COMPACT	Sets a Boolean value specifying whether the list should be compacted by removing white space between the items
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

DT

Starts a definition term in a definition list.

DT Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

EM

Renders text as emphasized; basically just bolds the text.

EM Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

EMBED

Embeds a document of any type into the page.

EMBED Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
ALT	Sets the alternate text for the element
BORDER	Sets the amount of border the element will have
CLASS	Sets the style sheet class this element should use
HEIGHT	Sets the height of the element

Attribute	Description
ID	Sets the ID of the element
NAME	Sets the name of the element
PLUGINSPAGE	Retrieves a URL of the plug-in to be used to view the embedded document
SRC	Sets a URL to be loaded
STYLE	Sets style-sheet-specific information for this element
TITLE	Sets a Tool Tip for this element
WIDTH	Sets the width of the element

FIELDSET

Draws a box around the contained elements.

FIELDSET Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

FONT

Allows you to specify a font face, size, and color for a section of text.

FONT Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
COLOR	Sets the color of the text for this element
FACE	Sets the font for this element
ID	Sets the ID for the element
SIZE	Sets the size of the font for this element
STYLE	Sets style-sheet-specific information for this element

FORM

Creates a form that can contain controls and elements whose values are sent to the server.

FORM Attributes

Attribute	Description
ACTION	Specifies the URL for processing
CLASS	Sets the style sheet class this element should use
ENCTYPE	Sets the encoding type of the form
ID	Sets the ID for the element

Attribute	Description
METHOD	Sets how the form will transfer its data to the server
NAME	Sets the name of the element
STYLE	Sets style-sheet-specific information for this element

FRAME

Specifies a individual frame within a frameset.

FRAME Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BORDERCOLOR	Sets the border color of the element
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding for this element
FRAMEBORDER	Sets whether or not the element will have a border
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
MARGINHEIGHT	Sets the height of the margin
MARGINWIDTH	Sets the width of the margin
NAME	Sets the name of the element
NORESIZE	Sets the frame at a fixed width so the user can not resize the frame
SCROLLING	Sets whether the frame will allow scrolling or not
SRC	Sets a URL to be loaded by the element
STYLE	Sets style-sheet-specific information for this element

FRAMESET

Specifies a frameset that can contain multiple frames and other nested framesets.

FRAMESET Attributes

Attribute	Description
BORDER	Sets the amount of border for the frameset to have
BORDERCOLOR	Sets the border color for this element
CLASS	Sets the style sheet class this element should use
COLS	Sets the widths of the columns in the frameset
FRAMEBORDER	Sets whether or not to display a border
FRAMESPACING	Sets the amount of space between frames
ID	Sets the ID of the element
ROWS	Sets the height of the rows in the frameset
STYLE	Sets style-sheet-specific information for this element

HEAD

Begins the heading of an HTML document, and contains tags holding information about the document.

HEAD Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
PROFILE	Sets one or more Uniform Resource Identifiers of meta-data profiles
STYLE	Sets style-sheet-specific information for this element

HN

The six headers (H1 -H6) that can render text in a range of sizes.

HN Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

HR

Places a horizontal rule on the page.

HR Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
COLOR	Sets the color of the element
ID	Sets the ID of the element
NOSHADE	Sets the element to render without 3-D shading
SIZE	Sets the size of the element
STYLE	Sets style-sheet-specific information for this element

HTML

Starts the HTML document itself.

I

Renders a range of text in italics.

I Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
STYLE	Sets style-sheet-specific information for this element

IFRAME

Creates an in-line floating frame in the page.

IFRAME Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BORDER	Sets the amount of border for the element
BORDERCOLOR	Sets the color of the border for the element
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding for this element
FRAMEBORDER	Sets whether or not the element will have a border
HEIGHT	Sets the height of the element
HSPACE	Sets the horizontal margin for the element
ID	Sets the ID for this element
MARGINHEIGHT	Sets the height of the margin
MARGINWIDTH	Sets the width of the margin
NAME	Sets the name of the element
NORESIZE	Sets the frame at a fixed width so the user can not resize the frame
SCROLLING	Sets whether the frame will allow scrolling or not
SRC	Sets a URL to be loaded by the element
STYLE	Sets style-sheet-specific information for this element
VSPACE	Sets the vertical margin for the element
WIDTH	Sets the width of the element

IMG

Embeds an image in the page.

IMG Attributes

Attribute	Description
ACCESSKEY	Sets or retrieves the accessibility key
ALIGN	Sets the horizontal alignment of the element
ALT	Sets the alternate text for the element
BORDER	Sets the amount of border for the element
ID	Sets the ID of the element
HEIGHT	Sets the height of the element
NAME	Sets the name of the element
SRC	Sets the URL to be loaded by the element
USEMAP	Sets what image map to use
WIDTH	Sets the width of the element

INPUT

Specifies an input control for a form. The input control could be a button, check box, radio button, text box, or password box.

INPUT Attributes

Attribute	Description
ACCEPT	Sets a set of comma-separated content types that will be accepted by the element
ACCESSKEY	Sets or retrieves the accessibility key
ALIGN	Sets the horizontal alignment of the element
ALT	Sets the alternate text for this element
CHECKED	Sets whether the field is checked
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or text
DATASRC	Sets the source of the data for binding for this element
HSPACE	Sets the horizontal margin of the element
ID	Sets the ID of the element
maxlength	Sets the maximum number of characters accepted for this element
NAME	Sets the name of the element
readonly	Specifies that the element is read only
SIZE	Sets the size of the element
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tabbing order for this element
TYPE	Sets the type of input this element will be
VALUE	Sets the initial value for the element

KBD

Renders text in a fixed width font, sort of like a typewriter.

KBD Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

LABEL

Defines a label for a control element

LABEL Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or text
DATASRC	Sets the source of the data for binding for this element
FOR	Sets the element for which the label is to be assigned
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

LI

Specifies an item in an ordered or unordered list.

LI Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID for the element
STYLE	Sets style-sheet-specific information for this element

LINK

Specifies a hyperlink to another document. This is used in the heading of the HTML document.

LINK Attributes

Attribute	Description
CLASS	Sets the style sheet class that this element should use
Href	Specifies the base URL to go to
ID	Sets the ID for the element
STYLE	Sets style-sheet-specific information for this element

MAP

Defines the collection of hot spots for an image map.

MAP Attributes

Attribute	Description
CLASS	Sets the style sheet class that this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element
NAME	Sets the name of the image map defined by the map element

MARQUEE

Creates a scrolling marquee on the page. This marquee can scroll vertically or horizontally.

MARQUEE Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of this element
BEHAVIOR	Sets the scrolling behavior of the element
BGCOLOR	Sets the background color of the element
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or Text
DATASRC	Sets the source of the data for binding for this element
DIRECTION	Sets the direction the text should scroll
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
LOOP	Sets the number of times the element should play
SCROLLAMOUNT	Sets the amount of characters the element should scroll by
SCROLDELAY	Sets the amount of delay before scrolling starts
STYLE	Sets style-sheet-specific information for this element
WIDTH	Sets the width of the element

META

Provides various types of information to search engines and other interpreters. This element goes in the heading of the HTML document.

META Attributes

Attribute	Description
CONTENT	Sets the meta content
HTTP-EQUIV	Sets information to be used to bind the content to the HTTP headers
NAME	Sets the name of the value specified in the content

NOFRAMES

Specifies the section of text to display for browsers that do not support framesets.

NOFRAMES Attributes

Attribute	Description
CLASS	Sets the style sheet class that this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

OBJECT

Inserts an object, such as a Flash piece, into the HTML document.

OBJECT Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BORDER	Sets the amount of border the element should have
CLASS	Sets the style sheet class that this element should use
CLASSID	Sets the class identifier for the element
CODE	Sets a URL for the location of the compiled class
CODEBASE	Sets the URL of the component for this element
CODETYPE	Sets the internet media type for this element
HEIGHT	Sets the height of this element
NAME	Sets the name for this element
STYLE	Sets style-sheet-specific information for this element
WIDTH	Sets the width of this element

OL

Creates an ordered list wherein each item is numbered with the LI tag.

OL Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
START	Sets a starting number for the element
STYLE	Sets style-sheet-specific information for this element
TYPE	Sets the type of ordered list this should be

OPTION

Specifies a list item for a select form element.

OPTION Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
LABEL	Sets the label for the element
SELECTED	Sets the element to be the selected item in the list
STYLE	Sets style-sheet-specific information for this element
VALUE	Sets the value for this element

P

Specifies a paragraph of text. Even though the ending tag is optional I recommend always putting it in for completeness.

P Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

PARAM

Specifies a parameter or a property in an OBJECT or an APPLET.

PARAM Attributes

Attribute	Description
ID	Sets the ID of the element
NAME	Sets the name of the element
TYPE	Sets the type of the element
VALUE	Sets the value of the element
VALUETYPE	Sets the type of the value specified

PRE

Renders text in a fixed-width font.

PRE Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

SAMP

Renders text in a smaller font for use as a code sample listing.

SAMP Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

SCRIPT

Denotes a client-side script block for the page that will be interpreted by the browser's scripting engine.

SCRIPT Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
SRC	Specifies a URL pointing to a scripting file
TYPE	Sets the language for the script block

SELECT

Creates a list box or a drop-down list in a form.

SELECT Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATASRC	Sets the source of the data for binding for this element

Attribute	Description
MULTIPLE	Sets whether you can select multiple items in the element
NAME	Sets the name of the element
SIZE	Sets the number of items to show at one time in the element
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tab order of this element

SMALL

Specifies a section of text that will be rendered in a smaller font than the current specified font.

SMALL Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

SPAN

Used with a style sheet to define non-standard attributes for a section of elements.

SPAN Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or Text
DATASRC	Sets the source of the data for binding for this element
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

STRONG

Renders a section of text in a bold-faced font.

STRONG Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

SUB

Renders text in subscript, using a smaller font than the current font specified.

SUB Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

SUP

Renders text in superscript, using a smaller font than the current font specified.

SUP Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

TABLE

Creates a table in the page that can contain TR, TD, and TH tags.

TABLE Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BACKGROUND	Sets a background image for the element
BGCOLOR	Sets the background color of the element
BORDER	Sets the amount of border the element should have
BORDERCOLOR	Sets the color of the border
BORDERCOLORDARK	Sets the color of the dark part of the border
BORDERLIGHTCOLOR	Sets the color of the light part of the border
CELLPADDING	Sets the amount of padding between the content and the border of the cell
CELLSPACING	Sets the amount of spacing between the cells
CLASS	Sets the style sheet class this element should use
FRAME	Sets the way the border frame around the table should display
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element
WIDTH	Sets the width of the element

TD

Creates a cell in a table.

TD Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BACKGROUND	Sets the background image of the element
BGCOLOR	Sets the background color of the element
BORDERCOLOR	Sets the color of the border
BORDERCOLORDARK	Sets the color of the dark part of the border
BORDERLIGHTCOLOR	Sets the color of the light part of the border
CLASS	Sets the style sheet class this element should use
COLSPAN	Sets the amount of cells this cell should span
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
NOWRAP	Sets the browser to not automatically wrap text
ROWSPAN	Sets the amount of rows this cell should span
STYLE	Sets style-sheet-specific information for this element
VALIGN	Sets the vertical alignment of this element
WIDTH	Sets the width of this element

TEXTAREA

Specifies a multi-line text input control for a form.

TEXTAREA Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
CLASS	Sets the style sheet class this element should use
COLS	Sets the number of columns in characters the text area should have
DATAFLD	Sets the field of a given data source to bind to this element
DATAFORMAT	Sets how to render the specified data source, as HTML or Text
DATASRC	Sets the source of the data for binding for this element
ID	Sets the ID of the element
NAME	Sets the name of the element
readonly	Sets the text area to read-only
ROWS	Sets the number of rows in characters the text area should have
STYLE	Sets style-sheet-specific information for this element
TABINDEX	Sets the tab order of the element
WRAP	Sets how the text should wrap

TH

Specifies a header cell in a table; the content is usually centered and bolded.

TH Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BACKGROUND	Sets the background image of the element
BGCOLOR	Sets the background color of the element
BORDERCOLOR	Sets the color of the border
BORDERCOLORDARK	Sets the color of the dark part of the border
BORDERLIGHTCOLOR	Sets the color of the light part of the border
CLASS	Sets the style sheet class this element should use
COLSPAN	Sets the amount of cells this cell should span
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
NOWRAP	Sets the browser to not automatically wrap text
ROWSPAN	Sets the amount of rows this cell should span
STYLE	Sets style-sheet-specific information for this element
VALIGN	Sets the vertical alignment of this element
WIDTH	Sets the width of this element

TITLE

Specifies the title of the document, which is contained in the heading of the document.

TR

Creates a row in a table.

TR Attributes

Attribute	Description
ALIGN	Sets the horizontal alignment of the element
BACKGROUND	Sets the background image of the element
BGCOLOR	Sets the background color of the element
BORDERCOLOR	Sets the color of the border
BORDERCOLORDARK	Sets the color of the dark part of the border
BORDERLIGHTCOLOR	Sets the color of the light part of the border
CLASS	Sets the style sheet class this element should use
HEIGHT	Sets the height of the element
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element
VALIGN	Sets the vertical alignment of this element
WIDTH	Sets the width of this element

U

Underlines the specified text.

U Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element

UL

Creates an un-ordered list. Items in the list are specified with the LI tag.

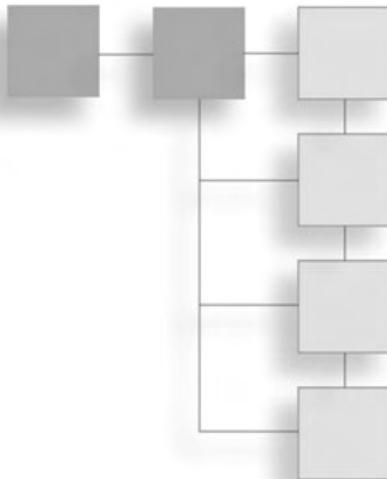
UL Attributes

Attribute	Description
CLASS	Sets the style sheet class this element should use
ID	Sets the ID of the element
STYLE	Sets style-sheet-specific information for this element
TYPE	Sets the type of bullet that LI will create

This page intentionally left blank

APPENDIX B

PHP LANGUAGE REFERENCE



This appendix is a complete guide to all of the PHP functions available to you. Each function is broken into sections that the function should fall under. Then each function is listed with the parameters it receives, what the function returns, and a short description of what the function does.

Apache Functions

Function	Returns	Description
apache_lookup_uri(string filename)	Class	Returns a class with properties about the URI specified.
apache_note(string note_name [, string note_value])	String	Retrieves the value in the request notes table for a specific note name. If the note_value is specified then it will set note_name to the value of note_value.
getallheaders()	Array	Returns an array of HTTP request headers from Apache.
virtual(string filename)	Integer	Performs an Apache sub-request to include the specified file.

Array Functions

Function	Returns	Description
array(...)	Array	Creates and returns an array with the supplied values.
array_keys(array)	Array	Returns an array that contains all the keys of the specified array.
array_merge(arrays)	Array	Merges the supplied arrays into one singular array.
array_pop(array)	Mixed	Pops the last element of the array and returns it.
array_push(array, variables)	Integer	Pushes the supplied variables onto the end of the specified array.
array_shift(array)	Mixed	Removes the first element from the array and returns it.

Function	Returns	Description
array_slice(array, offset [,length])	Array	Returns a sub-array from the specified array starting at index offset. If the length is specified then it will return an array starting from index offset with a specified length.
array_splice(input, offset [,length] [,replacement])	Array	Will remove a sub-array from input and replace it with the array replacement.
array_unshift(array, variables)	Integer	Adds the variables to the beginning of the array.
array_values(array)	Array	Returns an array containing all the values of the array.
array_walk(array, function [,parameter])	Bool	Traverses the supplied array, applying the logic of a specified function to the elements in the array.
arsort(array)	void	Sorts an array in descending order, retaining the key values.
asort(array)	void	Sorts an array in ascending order, retaining the key values.
compact(varnames)	Array	Merges the specified variables into a single array.
count(array)	Integer	Returns the count of elements in the specified array.
current(array)	Mixed	Returns the current element in the specified array.
each(array)	Array	Returns a four-element sub-array that contains the key and value of the current element. The key is contained in indices 0 and "key", and the value is contained in indices 1 and "value".
end(array)	Mixed	Sets the last element of the array to the current index and returns the value of that last element.
extract(array [,extract_type] [,prefix])	int	This will import variables into the symbol table from the specified array. The extract_type parameter tells extract() what to do if a collision is detected, and the prefix parameter specifies a prefix for the elements to be inserted.
in_array(value, array)	Boolean	Returns true if the value exists anywhere in the array, otherwise it returns false.
key(array)	Mixed	Returns the key of the current element in the array.
ksort(array)	Integer	Sorts the array by its key values.
list(variables)	void	Creates an array of the supplied variables.
next(array)	Mixed	Moves the current index pointer to the next index in the array. This will return false if you have reached the last index in the array.
pos(array)	Mixed	Returns the current element of the array.
prev(array)	Mixed	Moves the current index pointer to the previous index in the array. This will return false if you have reached the first index in the array.
range(low, high)	Array	Returns an array of the integers between low and high.
reset(array)	Mixed	Returns the first element of the array and sets the current index to the first available index in the array.
rsort(array)	void	Sorts the array in descending order.
shuffle(array)	void	Sorts the array into a random order
sizeof(array)	Integer	Returns the number of elements in the supplied array.
sort(array)	void	Sorts the array into ascending order.
uasort(array, function)	void	Sorts the array using the logic of the specified function.
uksort(array, function)	void	Sorts the array by the key values using the logic of the specified function.
usort(array, function)	void	Sorts the array using the logic of the specified function.

Aspell Functions

Function	Returns	Description
aspell_check(dictionary, word)	Boolean	Returns true if the spelling of the word is recognized in the specified dictionary.
aspell_check_raw(dictionary, word)	Boolean	Checks the spelling of the word with the dictionary without changing the string at all.
aspell_new(master, personal)	Integer	Loads a new dictionary and returns a link identifier to the new dictionary.
aspell_suggest(dictionary, word)	Array	Returns an array of suggested spellings for the specified word.

BCMath Functions

Function	Returns	Description
bcaadd(string1, string2, [scale])	String	Returns the sum of string1 and string2. The optional parameter scale specifies the number of decimal places to go out.
bccomp(string1, string2, [scale])	Integer	Compares string1 and string2. If they are equal it will return 0, if string1 is greater than string2 it will return 1, and if string2 is greater than string1 then it will return -1. The optional parameter scale specifies the number of decimal places to go out.
bcddiv(string1, string2, [scale])	String	Divides string1 by string2. The optional parameter scale specifies the number of decimal places to go out.
bcmod(string1, string2)	String	Returns the modulus of string1 and string2.
bcmul(string1, string2, [scale])	Multiples string1 by string2.	The optional parameter scale specifies the number of decimal places to go out.
bcpow(string1, string2, [scale])	String	Takes string1 to the power of string2. The optional parameter scale specifies the number of decimal places to go out.
bcscale(scale)	String	Sets the default scale parameter value.
bcsqrt(string1, [scale])	String	Returns the square root of string1. The optional parameter scale specifies the number of decimal places to go out.
bcsub(string1, string2, [scale])	String	Subtracts string2 from string1. The optional parameter scale specifies the number of decimal places to go out.

Calendar Functions

Function	Returns	Description
easter_date([year])	Integer	Returns the timestamp for midnight on Easter Day of the specified year. If year is not specified then the current year is assumed.
easter_days([year])	Integer	Returns the number of days after March 21 st on which Easter Day falls. If year is not specified then the current year is assumed.
FrenchToJD(month, day, year)	Integer	Converts a date in the French Republican calendar to a Julian Day.
GregorianToJD(month, day, year)	Integer	Converts the supplied Gregorian Date to a Julian Day.
JDDayOfWeek(julianday, mode)	Mixed	Returns the day of the week of the specified Julian Day in the specified mode.
JDMonthName(julianday, mode)	String	Returns the month name of the specified Julian Day in the specified mode.
JDTToFrench(julianday)	String	Converts a Julian Day into the French Republic calendar.

Function	Returns	Description
JDTogregorian(julianday)	String	Converts a Julian Day into a Gregorian date.
JDTojewish(julianday)	String	Converts a Julian Day into the Jewish calendar.
JDToJulian(julianday)	String	Converts a Julian Day count to a string representing a Julian Date.
JewishToJD(jewish)	String	Converts a Jewish date to a Julian Date.
JulianToJD(month, day, year)	String	Converts a string representing a Julian Date to a Julian Day.

Date and Time Functions

Function	Returns	Description
checkdate(month, day, year)	Boolean	Verifies that the specified month, day, and year is a valid date. If it is a valid date then true is returned, otherwise false is returned.
date(format, [timestamp])	String	Formats a date. If timestamp is not supplied then the current timestamp is used.
getdate(timestamp)	Array	Returns an array with date/time settings for the timestamp.
gettimeofday()	Array	Returns an array with settings for the current time.
gmdate(format, [timestamp], [minute], [second], [month], [day], [year], [is_dst])	String	Returns the timestamp for the GMT time/date that corresponds to the local time. Any of the optional parameters that are not used are assumed to be the current time.
gmstrftime(format, [timestamp])	String	Formats a GMT/CUT time/date according to the specified format.
microtime()	String	Returns a string containing the microseconds and seconds since the epoch.
mktime([hour], [minute], [second], [month], [day], [year], [is_dst])	Integer	Returns the timestamp for the specified date. Any of the optional parameters not specified are assumed to be the current time.
strftime(format, [timestamp])	String	Formats the local date/time according to the specified format.
time()	Integer	Returns a current timestamp.

DBA Functions

Function	Returns	Description
dba_close(dbHandle)	void	Closes the database that is specified.
dba_delete(key, dbHandle)	Boolean	Deletes the entry with the specified key in the specified database. If it succeeds then it returns true, otherwise it returns false.
dba_exists(key, dbHandle)	Boolean	Returns true if the specified key in the specified database exists, otherwise it returns false.
dba_fetch(key, dbHandle)	String	Returns a serialized string with the specified key in the database.
dba_firstkey(dbHandle)	String	Returns the first key in the specified database.
dba_insert(key, value, dbHandle)	Boolean	Inserts a key/value pair into the specified database. Returns true on success, false on failure.
dba_nextkey(dbHandle)	String	Returns the next key from the database.
dba_open(path, mode, dbType)	Integer	Returns a handle to the database with the specified path. Possible modes are "r", "w", "c", and "n".
dba_optimize(dbHandle)	Boolean	Optimizes the specified database. If the optimization succeeded then the function will return true, otherwise it will return false.
dba_popen(path, mode, dbType)	Integer	Opens the database in persistent mode. Returns a handle to the database with the specified path. Possible modes are "r", "w", "c", and "n".

Function	Returns	Description
dba_replace(key, value, dbHandle)	Boolean	Replaces or inserts a key/value pair into the specified database.
dba_sync(dbHandle)	Boolean	Synchronizes the database. Returns true on success, and false on failure.

Directory Functions

Function	Returns	Description
chdir(directory)	Boolean	Sets directory to the current directory.
closedir(dirHandle)	void	Closes the directory stream.
dir(directory)	Directory Object	Returns an object that represents directory.
opendir(directory)	Integer	Opens the specified directory and returns a handle to it.
readdir(dirHandle)	String	Returns the next entry in the specified directory handle.
rewinddir(dirHandle)	void	Resets the stream to the first object in the directory.

Dynamic Extension Loading

Function	Returns	Description
dl(extension)	Integer	Loads an extension dynamically. Returns 0 if loading failed and non-zero if loading succeeded.

Encryption Functions

Function	Returns	Description
mcrypt_cbc(cipher, key, data, mode, [iv])	String	Encrypts or decrypts the data in CBC mode.
mcrypt_cfb(cipher, key, data, mode, iv)	String	Encrypts or decrypts the data in CFB mode.
mcrypt_create_iv(size, source)	String	Creates an initialization vector from the source using random numbers.
mcrypt_ecb(cipher, key, data, mode)	String	Encrypts or decrypts the data in ECB mode.
mcrypt_get_block_size(cipher)	Integer	Returns the block size of the cipher,
mcrypt_get_cipher_name(cipher)	String	Returns the name of the specified cipher.
mcrypt_get_key_size(cipher)	Integer	Returns the size of the key for the cipher.
mcrypt_ofb(cipher, key, data, mode, iv)	String	Encrypts or decrypts the data in OFB mode.

Execution Functions

Function	Returns	Description
escapeshellcmd(command)	String	Escapes shell metacharacters in the command.
exec(command, [array], [return_var])	String	Executes the specified command. The array, if passed in, will receive any output from the command. If the return_var is specified this will contain the result code from the command.

Function	Returns	Description
passthru(command, [return_var])	void	Executes the specified command and displays the raw output.
system(command, [return_var])	String	Executes the specified command and displays the output.

Forms Data Format Functions

Function	Returns	Description
fdf_close(fdfdoc)	Bbool	Closes the FDF document
fdf_create()	Integer	Creates a new FDF document.
fdf_get_file(fdfdoc)	string	Returns the value of the /F key in the FDF document.
fdf_get_status(fdfdoc)	string	Returns the value of /STATUS key in the FDF document.
fdf_get_value(fdfdoc, fieldname)	string	Returns the value of the specified field in the FDF document.
fdf_next_field_name(fdfdoc, fieldname)	string	Returns the name of the field that follows the specified fieldname.
fdf_open(filename)	Integer	Opens a FDF document.
fdf_save(filename)	Bool	Saves a FDF document.
fdf_set_ap(fdfdoc, fieldname, face, filename, pagenumber)	Bool	Sets the appearance of the named field in the FDF document.
fdf_set_file(fdfdoc, filename)	Bool	Sets the value of the /F key in the FDF document.
fdf_set_status(fdfdoc, status)	Bool	Sets the value of the /STATUS key in the FDF document.
fdf_set_value(fdfdoc, fieldname, value, is_name)	Bool	Sets the value of the specified fieldname in the FDF document. The <i>is_name</i> argument specifies whether or not the value is to be a PDF name (1) or a string (0).

File System Functions

Function	Returns	Description
basename(path)	String	Returns the file name component from a fully qualified path.
chgrp(filename, group)	Bool	Changes the group of the filename.
chmod(filename, mode)	Bool	Changes the mode of the filename.
chown(filename, user)	Bool	Changes the owner of the filename.
clearstatcache()	void	Clears the state cache.
copy(source, dest)	Bool	Copies a file from the source to the destination.
dirname(path)	String	Returns the directory name component from a fully qualified path.
diskfreespace(dir)	float	Returns the amount of free space in the specified directory.
fclose(fp)	Bool	Closes a file stream.
feof(fp)	Boolean	Checks to see if you are at the end of the specified file stream. If you are at the end of the stream feof() will return true, otherwise it will return false.
fgetc(fp)	String	Reads the next character from the file stream.
fgetcsv(fp, length, [delimiter])	Array	Returns an array with the next line separated by commas or a specified delimiter.
fgets(p, length)	String	Reads a line of up to length-1.
fegetss(fp, length)	String	Reads a line of up to length-1 while stripping out any HTML tags.
file(filename)	Array	Reads the entire specified file into an array.
file_exists(filename)	Boolean	Returns true if the file exists, and false if the file does not exist.

Function	Returns	Description
fileatime(filename)	Integer	Returns the time the file was last accessed.
filectime(filename)	Integer	Returns the time the file was last changed.
filegroup(filename)	Integer	Returns the ID of the owner group.
fileinode(filename)	Integer	Returns the inode number for the file.
filemtime(filename)	Integer	Returns the time the file was last modified.
fileowner(filename)	Integer	Returns the ID of the owner of the file.
fileperms(filename)	Integer	Returns the permissions of the specified file.
filesize(filename)	Integer	Returns the size in bytes of the file.
filetype(filename)	String	Returns the type of file.
flock(fp, operation)	Boolean	Sets or releases a lock on the specified file stream.
fopen(filename, mode)	Integer	Opens a file in the specified mode.
fpassthru(fp)	Integer	Outputs data from the current position in the file to the end of the file.
fputs(fp, string, [length])	Integer	Writes the specified string to the file stream.
fread(fp, length)	String	Reads length bytes from the specified file stream.
fseek(fp, offset)	Integer	Moves the file pointer to offset in the specified file stream.
ftell(fp)	Integer	Returns the current position of the file pointer.
fwrite(fp, string, [length])	Integer	Writes string to the specified file stream.
is_dir(filename)	Boolean	Checks to see if the specified file is a directory.
is_executable(filename)	Boolean	Checks to see if the specified file is a executable.
is_file(filename)	Boolean	Checks to see if the specified file is a file.
is_link(filename)	Boolean	Checks to see if the specified file is a symbolic link.
is_readable(filename)	Boolean	Checks to see if the specified file is readable.
is_writable(filename)	Boolean	Checks to see if the specified file is writable.
link(target, link)	Boolean	Creates a link.
linkinfo(path)	Integer	Returns information about the specified link.
lstat(filename)	Array	Returns information about the specified file.
mkdir(pathname, mode)	Boolean	Creates a directory with the specified mode.
pclose(fp)	Integer	Closes a file pointer to a pipe that has been opened with the <code>popen()</code> function.
popen(command, mode)	Integer	Opens a pipe by using the specified command.
readfile(filename)	Integer	Reads and outputs the specified file.
readlink(path)	String	Returns the target of the specified symbolic link.
rename(from, to)	Boolean	Renames a file from a file name to a file name.
rewind(fp)	Boolean	Resets the file pointer to the beginning of the specified file stream.
rmdir(path)	Boolean	Removes the specified directory.
set_file_buffer(fp, buffer)	Integer	Sets the size of the buffer for the file stream.
stat(filename)	Array	Returns information about the specified file.
symlink(target, link)	Boolean	Creates a symbolic link.
tempnam(dir, prefix)	String	Creates a unique temporary file name in the specified directory.
touch(filename, time)	Boolean	Sets the modification time of the specified file name.
umask([mask])	Integer	Sets PHP's specific umask and returns the old umask.
unlink(filename)	Boolean	Deletes the specified file.

General Math Functions

Function	Returns	Description
abs(number)	Mixed	Returns the absolute value of the number.
acos(arg)	Float	Returns the arc cosine of arg.
asin(arg)	Float	Returns the arc sin of arg.
atan(arg)	Float	Returns the arc tan of arg.
atan2(y, x)	Float	Returns the arc tan of y and x.
base_convert(number, base1, base2)	String	Converts number from base1 to base2.
Bindec(binary)	Integer	Converts a binary string to a decimal string.
ceil(number)	Float	Returns the ceiling of a number.
cos(arg)	Float	Returns the cosine of arg.
DecBin(number)	String	Converts a decimal number to a binary number.
DecHex(number)	String	Converts a decimal number to a hex number.
DecOct(number)	String	Converts a decimal number to a octal number.
exp(arg)	Float	Returns e to the power of arg.
floor(number)	Float	Returns the floor of a number.
getrandmax()	Integer	Show the greatest random number that can be returned from rand().
HexDec(number)	Integer	Converts a hex number into a decimal.
log(arg)	Float	Returns the natural log of arg.
log10(arg)	Float	Returns the base 10 log of arg.
max(arg1, arg2, ...)	Mixed	Returns the greatest value in the list of arguments.
min(arg1, arg2, ...)	Mixed	Returns the smallest value in the list of arguments.
mt_getrandmax()	Integer	Returns the largest value the mt_rand() can return.
mt_rand([min], [max])	Integer	Returns a Mersenne Twister random value.
mt_srand(seed)	void	Seeds the Mersenne Twister random number generator,
number_format(number, [dex_place], [dec_point], [thousands])	String	Formats the specified number to a certain number of decimal places.
OctDec(number)	Integer	Converts a octal number to a decimal number.
pi()	Float	Returns pi.
pow(x, y)	Float	Returns x to the power of y.
rand([min], [max])	Integer	Returns a random number.
round(number, precision)	Float	Returns the rounded value of number to the specified precision.
sin(arg)	Float	Returns the sin of arg.
srand(seed)	void	Seeds the random number generator.
tan(arg)	Float	Returns the tangent of arg.

HTTP Functions

Function	Returns	Description
header(string)	Integer	Sends the specified HTTP header.
setcookie(name, [value], [expire], [path], [domain], [secure])	Boolean	Creates a cookie on the client's computer with the specified name and value.

Image Functions

Function	Returns	Description
GetImageSize(filename, [imageinfo])	Array	Returns the size of the image.
ImageArc(img, x, y, width, height, start, end, col)	Integer	Draws a partial ellipse in the specified image, centered at x, y with a specific width and height.
ImageChar(img, font, x, y, character, color)	Integer	Draws the specified character at x, y in the image.
ImageCharUp(img, font, x, y, character, color)	Integer	Draws the specified character facing upwards in the image.
ImageColorAllocate(img, red, green, blue)	Integer	Allocates the specified RGB color value for the image.
ImageColorAt(img, x, y)	Integer	Returns the index of the color at x, y.
ImageColorClosest(img, red, green, blue)	Integer	Returns the index to the closest color in the color palette for the image.
ImageColorExact(img, red, green, blue)	Integer	Returns the index of the specified color in the color palette of the image.
ImageColorResolve(img, red, green, blue)	Integer	Finds the specified color in the palette; if it doesn't exist then it returns the index to the closest color in the palette.
ImageColorSet(img, index, red, green, blue)	Boolean	Sets the specified index to the RGB color value.
ImageColorsForIndex(img, index)	Array	Returns an array containing the RGB values for the specified index in the color palette.
ImageColorsTotal(img)	Integer	Returns the number of colors in the specified images color palette.
ImageColorTransparent(img, [color])	Integer	Sets color as the transparent color in the palette.
ImageCopyResized(dest_img, src_img, destX, destY, srcX, srcY, destWidth, destHeight, srcWidth, srcHeight)	Integer	Copies an area from the source image to an array of the destination image. If the heights are different then the destination image is resized.
ImageCreate(width, height)	Integer	Creates a new image.
ImageCreateFromGif(filename)	Integer	Creates a new image from the specified file.
ImageDashedLine(img, x1, y1, x2, y2, color)	Integer	Draws a dashed line in the image from point x1, y1 to point x2, y2.
ImageDestroy(img)	Integer	Destroys the specified image.
ImageFill(img, x, y, color)	Integer	Fills the image starting at point x, y.
ImageFilledPolygon(img, points, num_points, color)	Integer	Draws a filled polygon in the image between the points.
ImageFilledRectangle(img, x1, y1, x2, y2, color)	Integer	Draws a filled rectangle in the specified image.
ImageFillToBorder(img, x, y, border,color)	Integer	Performs a flood fill between the specified border color starting at point x, y.
ImageFontHeight(font)	Integer	Returns the height of the specified font in pixels.
ImageFontWidth(font)	Integer	Returns the width of the specified font in pixels.
ImageGif(img, [filename])	Integer	Sends the image to a file or to the browser.
ImageInterlace(img, [interlace])	Integer	Turns interlacing on or off in the image.
ImageLine(img, x1, y1, x2, y2, color)	Integer	Draws a line in the image from point x1, y1 to point x2, y2.

Function	Returns	Description
ImageLoadFont(filename)	Integer	Loads a bitmap font from the file.
ImagePolygon(img, points, num_points, color)	Integer	Draws a polygon in the image, much like polygon fill but it doesn't fill the image.
ImagePSBBox(text, font, size, space, width, angle)	Array	Calculates the coordinates for the bounding box of text area using a PostScript font.
ImagePSEncodeFont(filename)	Integer	Loads a specific character-encoding vector for a PostScript font.
ImagePSFreeFont(fontindex)	void	Releases the PostScript font from memory.
ImagePSLoadFont(filename)	Integer	Loads a PostScript font into memory.
ImagePSText(img, text, font, size, foreground, background, x, y, [space],[tightness], [angle], [antialias_steps])	Array	Draws the text to the image using a PostScript font.
ImageRectangle(img, x1, y1, x2, y2, color)	Integer	Draws a rectangle to the image. Acts much like ImageRectangleFill() but the rectangle is not filled when it is drawn.
ImageSetPixel(img, x, y, color)	Integer	Sets the pixel at point x, y to a color.
ImageString(img, font, x, y, size, color)	Integer	Draws a string starting at x, y to the image.
ImageStringUp(img, font, x, y, size, color)	Integer	Draws a string starting at point x, y facing upwards.
ImageSX(img)	Integer	Gets the width of the image.
ImageSY(img)	Integer	Gets the height of the image.
ImageTTFBox(size, angle, font, text)	Array	Returns a bounding box for a TrueType font.
ImageTTFFText(img, text, size, angle, x, y, color)	Array	Draws the text in the image using the specified TrueType font.

IMAP Functions

Function	Returns	Description
imap_8bit(string)	String	Converts a 8-bit string to a printable string.
imap_alerts()	Array	Returns an array of all the IMAP alert messages that have occurred.
imap_append(stream, mailbox, message, flags)	Boolean	Appends a message to the specified mailbox.
imap_base64(text)	String	Decodes the specified base-64 encoded text.
imap_binary(string)	String	Converts a 8-bit string to a base-64 string.
imap_body(stream, message_num, flags)	String	Returns the text of the specified message number.
imap_check(stream)	Array	Gets information about the specified mailbox.
imap_clearflag_full(stream, sequence, flag, options)	Boolean	Clears a specific flag on the stream.
imap_close(stream, flags)	Boolean	Closes a previously opened IMAP stream.
imap_create_mailbox(stream, mailbox)	Boolean	Creates a mailbox.
imap_delete(stream, message_num)	Boolean	Marks a message for deletion.
imap_delete_mailbox(stream, mailbox)	Boolean	Deletes a mailbox.

Function	Returns	Description
imap_errors()	Array	Returns an array of all the IMAP errors that have occurred.
imap_expunge(stream)	Boolean	Deletes all the marked messages.
imap_fetchbody(stream, message_num, part_num, flags)	String	Gets the specified section of the message.
imap_fetchheader(stream, message_num, flags)	String	Gets the header for the specified message number.
imap_fetchstructure(stream, message_num)	Array	Returns the structure of the message.
imap_getmailboxes(stream, ref, pat)	Array	Returns an array of the mailboxes.
imap_getsubscribed(stream, ref, pat)	Array	Returns an array of all mailboxes in which a user is currently subscribed.
imap_header(stream, message_num, fromlength, subjectlength, defaulthost)	Object	Returns an object that represents the header of the mail.
imap_headers(stream)	Array	Returns an array that contains all the headers for every message in the stream.
imap_last_error()	String	Gets the last error that occurred in IMAP.
imap_listmailbox(stream, ref, pat)	Array	Returns an array that contains all the mailbox names.
imap_listsubscribed(stream, ref, pat)	Array	Returns an array that contains all the subscribed mailboxes in the stream.
imap_mail_copy(stream, messagelist, mailbox, flags)	Boolean	Copies a set of messages to another mailbox.
imap_mail_move(stream, messagelist, mailbox)	Boolean	Moves a set of messages to another mailbox.
imap_mailboxmsginfo(stream)	Array	Gets information about the current mailbox.
imap_msgno(stream, UID)	Integer	Returns the message number for the UID.
imap_num_msg(stream)	Integer	Gets the total number of messages in the mailbox.
imap_num_recent(stream)	Integer	Gets the total number of new messages in the mailbox.
imap_open(mailbox, username, password, flags)	Integer	Opens a IMAP stream for a specific mailbox.
imap_ping(stream)	Boolean	Pings the IMAP stream.
imap_qprint(string)	String	Converts a printable string to a 8-bit string.
imap_rename_mailbox(stream, oldname, newname)	Boolean	Renames a mailbox from oldname to newname.
imap_reopen(stream, mailbox, [flags])	Boolean	Reopens the IMAP stream.
imap_rfc822_parse_adrlist(address, default_host)	Array	Parses an address string and returns an array that contains the mailbox, host, personal name, and domain source.
imap_rfc822_write_address(mailbox, host, personal)	String	Returns a e-mail address.
imap_scannmailbox(stream, string)	Array	Returns an array of messages that match the search string.
imap_setflag_full(stream, sequence, flag, options)	Boolean	Sets the specified flag.
imap_sort(stream, criteria, reverse, options)	Array	Returns an array of message numbers that meet the search criteria.
imap_status(stream, mailbox, options)	Object	Gets information about the mailbox.

Function	Returns	Description
imap_subscribe(stream, mailbox)	Boolean	Subscribes to the mailbox.
imap_uid(stream, message_num)	Integer	Returns the UID for the message.
imap_undelete(stream, message_number)	Boolean	Unmarks a message for deletion.
imap_unsubscribe(stream, mailbox)	Boolean	Unsubscribes from the mailbox.

Informix Functions

Function	Returns	Description
ifx_affected_rows(id)	Integer	Returns the number of rows that were affected by the query.
ifx_blobinfile_mode(mode)	void	Sets the default mode for SELECT statements,
ifx_byteasvarchar(mode)	void	Sets the default byte mode for SELECT statements.
ifx_close([linked])	Integer	Closes the currently opened connection.
ifx_connect([database], [userid], [password])	Integer	Opens a connection to the Informix database.
ifx_copy_blob(id)	Integer	Copies a BLOB object.
ifx_create_blob(type, mode, param)	Integer	Creates a BLOB.
ifx_create_char(param)	Integer	Creates a char object.
ifx_do(id)	Integer	Executes a prepared SQL statement.
ifx_error()	String	Returns the last error that occurred in Informix.
ifx_errormsg([errorcode])	String	Returns the last error that occurred or the error message for the specified code.
ifx_fetch_row(id, [position])	Array	Fetches a row from the database.
ifx_fieldproperties(id)	Array	Returns an array of field names and their properties.
ifx_fieldtypes(id)	Array	Returns an array of field names and their types.
ifx_free_blob(id)	Integer	Releases the BLOB object from memory.
ifx_free_char(id)	Integer	Releases the specified char object from memory.
ifx_free_result(id)	Integer	Releases the result set from memory.
ifx_free_slob(id)	Integer	Releases the SLOB object from memory.
ifx_get_blob(id)	Integer	Returns the specified BLOB object.
ifx_get_char(id)	Integer	Returns the specified char object.
ifx_getsqlca(id)	Array	Returns the contents of sqlca.sqlerrd[0-5] after a query has been run.
ifx_htmltbl_result(id, [html_table_options])	Integer	Outputs the record set as a HTML table.
ifx_nullformat(mode)	void	Sets the default return value to NULL.
ifx_num_fields(id)	Integer	Gets the number of fields in the record set.
ifx_num_rows(id)	Integer	Gets the number of rows in the record set.
ifx_pconnect([database], [userid], [password])	Integer	Opens a persistent connection to a database.
ifx_prepare(query, [linked], [cursor_type], [blobidarray])	Integer	Prepares a SQL query to execute.
ifx_query(query, [linked], [cursor_type], [blobidarray])	Integer	Runs the specified SQL query against the open database.
ifx_textasvarchar(mode)	void	Sets the default text mode for SELECT statements.
ifx_update_blob(id, content)	Boolean	Updates the content of a BLOB object.

Function	Returns	Description
ifx_update_char(id, content)	Integer	Updates the content in the char object.
ifxus_close_slob(id)	Integer	Releases the specified SLOB object from memory.
ifxus_create_slob(mode)	Integer	Creates a SLOB object.
ifxus_open_slob(id, mode)	Integer	Opens a SLOB object.
ifxus_read_slob(id, bytes)	Integer	Reads a specified number of bytes from a SLOB object.
ifxus_seek_slob(id, mode, offset)	Integer	Sets the current pointer position in the SLOB object.
ifxus_tell_slob(id)	Integer	Gets the current position of the pointer in the SLOB object.
ifxus_write_slob(id, string)	Integer	Writes a string to the SLOB object.

LDAP Functions

Function	Returns	Description
ldap_add(link_id, dn, entry)	Boolean	Adds an entry for the specified dn.
ldap_bind(link_id, [bind_rdn], [password])	Boolean	Binds the LDAP directory with a specified RDN and password.
ldap_close(link_id)	Boolean	Closes an open link to the LDAP directory.
ldap_connect([hostname], [port])	Integer	Connects to an LDAP server.
ldap_count_entries(link_id, result)	Integer	Returns the number of entries in the search.
ldap_delete(lnk_id, dn)	Boolean	Deletes a dn from a directory in LDAP.
ldap_dn2ufn(dn)	Integer	Converts a specified dn to a user-friendly name.
ldap_explode_dn(dn, [attributes])	Array	Splits a dn into component parts.
ldap_first_attribute(link_id, result, ber_id)	String	Retrieves the first attribute in the entry.
ldap_first_entry(link_id, result)	Integer	Gets the result id for the specified entry.
ldap_free_result(result)	Boolean	Releases the specified result from memory.
ldap_get_attributes(link_id, result)	Array	Gets the attributes for the specified entry.
ldap_get_dn(link_id, result)	String	Gets the dn for the entry.
ldap_get_entries(link_id, result)	Array	Returns an array of entries.
ldap_get_values(link_id, result, attributes)	Array	Gets an array of values for the specified attributes.
ldap_list(link_id, base_dn, filter, [attributes])	Integer	Performs a LDAP_SCOPE_ONELEVEL search with the specified filters.
ldap_mod_add(link_id, dn, entry)	Boolean	Adds an attribute to the specified link.
ldap_mod_replace(link_id, dn, entry)	Boolean	Replaces an attribute in the specified link.
ldap_modify(link_id, dn, entry)	Boolean	Modifies the specified entry.
ldap_next_attribute(link_id, result, ber_id)	String	Gets the next attribute in the result set.
ldap_next_entry(link_id, result)	Integer	Gets the next entry for the specified result set.
ldap_read(link_id, base_dn, filter, [attributes])	Integer	Performs a LDAP_SCOPE_BASE search using the specified filters.
ldap_search(link_id, base_dn, filter, [attributes])	Integer	Performs a LDAP_SCOPE_SUBTREE search with the specified filters.
ldap_unbind(link_id)	Boolean	Unbinds the specified LDAP directory.

Mail Function

Function	Returns	Description
mail(to, subject, message, [headers])	Boolean	Sends an e-mail from the admin of PHP to the specified e-mail account.

PHP Options

Function	Returns	Description
error_log(message, message_type, [dest], [headers])	Integer	Sends an error message.
error_reporting([level])	Integer	Sets or returns the current error reporting level.
extension_loaded(extension_name)	Boolean	Returns true if the specified extension is loaded, otherwise it returns false.
get_cfg_var(var)	String	Returns the value of the specified PHP configuration.
get_current_user()	String	Returns the value of the current user.
get_magic_quotes_gpc()	Long	Returns the current settings for magic quotes.
get_magic_quotes_runtime()	Long	Returns the current settings for magic quotes.
getenv(var)	String	Returns the value of a specific environment variable.
getlastmod()	Integer	Returns the time when the current page was last modified.
getmyinode()	Integer	Returns the inode of the current running script.
getmypid()	Integer	Returns the process ID that PHP is running on.
getmyuid()	Integer	Returns the user ID for the current PHP script's owner.
getrusage([who])	Array	Returns the current resource usage.
phpinfo()	Integer	Outputs all configuration information about the PHP configuration.
phpversion()	String	Returns the current version of PHP that you are running.
putenv(value)	void	Sets the value of the environment variable.
set_magic_quotes_runtime(setting)	Boolean	Enables or disables magic quotes.
set_time_limit(seonds)	void	Sets the timeout for PHP scripts.

Miscellaneous Functions

Function	Returns	Description
connection_aborted()	Integer	Indicates that a connection has been aborted by a user.
connection_status()	Integer	Gets the status of the connection.
connection_timeout()	Boolean	Indicates that the script has timed out.
die(message)	void	Outputs the message and kills the execution of the script.
eval(string)	Mixed	Evaluates the string as PHP code.
exit()	void	Stops execution of the current script.
function_exists(name)	Boolean	Checks to see whether or not a function exists.
ignore_user_abort([setting])	Integer	Sets or returns whether a client disconnecting will stop execution of a script.
iptcparse(iptcblock)	Array	Parses the iptcblock into an array.
leak(bytes)	void	Leaks the specified amount of memory.
pack(format, [args])	String	Packs the argument into a binary string.

Function	Returns	Description
register_shutdown_function(function)	void	Specifies a function to execute when the current script terminates.
serialize(data)	String	Serializes the data into a string.
sleep(seconds)	void	Pauses the execution of the script for the specified amount of seconds.
uniqid(prefix)	String	Generates a unique id based on the current time.
unserialize(data)	Mixed	Unserializes the string.
usleep(microseconds)	void	Pauses the script for the specified amount of microseconds.

Network Functions

Function	Returns	Description
checkdnsrr(host, [type])	Integer	Searches DNS files for the host and specific type of record.
closelog()	Integer	Closes the open stream to the current log.
debugger_off()	Integer	Disables the remote debugger.
debugger_on(server)	Integer	Enables the remote debugger.
fsockopen(hostname, port, [error_num], [error_str], [timeout])	Integer	Opens a socket connection to the specified host and port.
gethostbyaddr(ip_address)	String	Returns the name of the host.
gethostbyname(hostname)	String	Returns the IP address of the host.
gethostbynamel(hostname)	Array	Returns an array of IP addresses that the specific host contains.
getmxrr(hostname, mxhost, [weight])	Integer	Returns the MX record for the specific host name from DNS.
openlog(identity, option, facility)	Integer	Opens a stream to the log file.
pfsocopen(hostname, port, [error_num], [eror_str], [timeout])	Integer	Opens a persistent socket connection to the specified host and port.
stream_set_blocking (stream, mode)	Bool	Sets the blocking mode or a specific socket.
syslog(priority, message)	Integer	Writes a message to the open log file stream.

NIS Functions

Function	Returns	Description
yp_err_string(errorcode)	String	Gets the last error that occurred.
yp_errno()	Integer	Returns the error code for the last error that occurred.
yp_first(domain, map)	Array	Gets the first key/value pair from the map.
yp_get_default_domain()	Integer	Returns the machine's default NIS domain name.
yp_master(domain, map)	String	Gets the key/value pair for the master NIS domain.
yp_match(domain, map, key)	String	Gets the value of the specified key.
yp_next(domain, map, key)	String	Retrieves the next key/value pair from the pointer.
yp_order(domain, map)	Integer	Returns the order number of the map.

Perl-Compatible Regular Expression Functions

Function	Returns	Description
preg_grep(pattern, input)	Array	Returns an array of the matching patterns from the input.
preg_match(pattern, input, [matches])	Integer	Performs a regular expression match on the input.
preg_match_all(pattern, input, matches, [order])	Integer	Matches all patterns in the input.
preg_quote(string)	String	Escapes special expression characters.
preg_replace(pattern, replacement, string)	Mixed	Replaces the string with replacement where it matches the pattern.

Regular Expression Functions

Function	Returns	Description
ereg(pattern, string, [regs])	Boolean	Searches the specified string for strings that match the regular expression pattern.
ereg_replace(pattern, replacement, string)	String	Replaces the string with replacement where it matches the pattern.
eregi(pattern, string, [regs])	Boolean	Performs a case-insensitive search on the string for the regular expression pattern.
eregi_replace(pattern, replacement, string)	String	Replaces any string with replacement where it matches the pattern, insensitive to case.
split(pattern, string, [limit])	Array	Splits the string into an array using the specified regular expression pattern.
sql_regcase(string)	String	Returns a regular expression from a case-insensitive match of the specified string.

Semaphore and Shared Memory Functions

Function	Returns	Description
sem_acquire(sem_id)	Boolean	Acquires a semaphore.
sem_get(key, [max_acquire], [perm])	Integer	Returns a semaphore ID.
sem_release(sem_id)	Boolean	Releases a specific semaphore from memory.
shm_attach(key, [memsize], [perm])	Integer	Creates a shared memory segment.
shm_detach(shm_id)	Boolean	Disconnects the shared memory segment.
shm_get_var(shm_id, variable_key)	Mixed	Returns a variable from the shared memory.
shm_put_var(shm_id, key, value)	Integer	Puts a variable into the shared memory segment.
shm_remove(shm_id)	Integer	Deletes a shared memory segment.
shm_remove_var(shm_id, key)	Integer	Removes a variable from the shared memory segment.

Session Functions

Function	Returns	Description
session_decode(string)	Boolean	Decodes the session data.
session_destroy()	Boolean	Ends the current session.
session_encode()	String	Encodes the session data.

Function	Returns	Description
session_id([sid])	String	Sets or gets the current session id.
session_is_registered(var)	Boolean	Checks if the specified variable is registered in the current session.
session_module_name([module])	String	Sets or gets the name of the current session module.
session_name([name])	String	Sets or gets the name of the current session.
session_register(var)	Boolean	Registers the specified variable with the current session.
session_save_path([path])	String	Sets or gets the current path where sessions are being saved.
session_start()	Boolean	Starts a session.
session_unregister(var)	Boolean	Unregisters a variable with the current session.

SNMP Functions

Function	Returns	Description
snmp_get_quick_print()	Boolean	Returns the value of the quick_print setting.
snmp_set_quick_print(Boolean)	void	Sets the value of the quick_print setting.
snmpget(hostname, community, object_id, [timeout], [retries])	String	Gets an SNMP object.
snmpset(hostname, community, object_id, type, value, [timeout], [retries])	Boolean	Sets the SNMP object.
snmpwalk(hostname, community, object_id, [timeout], [retries])	Array	Gets an array of all the SNMP objects.
snmpwalkoid(hostname, community, object_id [timeout], [retries])	Array	Gets an array of object IDs and their corresponding values.

String Functions

Function	Returns	Description
addslashes(string)	String	Adds escape slashes to the string.
bin2hex(string)	String	Converts the binary string into a hexadecimal string.
chop(string)	String	Removes trailing whitespace.
chr(ascii)	String	Returns the character for the ASCII code.
chunk_split(string, [chunk_len], [end])	String	Splits the string into smaller chunks by inserting the character end at every specified chunk_len.
convert_cyr_string(string, from, to)	String	Converts the string from one Cyrillic set to another.
crypt(string, [salt])	String	DES-encrypts the string using the salt value.
echo(string)	void	Outputs a string to the screen.
explode(separator, string)	Array	Splits the string into an array.
flush()	void	Flushes the output buffer.
get_meta_tags(filename, [include_path])	Array	Gets an array of all the META tags from the specified file.
htmlentities(string)	String	Converts the string into an HTML entity.
htmlspecialchars(string)	String	Converts any HTML special characters in the string into a HTML entity.
implode(delimiter, array)	String	Does the exact opposite of the explode function. It joins the array into a single string using the specified delimiter.

Function	Returns	Description
join(delimiter, array)	String	Joins the array into a single string using the specified delimiter.
ltrim(string)	String	Strips the white space from the beginning of the string.
md5(string)	String	Calculates the MD5 hash of the specified string.
n12br(string)	String	Inserts the tag before all the line breaks in the string.
ord(string)	int	Returns the specified ASCII value of the first letter of the string.
parse_str(string)	void	Parses the string into variables like it was a query string.
print(string)	Integer	Prints the specified string.
printf(string, [arg])	void	Outputs a formatted string.
quoted_printable_decode(string)	String	Converts a quoted printable string to a 8-bit string.
QuoteMeta(string)	String	Escapes meta characters in the string.
rawurldecode(string)	String	Decodes a URL-encoded string.
rawurlencode(string)	String	Encodes a string to a URL-encoded string.
setlocale(category, locale)	String	Sets the locale information for functions in the specified category.
similar_text(string1, string2, [percent])	int	Calculates the similarity between string1 and string2.
soundex(string)	String	Calculates the soundex key for the string.
sprintf(format, [args])	String	Returns a formatted string.
str_replace(pattern, replacement, string)	String	Replaces all occurrences of pattern with replacement in string.
strchr(string1, string2)	String	Finds the first occurrence of string2 in string1.
strcmp(string1, string2)	int	Compares string1 against string2.
strcspn(string1, string2)	int	Returns the number of characters in the beginning of string1 that do not match the beginning of the characters in string2.
strip_tags(string)	String	Strips all the HTML and PHP tags from the specified string.
stripslashes(string)	String	Strips all escape character slashes from the string.
strlen(string)	int	Returns the length in characters of the string.
strpos(string1, string2)	int	Finds the first occurrence of string2 in string1.
strrev(string)	String	Returns the specified string in reverse order.
strrpos(string1, string2)	int	Finds the last occurrence of string2 in string1.
strstr(string1, string2)	String	Finds the first occurrence of string2 in string1.
strtok(string1, string2)	String	Tokenizes string1 into segments separated by string2.
strtolower(string)	String	Converts all characters in the string to lowercase.
strtoupper(string)	String	Converts all characters in the string to uppercase.
strtr(string, from, to)	String	Replaces all occurrences of from in string with to.
substring(string, start, [length])	String	Returns the characters in string from the specified start point.
ucfirst(string)	String	Converts the first character of the string to uppercase.
ucwords(string)	String	Converts the first character of each word to uppercase.

URL Functions

Function	Returns	Description
base64_decode(string)	String	Decodes the specified base64 string.
base64_encode(string)	String	Returns a base64 encoded string.
parse_url(URL)	Array	Breaks up the specified URL into an array.

Function	Returns	Description
urldecode(string)	String	Decodes the URL-encoded string.
urlencode(string)	String	URL encodes a string.

Variable Functions

Function	Returns	Description
doubleval(var)	Integer	Converts the variable to a double.
empty(var)	Boolean	Checks to see if the variable has a non-zero value.
gettype(var)	String	Gets the data type of the variable.
intval(var, [base])	Integer	Gets the value of the variable using the specified base.
is_array(var)	Boolean	Checks to see if the variable is an array.
is_double(var)	Boolean	Checks to see if the variable is a double.
is_float(var)	Boolean	Checks to see if the variable is a float.
is_int(var)	Boolean	Checks to see if the variable is an int.
is_long(var)	Boolean	Checks to see if the variable is a long.
is_object(var)	Boolean	Checks to see if the variable is an object.
is_real(var)	Boolean	Checks to see if the variable is a real number.
is_string(var)	Boolean	Checks to see if the variable is a string.
isset(var)	Boolean	Checks to see if the variable has been set yet.
settype(var, type)	Boolean	Sets the variable to the specified type.
strval(var)	String	Returns the string value of the variable.
unset(var)	void	Unsets a set variable.

WDDX Functions

Function	Returns	Description
wddx_add_vars(packet_id, vars)	Boolean	Serializes the variables then adds them to the specified packet_id.
wddx_deserialize(packet)	Mixed	Deserializes the specified packet.
wddx_packet_end(packet_id)	String	Ends the specified packet.
wddx_packet_start([comment])	Integer	Starts a new packet.
wddx_serialize_value(var, [comment])	String	Serializes a single value into the packet.
wddx_serialize_vars(vars)	String	Serializes an array of variables into the packet.

Compression Functions

Function	Returns	Description
gzclose(gz)	Integer	Closes a gz file stream.
gzeof(gz)	Integer	Checks to see if you are at the end of the gz file stream.
gzfile(filename)	Array	Reads the entire contents of a gz file into an array.
gzgetc(gz)	String	Gets a character from the gz file stream.
gzgets(gz, length)	String	Gets a line from the gz file stream.
gzgetss(gz, length)	String	Gets a line from the gz file stream stripping HTML.

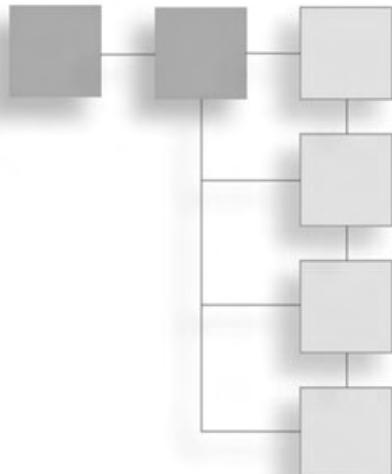
Function	Returns	Description
gzopen(filename, mode)	Integer	Opens the specified gz file.
gzpassthru(gz)	Integer	Outputs the gz file from the current file pointer.
gzputs(gz, string, [length])	Integer	Writes the string to the gz file stream.
gzread(gz, length)	String	Reads the specified amount of bytes from the gz file stream.
gzrewind(gz)	Integer	Resets the file pointer to the beginning of the gz file stream.
gzseek(gz, offset)	Integer	Sets the file pointer to the offset in the gz file stream.
gztell(gz)	Integer	Gets the current position of the file pointer in the gz file stream.
readgzfile(filename)	Integer	Reads the file and outputs the contents.
gzwrite(gz, string, [length])	Integer	Writes the string to the gz file stream.

XML Parser Functions

Function	Returns	Description
utf8_decode(string)	String	Converts the UTF-8 string to an ISO-8859-1 string.
utf8_encode(string)	String	Converts a ISO-8859-1 string to a UTF-8 string.
xml_error_string(code)	String	Returns the error for the specified code.
xml_get_current_byte_index(parser)	Integer	Returns the current byte index.
xml_get_current_column_number(parser)	Integer	Returns the current column number the parser is on.
xml_get_current_line_number(parser)	Integer	Returns the current line number the parser is on.
xml_get_error_code(parser)	Integer	Returns the last error code that occurred.
xml_parse(parser, data, [is_final])	Boolean	Parses the data.
xml_parser_create([encoding])	Integer	Creates a XML parser.
xml_parser_free(parser)	Boolean	Destroys the XML parser.
xml_parser_get_option(parser, option)	Mixed	Gets the value of the specified option.
xml_parser_set_option(parser, option, value)	Boolean	Sets the value of the specified option
xml_set_character_data_handler(parser, handler)	Boolean	Registers a character data handler.
xml_set_default_handler(parser, handler)	Boolean	Sets the default handler.
xml_set_element_handler(parser, start_element_handler, end_element_handler)	Boolean	Sets the element handler.
xml_set_external_entity_ref_handler(parser, handler)	Boolean	Sets the external reference handler.
xml_set_notation_decl_handler(parser, handler)	Boolean	Sets the notation declaration handler.
xml_set_processing_instruction_handler(parser, handler)	Boolean	Sets the processing instruction handler.
xml_set_unparsed_entity_decl_handler(parser, handler)	Boolean	Sets the unparsed entity declaration handler.

APPENDIX C

SUPPORT—DEBUGGING APPLICATIONS



In this appendix you will learn how to debug your PHP applications to figure out errors, which is very important. I only call this section *support* because you may have trouble getting some examples to work if your PHP version is different or if you use a special configuration. This appendix will give you the skills necessary to fix errors as they arise. With practice, debugging will become second nature to you.

You can break errors into three broad categories.

- Syntax/compilation errors
- Semantic/runtime errors
- Logic errors

Syntax Errors

Syntax errors are raised by the parser when there is malformed code. These are the easiest errors to track down because the parser tells you which line the error occurred on. Take the following line of code, for example:

```
$x = $point["x";
```

This will produce a parser error that looks like Figure C.1.

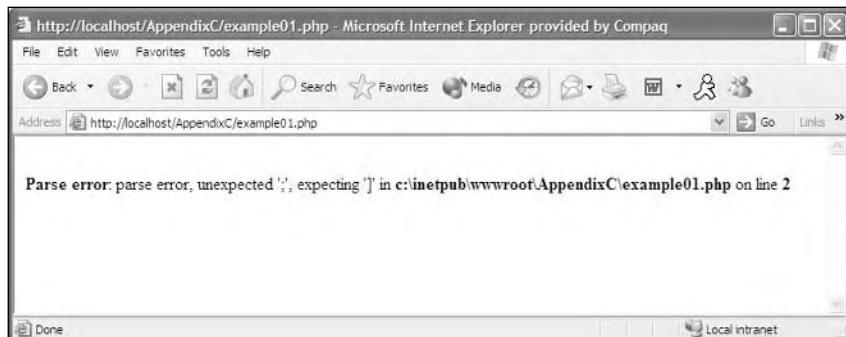


Figure C.1 A parser error.

The error tells you that the parser found an unexpected character, and it tells you the character that the parser was expecting. It even tells you what line the error occurred on. The fix for this error is fairly straightforward:

```
$x = $point["x"];
```

This is straightforward, but what if you did something like this:

```
<?php  
for($i = 0;$i < $count; $i++)  
?>  
<b>Menu Item</b>  
<?php  
echo($i);  
}  
?>
```

This produces the error shown in Figure C.2.

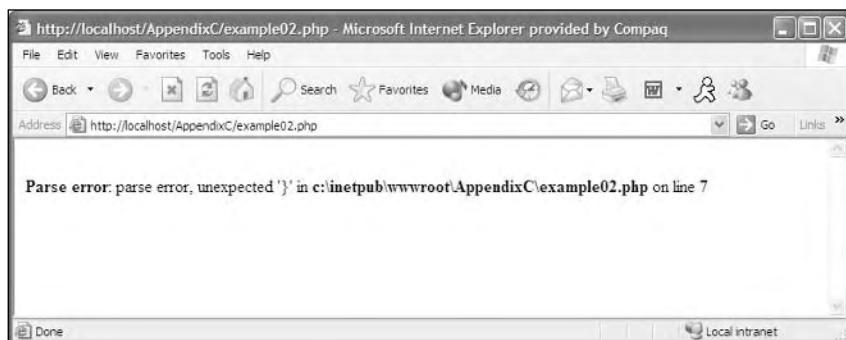


Figure C.2 Another parser error.

The error is pointing to the bracket at the end of the code. But this brace should be there, right? So there must be an error somewhere before line 7. Sure enough, the beginning brace was missed after the start of the for loop. One way to fix this error is to add the brace. Another perfectly legitimate way to fix this error is to remove the ending brace altogether. Remember, a for loop doesn't have to have braces if there is only one line of code to execute.

Semantic Errors

Semantic errors are a bit harder to track down than syntax errors. The errors can be somewhat vague even though they give line numbers. Take a look at the following example:

```
<?php  
socket_create($socket, 'localhost', 'tcp');  
?>
```

This generates the error shown in Figure C.3. This error says that the second parameter is supposed to be a long but a string was given. But the second parameter is perfectly legitimate. It is the third parameter that is incorrect. It should read:

```
socket_create($socket, 'localhost', SOL_TCP);
```

See what I mean when I say that the errors are somewhat vague? These errors are still fairly easy to track down. Usually when you stare at the specified line of code for a moment, the fix dawns on you. If the fix doesn't come to you then go to www.php.net and type in the function name in the search box. This will bring up the documentation for the particular function so you can determine where you went wrong.

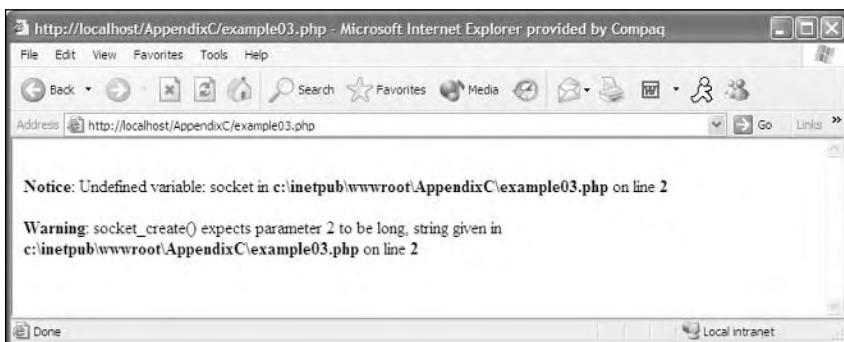


Figure C.3 A semantic error.

Logic Errors

Of the three general categories of errors, logic errors are the hardest to track down. The reason for this is because the code is syntactically correct and runs. The code just doesn't run as expected. The most frustrating part of logic errors is that the code you wrote may look like it is working, but then something unexpected occurs later down the line. Take a quick look at the following code example and see if you can spot the logic error.

```
<?php
    $someArray = array("Knife", "Gun", "Health", "Shield");
    for($loop = 0; $loop < count($someArray); $loop++)
    {
        if($someArray[$loop] == "Health")
        {
            unset($someArray[$loop]);
        }
        echo("$loop = $someArray[$loop]<br>");
    }
?>
```

Did you spot the error? Take a look at Figure C.4 to see the output of this example and see if you can figure out what is happening.

Notice how the output stops at the third element in the array. Shouldn't the last line say 3 = Shield? The error in logic here is that you are iterating through the number of elements in the array. Once you hit the "Health" element you unset the index in the array. This results in the count of the array being decreased by one. So when the loop starts the next set should print out 3 = Shield. The index is now greater than the count of the array.

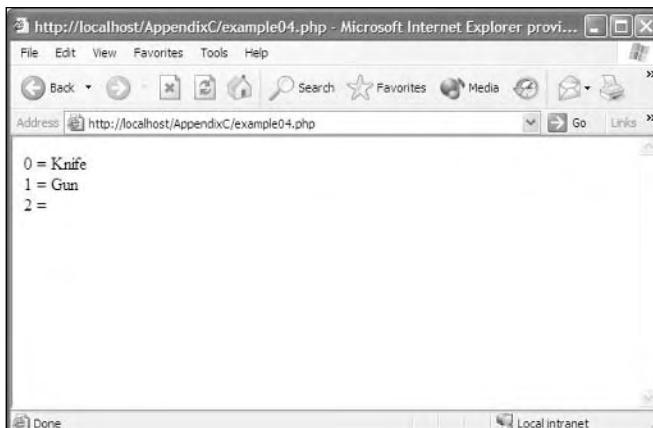


Figure C.4 A logic error.

Notice that there are no errors on this page. As I mentioned, logic errors can be hard to track down when something doesn't work as expected.

There is actually a fourth category, environmental errors, but you have absolutely no control over these. An environmental error is when there is an inherent bug in the program you are using. If you run into an untraceable error, chances are fairly high that it is an environmental error. However, you can't just chalk up hard-to-find logic errors as environmental errors. If you believe you have run into an environmental error go to the Web site and take a look at the forums and the known bug lists. If you find something related to your problem, then there is most likely an easy fix for it. Usually the fix comes in the form of an update or a configuration setting.

PHP and Error Reporting

PHP is capable of four levels of error reporting. PHP reports fatal errors, parser errors, warnings, and notices. You can set your desired level of error reporting in the php.ini file under the errors header. Or you may set the level of error reporting on the fly for your application. To set the error reporting level on the fly you can use the following function:

```
int error_reporting(int level);
```

The `error_reporting()` function takes an integer of the level of reporting you would like. Each of the four levels of errors has an integer value.

- 1 -Fatal Errors
- 2 -Warnings
- 4 -Parser Errors
- 8 -Notices

So to figure out the level of error reporting you would like, you simply add the values together. This is much like the permissions on a UNIX machine. For example, if you would like to report only fatal errors and warnings you would do the following:

```
error_reporting(3);
```

See how that works? Take a look at one more. Let's say you wanted to report only notices, warnings, and fatal errors. You would use the following line:

```
error_reporting(11);
```

By the same token, if you supplied 0 as the value for the level then no errors would be reported. This is a handy function to use. While you are developing an application you can turn on your errors so you can debug problems. But when you put something into production you could simply set the error reporting level to 0 and have no fear that a user will see a big nasty error in the middle of a page.

Handling Errors

Handling errors in PHP is fairly straightforward. Most of the time a function will return 0 if it fails for one reason or another, so to catch this you can always use an if statement.

```
<?php  
if(!someFunction($value))  
{  
    echo("An error has occurred");  
    return;  
}  
?>
```

This allows you to do whatever you want when an error occurs. You could just let users know something has gone wrong, or you could even go as far as to take them to a detailed error page that allows them to e-mail the administrator with a notification of the error.

However, there are some functions that will generate an error if they fail. If this is the case you can suppress the error by using the at (@) symbol.

```
<?php  
if(!@mysql_connect($db, $username, $password))  
{  
    echo("An error has occurred while trying to open the database");  
    return;  
}  
?>
```

This will suppress the PHP error message but the function will still return 0, letting you know that it has failed. Be aware that if you suppress a fatal error no message will be displayed but processing of the script will halt dead in its tracks. So I recommend running your code without the suppression when testing to see where errors occur and adding the suppression in when you are finished debugging your code.

If you suppress error messages you can always retrieve the full error message through the \$php_errormsg variable. This variable will always contain the last error that occurred in the PHP interpreter.

When writing this book I ran into an error that occurred when trying to write to a db file that I created and I had errors turned off. So every time I executed the script nothing would show up on the screen and no file was ever created. It took me a few minutes to realize what was going on. When I finally did, I got an error message that looked like Figure C.5.

I first thought to myself, “What the hell is that; it makes no sense at all.” Even though this error was quite cryptic, it gave me a direction. Since it was having a problem with the dba_open() line, I thought I must not have included the db extensions. When I checked the

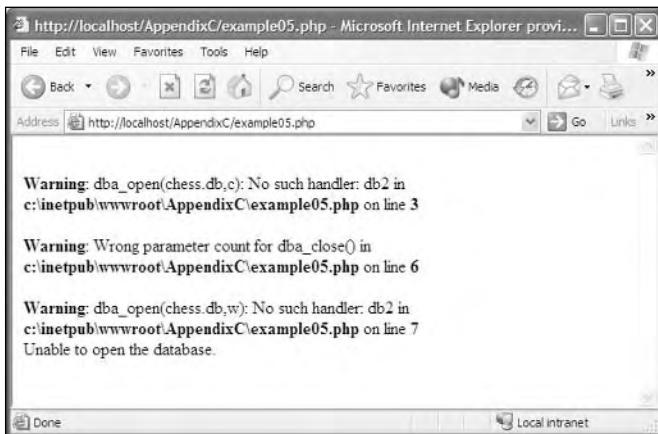


Figure C.5 Error message.

php.ini file I discovered I had included the db extensions. Then, after several hours of searching the forums, I ran the `phpinfo()` function. When I did this I discovered that the version of PHP that I was running did not support the db2 format, but it did support the db3 format. Once I made this simple change everything worked perfectly. It just goes to show that using those errors may not lead you to an immediate fix but it will at least point you in the right direction.

Application and Installation Problems

While installing PHP or one of the other applications included on the CD you may run into some configuration or setup issues. I can't cover all the issues in this book, but I can recommend just going to the manufacturer's Web site and looking at their documentation. These companies have done a great job of detailing very common problems that occur during installation and setup. If your problem is not in the available documentation, there is always customer support.

The one thing that I am going to include in this book is the set of frequently asked questions listed on www.php.net. I imagine these will be the most common problems you will run into and I want to save you the time of trying to find these answers all on your own.

1. I got the latest version of PHP using the anonymous CVS service, but there's no configure script!

You must have the GNU autoconf package installed so you can generate the configure script from `configure.in`. Just run `./buildconf` in the top-level directory after getting the sources from the CVS server. (Also, unless you run `configure` with the `--enable-maintainer-mode` option, the `configure` script will not automatically get

rebuilt when the configure.in file is updated, so you should make sure to do that manually when you notice configure.in has changed. One symptom of this is finding things like @VARIABLE@ in your Makefile after configure or config.status is run.)

2. I'm having problems configuring PHP to work with Apache. It says it can't find httpd.h, but it's right where I said it is!

You need to tell the configure/setup script the location of the top level of your Apache source tree. This means that you want to specify --with-apache=/path/to/apache and not --with-apache=/path/to/apache/src.

3. While configuring PHP, you come across an error similar to the following:

```
checking lex output file root... ./configure: lex: command not found  
configure: error: cannot find output from lex; giving up
```

Be sure to read the installation instructions carefully and note that you need both Flex and Bison installed to compile PHP. Depending on your setup, you will install Bison and Flex from either source or a package, such as an RPM.

4. When I try to start Apache, I get the the following message:

```
fatal: relocation error: file /path/to/libphp4.so:  
symbol ap_block_alarms: referenced symbol not found
```

This error usually comes up when you compile the Apache core program as a DSO library for shared usage. Try to reconfigure Apache, making sure to use at least the following flags:

```
--enable-shared=max --enable-rule=SHARED_CORE
```

For more information, read the top-level Apache install file.

5. When I run configure, it says that it can't find the include files or library for GD, gdbm, or some other package!

You can make the configure script look for header files and libraries in non-standard locations by specifying additional flags to pass to the C preprocessor and linker, such as:

```
CPPFLAGS=-I/path/to/include LDFLAGS=-L/path/to/library ./configure
```

If you're using a csh-variant for your login shell (why?), it would be:

```
env CPPFLAGS=-I/path/to/include LDFLAGS=-L/path/to/library ./configure
```

6. When it is compiling the file language-parser.tab.c, it gives me errors that say yytname undeclared.

You need to update your version of Bison. You can find the latest version at www.gnu.org/software/bison/bison.html.

7. When I run make, it seems to run fine but then fails when it tries to link the final application, complaining that it can't find some files.

Some old versions of make don't correctly put the compiled versions of the files in the functions directory into that same directory. Try running `cp *.o functions` and then re-running make to see if that helps. If it does, you should really upgrade to a recent version of GNU make.

8. When linking PHP, it complains about a number of undefined references.

Take a look at the link line and make sure that all of the appropriate libraries are being included at the end. Common ones that you might have missed are 'ldl' and any libraries required for any database support you included.

If you're linking with Apache 1.2.x, did you remember to add the appropriate information to the EXTRA_LIBS line of the configuration file and re-rerun Apache's configure script? See the install file that comes with the distribution for more information.

Some people have also reported that they had to add 'ldl' immediately following libphp4.a when linking with Apache.

9. I can't figure out how to build PHP with Apache 1.3.

This is actually quite easy. Follow these steps carefully:

1. Grab the latest Apache 1.3 distribution from www.apache.org/dist/httpd/.
2. Ungzip and untar it somewhere, for example /usr/local/src/apache-1.3.
3. Compile PHP by first running `./configure --with-apache=<path>/apache-1.3` (substitute <path> with the actual path to your Apache-1.3 directory).
4. Type `make` followed by `make install` to build PHP and copy the necessary files to the Apache distribution tree.
5. Change directories into to your /<path>/apache-1.3/src directory and edit the configuration file. Add to the file: `AddModule modules/php4/libphp4.a`.
6. Type: `./configure` followed by `make`. You should now have a PHP-enabled httpd binary!

Note

You can also use the new Apache `./configure` script. See the instructions in the README.configure file, which is part of your Apache distribution. Also have a look at the install file in the PHP distribution.

10. I have followed all the steps to install the Apache module version on UNIX, and my PHP scripts show up in my browser or I am being asked to save the file.

This means that the PHP module is not getting invoked for some reason. Three things to check before asking for further help:

Make sure that the httpd binary you are running is the actual new httpd binary you just built. To do this, try running: /path/to/binary/httpd -l .

If you don't see mod_php4.c listed then you are not running the right binary. Find and install the correct binary.

1. Make sure you have added the correct Mime Type to one of your Apache .conf files. It should be: AddType application/x-htpd-php3 .php3 (for PHP 3) or AddType application/x-htpd-php .php (for PHP 4)
 2. Also make sure that this AddType line is not hidden away inside a <Virtualhost> or <Directory> block which would prevent it from applying to the location of your test script.
 3. Finally, the default location of the Apache configuration files changed between Apache 1.2 and Apache 1.3. You should check to make sure that the configuration file you are adding the AddType line to is actually being read. You can put an obvious syntax error into your httpd.conf file or some other obvious change that will tell you if the file is being read correctly.
- 11. It says to use --activate-module=src/modules/php4/libphp4.a, but that file doesn't exist, so I changed it to --activate-module=src/modules/php4/libmodphp4.a and it doesn't work! What's going on?**
- Note that the libphp4.a file is not supposed to exist. The Apache process will create it!
- 12. When I try to build Apache with PHP as a static module using --activate-module=src/modules/php4/libphp4.a it tells me that my compiler is not ANSI-compliant.**
- This is a misleading error message from Apache that has been fixed in more recent versions.
- 13. When I try to build PHP using --with-apxs I get strange error messages.**

There are three things to check here. First, for some reason when Apache builds the apxs Perl script, it sometimes ends up getting built without the proper compiler and flags variables. Find your apxs script (try the command `which apxs`). It's sometimes found in /usr/local/apache/bin/apxs or /usr/sbin/apxs. Open it and check for lines similar to these:

```
my $CFG_CFLAGS_SHLIB = ' ' ;           # substituted via Makefile.tmp1  
my $CFG_LD_SHLIB    = ' ' ;           # substituted via Makefile.tmp1  
my $CFG_LDFLAGS_SHLIB = ' ' ;          # substituted via Makefile.tmp1
```

If this is what you see, you have found your problem. They may contain just spaces or other incorrect values, such as '`q()`'. Change these lines to say:

```
my $CFG_CFLAGS_SHLIB = '-fpic -DSHARED_MODULE'; # substituted via Makefile tmpl
my $CFG_LD_SHLIB     = 'gcc';                  # substituted via Makefile tmpl
my $CFG_LDFLAGS_SHLIB = q(-shared);           # substituted via Makefile tmpl
```

The second possible problem should only be an issue on Red Hat 6.1 and 6.2. The apxs script Red Hat ships is broken. Look for this line:

```
my $CFG_LIBEXECDIR = 'modules'; # substituted via APACI install
```

If you see the above line, change it to this:

```
my $CFG_LIBEXECDIR = '/usr/lib/apache'; # substituted via APACI install
```

Last, if you reconfigure/reinstall Apache, add a make clean to the process after ./configure and before make.

14. During make, I get errors in microtime, and a lot of RUSAGE_ stuff.

During the make portion of installation if you encounter problems that look similar to this:

```
microtime.c: In function `php_if_getrusage':
microtime.c:94: storage size of `usg' isn't known
microtime.c:97: `RUSAGE_SELF' undeclared (first use in this function)
microtime.c:97: (Each undeclared identifier is reported only once
microtime.c:97: for each function it appears in.)
microtime.c:103: `RUSAGE_CHILDREN' undeclared (first use in this function)
make[3]: *** [microtime.lo] Error 1
make[3]: Leaving directory `/home/master/php-4.0.1/ext/standard'
make[2]: *** [all-recurse] Error 1
make[2]: Leaving directory `/home/master/php-4.0.1/ext/standard'
make[1]: *** [all-recurse] Error 1
make[1]: Leaving directory `/home/master/php-4.0.1/ext'
make: *** [all-recurse] Error 1
```

then your system is broken. You need to fix your /usr/include files by installing a glibc-devel package that matches your glibc. This has absolutely nothing to do with PHP. To prove this to yourself, try this simple test:

```
$ cat >test.c <<X
#include <sys/resource.h>
X
$ gcc -E test.c >/dev/null
```

If that spews out errors, you know your include files are messed up.

- 15. When compiling PHP with MySQL, configure runs fine but during make I get an error similar to the following: ext/mysql/libmysql/my_tempnam.o(.text+0x46): In function my_tempnam': /php4/ext/mysql/libmysql/my_tempnam.c:103: The use of tempnam' is dangerous, better use mkstemp', what's wrong?**

First, it's important to realize that this is a warning and not a fatal error. Because this is often the last output seen during make it may seem like a fatal error, but it's not. Of course, if you set your compiler to die on Warnings, it will. Also keep in mind that MySQL support is enabled by default.

Note

As of PHP 4.3.2, you'll also see the following text after the build (make) completes:

Build complete (it is safe to ignore warnings about tempnam and tmpnam).

- 16. I want to upgrade my PHP. Where can I find the ./configure line that was used to build my current PHP installation?**

Either you look at config.nice file, in the source tree of your current PHP installation, or, if this is not available, you simply run the `php_info()` function. On the top of the output the `./configure` line that was used to build this PHP installation is shown.

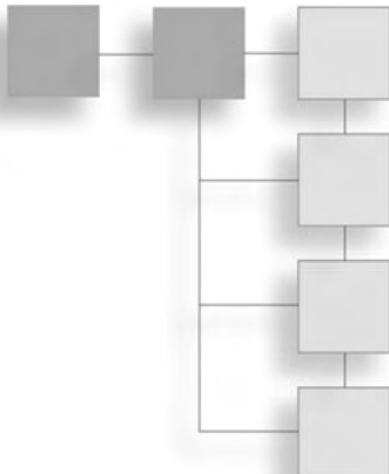
- 17. When building PHP with the GD library it either gives strange compile errors or segfaults on execution.**

Make sure your GD library and PHP are linked against the same depending libraries (e.g., libpng).

That is the complete Frequently Asked Questions section from www.php.net. Now you don't have to fumble around on the Internet. You can just go straight to this appendix to find most of your configuration and installation answers.

APPENDIX D

GD SDK LANGUAGE REFERENCE



This appendix is the language reference for Boutell's GD graphics library. Each function is listed with the parameters it receives, what the function returns, and a short description of what the function does.

GD Functions

Function	Returns	Description
GetImageSize(filename, [image_info])	Array	Gets the size of the image.
Image2wbmp(image, [filename], [threshold])	Integer	Saves the image as a WBMP.
ImageAlphaBlending(image, blendmode)	Integer	Switches alpha blending on or off by setting blendmode to true or false, respectively.
ImageArc(image, cx, cy, width, height, start, end, col)	Integer	Draws a partial ellipse in the specified image. The center point of the arc is at cx, cy.
ImageChar(image, font, x, y, c, col)	Integer	Draws a character horizontally on the specified image.
ImageCharUp(image, font, x, y, c, col)	Integer	Draws a character vertically on the specified image.
ImageColorAllocate(img, red, green, blue)	Integer	Allocates the specified RGB color value for the image.
ImageColorAt(img, x, y)	Integer	Returns the index of the color at x, y.
ImageColorClosest(img, red, green, blue)	Integer	Returns the index to the closest color in the color palette for the image.
ImageColorClosestAlpha(image, red, green, blue, alpha)	Integer	Returns the index of the closest color to the specified color plus the alpha palette for the specified image.
ImageColorClosestHwb(image, red, green, blue)	Integer	Returns the index of the color with the hue, white, and blackness closest to the RGB value specified.

Function	Returns	Description
ImageColorDeallocate(image, index)	Integer	Deallocates a color in the specified image.
ImageColorExact(img, red, green, blue)	Integer	Returns the index of the specified color in the color palette of the image.
ImageColorExactAlpha(image, red, green, blue, alpha)	Integer	Returns the index of the specified color in the color palette plus the alpha channel for the image.
ImageColorResolve(img, red, green, blue)	Integer	Finds the specified color in the palette. If it doesn't exist, it returns the index to the closest color in the palette.
ImageColorResolveAlpha(img, red, green, blue, alpha)	Integer	Finds the specified color in the palette. If it doesn't exist, it returns the index to the closest color in the palette plus the images alpha channel.
ImageColorSet(img, index, red, green, blue)	Boolean	Sets the specified index to the RGB color value.
ImageColorsForIndex(img, index)	Array	Returns an array containing the RGB values for the specified index in the color palette.
ImageColorsTotal(img)	Integer	Returns the number of colors in the specified images color palette.
ImageTrueColorToPalette(image, dither, colors)	void	Converts a true-color image into a palletized image.
ImageColorTransparent(img, [color])	Integer	Sets the transparent color in the palette.
ImageCopy(dest_image, src_image, dest_x, dest_y, src_x, src_y, src_w, src_h)	Integer	Copies an area from the source image to the destination image.
ImageCopyMerge(dest_image, src_image, dest_x, dest_y, src_x, src_y, src_w, src_h, pct)	Integer	Copies an area from the source image to the destination image. The amount of the merge is specified by the value pct (0–100).
ImageCopyMergeGray(dest_image, src_image, dest_x, dest_y, src_x, src_y, src_w, src_h, pct)	Integer	Works exactly like <code>ImageCopyMerge()</code> but the area being copied is first converted to grayscale.
ImageCopyResampled(dest_image, src_image, dest_x, dest_y, src_x, src_y, dest_w, dest_h, src_w, src_h)	Integer	Copies and resamples the area from the source image.
ImageCopyResized(dest_img, src_img, destX, destY, srcX, srcY, destWidth, destHeight, srcWidth, srcHeight)	Integer	Copies an area from the source image to an array of the destination image. If the heights are different then the destination image is resized.
ImageCreate(width, height)	Integer	Creates a new image.
ImageCreateFromGif(filename)	Integer	Creates a new image from the specified file.
ImageCreateFromJpeg(filename)	Integer	Creates a new image from the specified file.
ImageCreateFromPng(filename)	Integer	Creates a new image from the specified file.
ImageCreateFromString(string)	Integer	Creates a new image from the data in the string.
ImageCreateFromWbmp(filename)	Integer	Creates a new image from the file.
ImageCreateFromXbm(filename)	Integer	Creates a new image from the file.
ImageCreateFromXpm(filename)	Integer	Creates a new image from the file.
ImageCreateTrueColor(width, height)	Integer	Creates a new true color image of width by height.
ImageDashedLine(img, x1, y1, x2, y2, color)	Integer	Draws a dashed line in the image from point x1, y1 to point x2, y2.

Function	Returns	Description
ImageDestroy(img)	Integer	Destroys the specified image.
ImageEllipse(image, x, y, width, height, color)	Integer	Draws an ellipse with the center point at x, y.
ImageFill(img, x, y, color)	Integer	Fills the image starting at point x, y.
ImageFilledArc(image, x, y, width, height, start, end, color, style)	Integer	Draws a filled-in arc centered at point x, y.
ImageFilledEllipse(image, x, y, width, height, color)	Integer	Draws a filled ellipse in the image with its center point at x, y.
ImageFilledPolygon(img, points, num_points, color)	Integer	Draws a filled polygon in the image between the points.
ImageFilledRectangle(img, x1, y1, x2, y2, color)	Integer	Draws a filled rectangle in the specified image.
ImageFillToBorder(img, x, y, border, color)	Integer	Performs a flood-fill between the specified border color starting at point x, y.
ImageFontHeight(font)	Integer	Returns the height of the specified font in pixels.
ImageFontWidth(font)	Integer	Returns the width of the specified font in pixels.
ImageGammaCorrect(image, inputgamma, outputgamma)	Integer	Corrects the gamma to the specified image.
ImageGif(img, [filename])	Integer	Sends the image to a file or to the browser.
ImageInterlace(img, [interlace])	Integer	Turns interlacing on or off in the image.
ImageJpeg(image, [filename], [quality])	Integer	Sends the image to a file or to the browser.
ImageLine(img, x1, y1, x2, y2, color)	Integer	Draws a line in the image from point x1, y1 to point x2, y2.
ImagePng(image, [filename])	Integer	Sends the image to a file or to the browser.
ImageSetPixel(image, x, y, color)	Integer	Draws a pixel at point x, y in the specified color.
ImagePolygon(img, points, num_points, color)	Integer	Draws a polygon in the image, much like polygon fill except it doesn't fill the image.
ImageRectangle(img, x1, y1, x2, y2, color)	Integer	Draws a rectangle to the image. Acts much like ImageRectangleFill() but the rectangle is not filled when it is drawn.
ImageString(img, font, x, y, string, color)	Integer	Draws a string starting at x, y to the image.
ImageTTFString(image, size, angle, x, y, color, fontfile, text)	Array	Draws the text starting at point x, y in the specified TrueType font.
ImageTTFText(image, size, angle, x, y, color, fontfile, text)	Array	Draws the text starting at point x, y in the specified TrueType font.
ImageWbmp(image, [filename], [foreground])	Integer	Creates a WBMP file from the specified image.

INDEX

A

ActionScript

Flash movies, 275-278
Ming, 258
add() function, 263
addEntry() function, 267-268
addFill() function, 265-270
algorithms, TCP/IP, 14
animating Flash movies, 271-275
Apache servers, installing, 18-21
arcs, drawing, 174-178
arguments, padding, 64-65
arithmetic operators, 78
array() function, 105
arrays
 Chess game board, 140-141
 declaring, 104-105
 for statements, 106-108
 indexes, 104-106
 initializing, 104-105
 looping, 106-110
 multi-dimensional, 110-112
 overview, 103-104
 sorting, 112-117
 strings, 105-106
 tic-tac-toe, 117-132
 while statements, 108-110
arsort() function, 115
asort() function, 115
assigning color, images, 161-162
assignment shortcuts, variables, 85-86

B

background

Battle Tank, 196, 205-208
Flash movies, 260
Battle Tank
 background, 196, 205-208
 bullets, 198-199, 204
 coordinates, 205-208
 CSS, 196-197
 globals, 198-199
 gravity, 198-199, 204
 images, 195-198, 201, 205-208
 integrating sockets, 224-228
 interface, 197-198
 loops, 201-205
 menu, 199-200
 overview, 193-195
 states, 194-195, 198-199
 tables, 197-198
 tanks, 196-197
 variables, 199-200, 204

binary operators

, 79-80
binding sockets, 215-216

bitmaps, Flash movies, 269-270

bits. *See operators*

bitwise operators, 82-85

blocks, code, 56

boards

 Chess, 140-141
 printing, 111-112

<BODY> HTML tag, 33-36

Boolean data type, 57

**
 HTML tag**, 36

browsers. *See clients*

bullets, Battle Tank, 198-199, 204

C

CAddWeaponsCmd() class, 245-248

Cartesian system. *See coordinates*

case sensitivity, variables, 57

CCommand() class, 235-240

CCreateAccountCmd() class, 240-245

CGame() class, 235, 251-255

CheckFull() function, 126-128

CheckWin() function, 126-128

Chess game

 database, 140-155

 defining globals, 141-142

 DrawBoard() function, 147-155

 GetSquare() function, 153

 move history, 154-155

 MovePiece() function, 147

 pieces, 143-151

 ProcessInput() function, 151-155

 PutPiece() function, 147

 render() function, 143-151

 StartBoard() function, 147

 states, 141-142

 TakePiece() function, 147

 UpdateMoveList() function, 153-154

 user input, 151-155

classes. *See also functions*

CAddWeaponsCmd(), 245-248

CCommand(), 235-240

CCreateAccountCmd(), 240-245

CGame(), 235, 251-255

CLockTableCmd(), 243

CommandFactory(), 248-255

CRenderCmd(), 243-244

CUnlockTableCmd(), 243

CVerifyAccountInfoCmd(), 243

Ming, 258-259

SWFAction(), 276-278

SWFBitmap(), 269-270

SWFDisplayItem(), 271-275

SWFFill(), 265-270

SWFGradient(), 267-268

SWFMorph(), 273-275

SWFMovie(), 259-261

SWFShape(), 262, 269-270

clients

 creating, 223-224

 pages, interpreting, 4

 server relationship, 3-5

 sockets, 213

CLockTableCmd() class, 243

code blocks, 56

color

 Flash movies, 260

 images, 166

 assigning, 161-162

 converting, 165

 filling, 162-163

 number, 165-166

 overview, 161

 transparency. *See transparency*

command factory, creating, 248-255

command line, running scripts, 220

CommandFactory() class, 248-255

commands

 command line, running scripts, 220

 creating

 command factory, 248-255

 Kiddy Cartel, 234-255

 sub-commands, 240-245

 su, 26

 tables, 243

comparison operators, 78-79

ComputerMove() function, 128-131

ComputerRandomMove() function, 128-131

concatenating operators, 81

conditional logic. *See statements*

connecting

 Kiddy Cartel database, 232

 servers, 220-223

constants

 defining, 118

 overview, 58-59

converting color, 165

cookies, passing, 6

coordinates

 Battle Tank, 205-208

 images, drawing, 167

copying images, 185-191

count() function, 107

CREATE TABLE function, 231-232

- creating**
 - clients, 223-224
 - commands
 - command factory, 248-255
 - Kiddy Cartel, 234-255
 - sub-commands, 240-245
 - databases, 134-136
 - directories, 117
 - error messages, 95-98
 - Flash movies, 259-261
 - images, 160-161
 - pages, 56
 - servers, 220-223
 - sockets, 215
 - tables, 231-232
 - CRenderCmd() class**, 243-244
 - CSS, Battle Tank**, 196-197
 - CUnlockTableCmd() class**, 243
 - current() function**, 109-110
 - CVerifyAccountInfoCmd() class**, 243
- D**
- data types**
 - arrays
 - Chess game board, 140-141
 - declaring, 104-105
 - for statements, 106-108
 - indexes, 104-106
 - initializing, 104-105
 - looping, 106-110
 - multi-dimensional, 110-112
 - overview, 103-104
 - sorting, 112-117
 - strings, 105-106
 - tic-tac-toe, 117-132
 - while statements, 108-110
 - Boolean, 57
 - declaring, 57
 - Flash movies, 275-278
 - strings
 - array indexes, 105-106
 - formatting numbers, 65
 - functions, 61-65
 - operators. *See operators*
 - padding arguments, 64-65
 - pattern matching, 66-71
 - type casting, 58
 - type juggling, 57-58
 - variables
 - Battle Tank, 199-200, 204
 - case sensitivity, 57
 - constants, 58-59
 - declaring, 57
 - dynamic values, 58
 - functions, 60-61
 - naming, 59-60
 - operators. *See operators*
 - session variables, 5-13
 - shortcut operators, 85-86
 - type casting, 58
 - type juggling, 57-58
 - variable, 58
 - database abstraction (DBA)**, 134
 - databases**
 - Chess game, 140-155
 - connecting, 232
 - creating, 134-136
 - DBA, 134
 - entries
 - deleting, 140
 - inserting, 137-139
 - updating, 139-140
 - Kiddy Cartel, 231-232
 - looping, 136-137
 - MySQL
 - installing, 230-231
 - PEAR, 229
 - non-relational, 134
 - object oriented, 133
 - opening, 134-136
 - overview, 133-134
 - relational, 133, 231-233
 - DBA (database abstraction)**, 134
 - dba_close() function**, 140
 - dba_delete() function**, 140
 - dba_fetch() function**, 136-137
 - dba_firstkey() function**, 136-137
 - dba_insert() function**, 137-139
 - dba_open() function**, 134-136
 - dba_replace() function**, 139
 - dba_sync() function**, 139-140
 - debugging**
 - games, 155
 - installing, 29-30
 - semicolons, 56

server states, 5
declaring
 arrays, 104-105
 data types, 57
 functions, 95
 variables, 57
define() function, 59
defined() function, 59
defining
 constants, 118
 globals
 Battle Tank, 198-199
 Chess game, 141-142
 tic-tac-toe, 118
DELETE FROM function, 233
deleting
 database entries, 140
 records, 233
delimiters, selecting, 56
directories, creating, 117
<DIV> HTML tag, 197
dl() function, 159
do statements, 93-94
documents. See files
DrawBoard() function
 Chess game, 147-155
 tic-tac-toe, 125-126
drawCurve() function, 263, 264
drawCurveTo() function, 264-265
drawing. See also images
 Flash movies, 262-265
 images, 167
 arcs, 174-178
 coordinates, 167
 ellipses, 174-178
 lines, 167-169
 pixels, 167-169
 polygons, 172-174
 rectangles, 169-172
drawLine() function, 262-263
drawLineTo() function, 262-263
dynamic values, variables, 58

E

each() function, 108-110
ellipses, drawing, 174-178
else statements, 88-90

empty() function, 61
enabling sockets, 219
EndGame() function, 124-125, 199-200
entries, databases
 deleting, 140
 inserting, 137-139
 updating, 139-140
ereg() function, 69-70
ereg_replace() function, 70
ereg_replace() function, 70
error messages, creating, 95-98
Execute() function, 239, 243-248
expressions
 functions, 69-71
 pattern matching, 66-69

F

files
 HTML
 formats, 37-38
 overview, 32-33
 including, 99
 make, 24
filling
 color, images, 162-163
 objects, Flash movies, 265-270
Flash
 Ming
 ActionScript, 258
 classes, 258-259
 installing, 258
 overview, 257-259
 movies
 ActionScript, 275-278
 animating, 271-275
 background color, 260
 bitmaps, 269-270
 creating, 259-261
 data types, 275-278
 drawing, 262-265
 filling objects, 265-270
 images, 262-265
 morphing, 273-275
 printing, 260
 size, 259-261
 speed, 260

fopen() function, 269-270
for statements. *See loops*
<FORM> HTML tag, 48-51
formatting
 numbers, strings, 65
 text, HTML tags, 35
forms
 HTML, 48-51
 processing, 71-75
fread() function, 269-270
functions. *See also classes*
 add(), 263
 addEntry(), 267-268
 addFill(), 265-270
 array(), 105
 arrays, sorting, 112-117
 arsort(), 115
 asort(), 115
 CheckFull(), 126-128
 CheckWin(), 126-128
 ComputerMove(), 128-131
 ComputerRandomMove(), 128-131
 count(), 107
 CREATE TABLE, 231-232
 current(), 109-110
 dba_close(), 140
 dba_delete(), 140
 dba_fetch(), 136-137
 dba_firstkey(), 136-137
 dba_insert(), 137-139
 dba_open(), 134-136
 dba_replace(), 139
 dba_sync(), 139-140
 declaring, 95
 define(), 59
 defined(), 59
 DELETE FROM, 233
 dl(), 159
 DrawBoard()
 Chess game, 147-155
 tic-tac-toe, 125-126
 drawCurve(), 263-264
 drawCurveTo(), 264-265
 drawLine(), 262-263
 drawLineTo(), 262-263
 each(), 108-110
 empty(), 61
 EndGame(), 124-125, 199-200
 ereg(), 69-70
 ereg_replace(), 70
 ereg_i(), 70
 ereg_i_replace(), 70
 Execute(), 239, 243-248
 fopen(), 269-270
 fread(), 269-270
 GameInit(), 200-201, 206-207
 getShape1(), 273-275
 getShape2(), 273-275
 GetSquare(), 153
 gettext(), 60
 HandleSubCommand(), 239
 ImageArc(), 174-178
 ImageColorAllocate(), 161-162
 ImageColorAt(), 166
 ImageColorSet(), 166
 ImageColorsTotal(), 165-166
 ImageColorTransparent(), 163-165
 ImageCopy(), 186-187
 ImageCopyMerge(), 189-191
 ImageCopyResampled(), 188-189
 ImageCopyResized(), 188-189
 ImageCreate(), 160-161
 ImageCreateFromJpeg(), 185
 ImageCreateFromPng(), 185
 ImageCreateTrueColor(), 160-161
 ImageFill(), 162-163
 ImageFilledArc(), 176-178
 ImageFilledPolygon(), 174, 207-208
 ImageFilledRectangle(), 171-172
 ImageLine(), 168-169
 ImagePolygon(), 172-174
 ImageRectangle(), 169-172
 ImageSetPixel(), 167-169
 ImageString(), 179-180
 ImageTrueColorToPalette(), 165
 ImageTTFBox(), 182-184
 ImageTTFText(), 180-183
 ImageWbmp(), 184-185
 include(), 99
 ini_set(), 7-13
 INSERT INTO, 233
 is_datatype(), 61
 isset(), 60
 key(), 109-110
 krsort(), 116
 ksort(), 115

list(), 108-110
 move(), 271-275
 movePenTo(), 262-263
 MovePiece(), 147
 moveTo(), 271-275
 MyFilledPolygon(), 174
 MyFilledRectangle(), 171-172
 MyPolygon(), 173-174
 MyRectangle(), 170
 MyTTFBox(), 183-184
 next(), 109-110
 nextFrame(), 272-273
 number_format(), 65
 OnError(), 239
 OnRollBack(), 239-248
 output(), 260
 overview, 94-95
 parameters, 95-98
 phpinfo(), 159, 219
 prev(), 109-110
 printf(), 64-65
 ProcessInput(), 151-155
 PutPiece(), 147
 Query(), 248
 recursion, 98-99
 regular expressions, 69-71
 render()
 Chess game, 143-151
 tic-tac-toe, 119-123
 Render(), 201-205, 225-228
 RenderInterface(), 197-198
 RenderTanks(), 196-197
 RenderTerrain(), 196
 reset(), 108-110
 rotate(), 271-275
 rsort(), 115
 serialize(), 137-139
 sessions, 10-13
 setBackground(), 260
 setDimension(), 259-261
 setLeftFill(), 265-270
 setLine(), 262-265
 setRate(), 260
 setRightFill(), 265-270
 settype(), 60
 socket_bind(), 215-216
 socket_create(), 215
 socket_listen(), 215-216

sockets, 216-219
 sort(), 113-114
 split(), 71
 sprintf(), 64-65
 StartBoard(), 147
 StartGame(), 124-125, 199-200
 strings, 61-65
 TakePiece(), 147
 tic-tac-toe, 123-131
 unset(), 61
 UPDATE, 233
 UpdateMoveList(), 153-154
 usort(), 116-117
 variables, 60-61

G

GameInit() function, 200-201, 206-207
games

Battle Tank
 background, 196, 205-208
 bullets, 198-199, 204
 coordinates, 205-208
 CSS, 196-197
 globals, 198-199
 gravity, 198-199, 204
 images, 195-198, 201, 205-208
 integrating sockets, 224-228
 interface, 197-198
 loops, 201-205
 menu, 199-200
 overview, 193-195
 states, 194-199
 tables, 197-198
 tanks, 196-197
 variables, 199-200, 204

boards

Chess, 140-141
 printing, 111-112

Chess

boards, 140-141
 database, 140-155
 defining globals, 141-142
 DrawBoard() function, 147-155
 GetSquare() function, 153
 move history, 154-155
 MovePiece() function, 147
 pieces, 143-151
 ProcessInput() function, 151-155

games (*continued*)

PutPiece() function, 147
 render() function, 143-151
 StartBoard() function, 147
 states, 141-142
 TakePiece() function, 147
 UpdateMoveList() function, 153-154
 user input, 151-155
 debugging, 155
Kiddy Cartel
 command factory, 248-255
 commands, 234-255
 database, 231-232
 overview, 229-230
 rules, 233-235
 specifications, 233-235
 sub-commands, 240-245
 tables, 233

MMO. *See* Kiddy Cartel

tic-tac-toe
 arrays, 117-132
 CheckFull() function, 126-128
 CheckWin() function, 126-128
 ComputerMove() function, 128-131
 ComputerRandomMove() function, 128-131
 defining constants, 118
 defining globals, 118
 directory, 117
 DrawBoard() function, 125-126
 EndGame() function, 124-125
 functions, 123-131
 HTML, 118-119
 render() function, 119-123
 StartGame() function, 124-125
 states, 119
 switch statement, 123

GD, 157-159

GemStone Web site, 133
GET method, 48, 71-75
getShape1() function, 273-275
getShape2() function, 273-275
GetSquare() function, 153
gettype() function, 60

gifs

HTML, 37-38

support, 157

globals

Battle Tank, 198-199

Chess game, 141-142

tic-tac-toe, 118

graphics. *See* images

gravity, 198-199, 204

H

HandleSubCommand() function, 239

<HEAD> HTML tag, 35

history, move, 154-155

HTML

documents, 32-33
 forms, 48-51, 71-75
 GET method, 48, 71-75
 images, 36-41, 44-48
 file formats, 37-38
 size, 38

transparency, 37

layouts, 44-48

PHP

code blocks, 56
 comparison, 55
 POST method, 48, 71-75
 tables, 41-48

tags

<BODY>, 33-36

, 36
 <DIV>, 197
 <FORM>, 48-51
 <HEAD>, 35
 <IFRAME>, 154-155
 , 39-41
 <INPUT>, 48-51
 <SELECT>, 49-51
 <TABLE>, 41-48
 <TD>, 42-48
 <TEXTAREA>, 49-51
 <TITLE>, 35
 <TR>, 42-48
 overview, 31-32
 printing, 56
 text formatting, 35
 tic-tac-toe, 118-119

I

if statements, 88-90

<IFRAME> HTML tag, 154-155

IIS (Internet Information Server), 14

- installing, 14-18

- PWS comparison, 14-15

ImageArc() function, 174-178**ImageColorAllocate() function,** 161-162**ImageColorAt() function,** 166**ImageColorSet() function,** 166**ImageColorsTotal() function,** 165-166**ImageColorTransparent() function,** 163-165**ImageCopy() function,** 186-187**ImageCopyMerge() function,** 189-191**ImageCopyResampled() function,** 188-189**ImageCopyResized() function,** 188-189**ImageCreate() function,** 160-161**ImageCreateFromJpeg() function,** 185**ImageCreateFromPng() function,** 185**ImageCreateTrueColor() function,** 160-161**ImageFill() function,** 162-163**ImageFilledArc() function,** 176-178**ImageFilledPolygon() function,** 174, 207-208**ImageFilledRectangle() function,** 171-172**ImageLine() function,** 168-169**ImagePolygon() function,** 172-174**ImageRectangle() function,** 169-172**images.** *See also* drawing

- Battle Tank, 195-198, 201, 205-208

- bullets, 198-199, 204

- color, 166

- assigning, 161-162

- converting, 165

- filling, 162-163

- number, 165-166

- overview, 161

- copying, 185-191

- creating, 160-161

- drawing, 167

- arcs, 174-178

- coordinates, 167

- ellipses, 174-178

- lines, 167-169

- pixels, 167-169

- polygons, 172-174

- rectangles, 169-172

- Flash movies, 262-265

- animating, 271-275

- bitmaps, 269-270

- filling objects, 265-270

- morphing, 273-275

- GD, 157-158

- gifs

- HTML, 37-38

- support, 157

- HTML, 36-48

- file formats, 37-38

- size, 38

- transparency, 37

- opacity, 189-191

- saving, 184-185

- size, 187-189

- text, 179-184

- translucency, 189-191

- transparency, 163-165

ImageSetPixel() function, 167-169**ImageString() function,** 179-180**ImageTrueColorToPalette() function,** 165**ImageTTFBox() function,** 182-184**ImageTTFFText() function,** 180-183**ImageWbmp() function,** 184-185

- HTML tag, 39-41

- include()** function, 99

- including files, 99

- indexes, arrays, 104-106

- ini_set()** function, 7-13

- initializing arrays, 104-105

- input**, users, 151-155

- <INPUT> HTML tag, 48-51

INSERT INTO function, 233**inserting**

- database entries, 137-139

- records, 233

installing

- GD, 158-159

- IIS, 14-18

- Ming, 258

- MySQL, 230-231

- PWS, 14-18

- servers

- Apache, 18-21

- IIS, 14-18

- PWS, 14-18

- UNIX, 19-21

- testing, 29-30

- UNIX, 24-26

- Windows, 26-29

interfaces, Battle Tank, 197-198**Internet Information Server.** *See IIS*

interpreting pages, browsers, 4
is_datatype() function, 61
isset() function, 60

J

jpg files, HTML, 37-38

K

key() function, 109-110
keyword, PRIMARY KEY, 232
Kiddy Cartel
 commands
 command factory, 248-255
 creating, 234-255
 sub-commands, 240-245
 database, 231-232
 overview, 229-230
 rules, 233-235
 specifications, 233-235
 tables, 233
krsort() function, 116
ksort() function, 115

L

layers, TCP/IP, 13-14
layouts, HTML, 44-48
libraries
 GD, 157-159
 Ming
 ActionScript, 258
 classes, 258-259
 installing, 258
 overview, 257-259
 PEAR, 229
lines, drawing, 167-169
list() function, 108-110
listening sockets, 215-216
locking tables, 243
logic. See statements
logical operators, 79-80
looping
 arrays, 106-110
 Battle Tank, 201-205
 databases, 136-137
 printing game boards, 111-112

M

make files, 24
massively-multiplayer online game (MMOG). *See Kiddy Cartel*
matching patterns
 functions, 69-71
 regular expressions, 66-69
math operators. *See operators*
menu, Battle Tank, 199-200
methods, 48-75
Ming
 ActionScript, 258
 classes, 258-259
 installing, 258
 overview, 257-259
MMO. *See Kiddy Cartel*
morphing Flash movies, 273-275
move() function, 271-275
move history, 154-155
movePenTo() function, 262-263
MovePiece() function, 147
moveTo() function, 271-275
movies, Flash
 ActionScript, 275-278
 animating, 271-275
 background color, 260
 bitmaps, 269-270
 creating, 259-261
 data types, 275-278
 drawing, 262-265
 filling objects, 265-270
 images, 262-265
 morphing, 273-275
 printing, 260
 size, 259-261
 speed, 260
moving. See coordinates
multi-dimensional arrays, 110-112
MyFilledPolygon() function, 174
MyFilledRectangle() function, 171-172
MyPolygon() function, 173-174
MyRectangle() function, 170
MySQL
 installing, 230-231
 PEAR, 229
MyTTFBox() function, 183-184

N

naming variables, 59-60
next() function, 109-110
nextFrame() function, 272-273
non-relational databases, 134
number_format() function, 65
numbers
 formatting strings, 65
 image colors, 165-166

O

object oriented databases, 133
objects, Flash movies, 265-270
ObjectStore Web site, 133
OnError() function, 239
OnRollBack() function, 239-248
opacity, images, 189-191
opening databases, 134-136
operators
 arithmetic, 78
 binary, 79-80
 bitwise, 82-85
 comparison, 78-79
 concatenating, 81
 logical, 79-80
 overview, 77
 precedence, 86-87
 ternary, 80-81
 variable assignment shortcuts, 85-86
output() function, 260

P

padding arguments, strings, 64-65
pages
 browsers, interpreting, 4
 creating, 56
parameters, passing, 95-98
passing
 parameters, 95-98
 sessions, 6-7
pattern matching, regular expressions, 66-69
PEAR (PHP Extension and Application Repository), 229
Personal Web server (PWS)
 IIS comparison, 14-15
 installing, 14-18

PHP

code blocks, 56
 HTML comparison, 55
 pages
 browsers, interpreting, 4
 creating, 56
PHP Extension and Application Repository (PEAR), 229
phpinfo() function, 159, 219
physics, velocity, 204
pieces, Chess game, 143-151
pixels, drawing, 167-169
placing. *See coordinates*
platforms overview, 23-24
png files, 37-38
polygons, drawing, 172-174
ports, servers, 5
POST method, 48, 71-75
precedence, operators, 86-87
prefixes, variables, 59-60
prev() function, 109-110
PRIMARY KEY keyword, 232
printf() function, 64-65
printing
 Flash movies, 260
 game boards, 111-112
 HTML tags, 56
 text, 56
processing forms, 71-75
ProcessInput() function, 151-155
protocols, sockets, 214-215
PutPiece() function, 147
PWS (Personal Web Server)
 IIS comparison, 14-15
 installing, 14-18

Q

Query() function, 248

R

records, 233
rectangles, drawing, 169-172
recursion, functions, 98-99
regular expressions
 functions, 69-71
 pattern matching, 66-69

relational databases, 133
MySQL
 installing, 230-231
 PEAR, 229
 overview, 231-233
render() function
 Chess game, 143-151
 tic-tac-toe, 119-123
Render() function, 201-205, 225-228
RenderInterface() function, 197-198
RenderTanks() function, 196-197
RenderTerrain() function, 196
reset() function, 108-110
rotate() function, 271-275
rsort() function, 115
rules, Kiddy Cartel, 233-235
running scripts, command line, 220

S

saving images, 184-185
scripts, command line, 220
security
 sessions, passing, 6
 tables, commands, 243
<SELECT> HTML tag, 49-51
selecting delimiters, 56
semicolons, troubleshooting, 56
serialize() function, 137-139
servers
 client relationship, 3-5
 creating, 220-223
 debugging states, 5
 installing
 Apache, 18-21
 IIS, 14-18
 PWS, 14-18
 UNIX, 19-21
 multiple connections, 220-223
 ports, 5
 sockets, 213
session variables, states, 5-13
sessions
 functions, 10-13
 passing
 cookies, 6
 security, 6
 URLs, 6-7
 states, session variables, 5-13

setBackground() function, 260
setDimension() function, 259-261
setLeftFill() function, 265-270
setLine() function, 262-265
setRate() function, 260
setRightFill() function, 265-270
settype() function, 60
shortcuts, assignment operators, 85-86
size
 Flash movies, 259-261
 HTML, 38
 images, 187-189
socket_bind() function, 215-216
socket_create() function, 215
socket_listen() function, 215-216
sockets
 Battle Tank, integrating, 224-228
 binding, 215-216
 clients, 213, 223-224
 creating, 215
 enabling, 219
 functions, 216-219
 listening, 215-216
 protocols, 214-215
 servers, 213, 220-223
 types, 214
sort() function, 113-114
sorting arrays, 112-117
specifications, Kiddy Cartel, 233-235
speed, Flash movies, 260
split() function, 71
sprintf() function, 64-65
SQL
 MySQL
 installing, 230-231
 PEAR, 229
 T-SQL, 231-233
StartBoard() function, 147
StartGame() function, 124-125, 199-200
statements
 do, 93-94
 else, 88-90
 for, 94, 106-108
 if, 88-90
 overview, 86
 switch, 90-93, 123
 while, 93-94, 108-110

states

Battle Tank, 194-195, 198-199
 Chess game, 141-142
 servers, debugging, 5
 session variables, 5-13
 tic-tac-toe, 119

strings. *See also* text

array indexes, 105-106
 formatting numbers, 65
 functions, 61-65
 operators. *See* operators
 padding arguments, 64-65
 pattern matching
 functions, 69-71
 regular expressions, 66-69

su command, 26**sub-commands, 240-245****support, gifs, 157**

SWFAction() class, 276-278
SWFBitmap() class, 269-270
SWFDisplayItem() class, 271-275
SWFFill() class, 265-270
SWFGradient() class, 267-268
SWFMorph() class, 273-275
SWFMovie() class, 259-261
SWFShape() class, 262, 269-270
switch statements, 90-93, 123

T**<TABLE> HTML tag, 41-48****tables**

Battle Tank, 197-198
 creating, 231-232
 HTML, 41-48
 locking commands, 243
 records, 233
 security, 243
 unlocking commands, 243

tags, HTML

<BODY>, 33-36

, 36
 <DIV>, 197
 <FORM>, 48-51
 <HEAD>, 35
 <IFRAME>, 154-155
 , 39-41
 <INPUT>, 48-51

<SELECT>, 49-51
 <TABLE>, 41-48
 <TD>, 42-48
 <TEXTAREA>, 49-51
 <TITLE>, 35
 <TR>, 42-48
 overview, 31-32
 printing, 56
 text formatting, 35

TakePiece() function, 147**tanks, 196-197****TCP/IP (Transmission Control Protocol/Internet Protocol), 13-14****<TD> HTML tag, 42-48****ternary operators, 80-81****terrain. *See* background****testing. *See* debugging****text. *See also* strings**

formatting, 35
 images, 179-184
 printing, 56
<TEXTAREA> HTML tag, 49-51

tic-tac-toe

arrays, 117-132
 CheckFull() function, 126-128
 CheckWin() function, 126-128
 ComputerMove() function, 128-131
 ComputerRandomMove() function, 128-131
 defining constants, 118
 defining globals, 118
 directory, 117
 DrawBoard() function, 125-126
 EndGame() function, 124-125
 functions, 123-131
 HTML, 118-119
 render() function, 119-123
 StartGame() function, 124-125
 states, 119
 switch statement, 123

<TITLE> HTML tag, 35**<TR> HTML tag, 42-48****translucency, 189-191****Transmission Control Protocol/Internet Protocol (TCP/IP), 13-14****transparency**

HTML, 37
 images, 163-165

troubleshooting. *See debugging*

T-SQL overview, 231-233

type casting, 58

type juggling, 57-58

types, sockets, 214

U

UNIX, 23

installing, 24-26

servers, 19-21

unlocking tables, 243

unset() function, 61

UPDATE function, 233

UpdateMoveList() function, 153-154

updating databases

entries, 139-140

records, 233

URLs, passing, 6-7

user input, Chess game, 151-155

usort() function, 116-117

V

values

dynamic, 58

operators

arithmetic, 78

binary, 79-80

bitwise, 82-85

comparison, 78-79

concatenating, 81

logical, 79-80

overview, 77

precedence, 86-87

ternary, 80-81

variable assignment shortcuts, 85-86

variable variables, 58

variables

Battle Tank, 199-200, 204

case sensitivity, 57

constants, 58-59

declaring, 57

dynamic values, 58

functions, 60-61

naming, 59-60

operators

arithmetic, 78

binary, 79-80

bitwise, 82-85

comparison, 78-79

concatenating, 81

logical, 79-80

overview, 77

precedence, 86-87

ternary, 80-81

variable assignment shortcuts, 85-86

session variables, 5-13

shortcut operators, 85-86

type casting, 58

type juggling, 57-58

variable, 58

velocity, 204

Versant Web site, 133

W-Z

Web

pages

browsers, interpreting, 4

creating, 56

servers. *See servers*

sites, 133

while statements, 93-94, 108-110

Windows, installing, 26-29

THOMSON

COURSE TECHNOLOGY™

Professional ■ Trade ■ Reference

GOT GAME?

COMING SPRING 2004!



3D Game Programming
All in One
1-59200-136-X ■ \$49.99



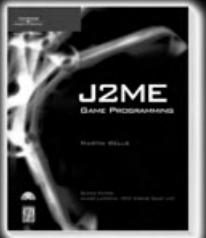
Beginning OpenGL
Game Programming
1-59200-369-9 ■ \$29.99



The Dark Side
of Game Texturing
1-59200-350-8 ■ \$39.99



3D Game Engine
Programming
1-59200-351-6 ■ \$59.99



J2ME
Game Programming
1-59200-118-1 ■ \$59.99



A division of Course Technology



Call 1.800.354.9706 to order
Order online at www.courseptr.com

Gamedev.net

The most comprehensive game development resource

- The latest news in game development
- The most active forums and chatrooms anywhere, with insights and tips from experienced game developers
- Links to thousands of additional game development resources
- Thorough book and product reviews
- Over 1000 game development articles!

Game design

Graphics

DirectX

OpenGL

AI

Art

Music

Physics

Source Code

Sound

Assembly

And More!

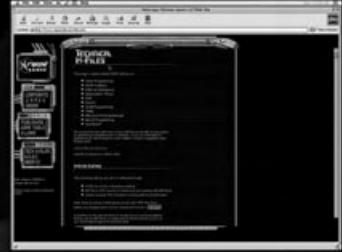
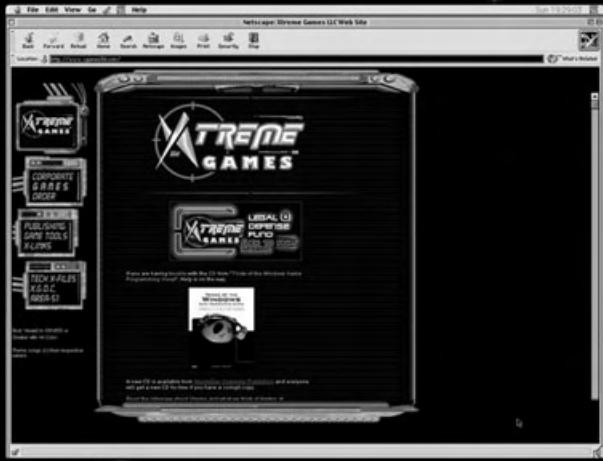


Gamedev.net

OpenGL is a registered trademark of Silicon Graphics, Inc.

Microsoft and DirectX are registered trademarks of Microsoft Corp. in the United States and/or other countries.

TAKE YOUR GAME TO THE **XTREME!**



Xtreme Games LLC was founded to help small game developers around the world create and publish their games on the commercial market. Xtreme Games helps younger developers break into the field of game programming by insulating them from complex legal and business issues. Xtreme Games has hundreds of developers around the world. If you're interested in becoming one of them, then visit us at www.xgames3d.com.

www.xgames3d.com



License Agreement/Notice of Limited Warranty

By opening the sealed disc container in this book, you agree to the following terms and conditions. If, upon reading the following license agreement and notice of limited warranty, you cannot agree to the terms and conditions set forth, return the unused book with unopened disc to the place where you purchased it for a refund.

License:

The enclosed software is copyrighted by the copyright holder(s) indicated on the software disc. You are licensed to copy the software onto a single computer for use by a single user and to a backup disc. You may not reproduce, make copies, or distribute copies or rent or lease the software in whole or in part, except with written permission of the copyright holder(s). You may transfer the enclosed disc only together with this license, and only if you destroy all other copies of the software and the transferee agrees to the terms of the license. You may not decompile, reverse assemble, or reverse engineer the software.

Notice of Limited Warranty:

The enclosed disc is warranted by Course PTR to be free of physical defects in materials and workmanship for a period of sixty (60) days from end user's purchase of the book/disc combination. During the sixty-day term of the limited warranty, Course PTR will provide a replacement disc upon the return of a defective disc.

Limited Liability:

The sole remedy for breach of this limited warranty shall consist entirely of replacement of the defective disc. IN NO EVENT SHALL COURSE PTR OR THE AUTHOR BE LIABLE FOR ANY other damages, including loss or corruption of data, changes in the functional characteristics of the hardware or operating system, deleterious interaction with other software, or any other special, incidental, or consequential DAMAGES that may arise, even if COURSE PTR and/or the author has previously been notified that the possibility of such damages exists.

Disclaimer of Warranties:

COURSE PTR and the author specifically disclaim any and all other warranties, either express or implied, including warranties of merchantability, suitability to a particular task or purpose, or freedom from errors. Some states do not allow for EXCLUSION of implied warranties or limitation of incidental or consequential damages, so these limitations mIGHT not apply to you.

Other:

This Agreement is governed by the laws of the State of Massachusetts without regard to choice of law principles. The United Convention of Contracts for the International Sale of Goods is specifically disclaimed. This Agreement constitutes the entire agreement between you and Course PTR regarding use of the software.