

Jumpstart your web development career using popular technologies including PHP and MySQL, Ajax, RSS, PayPal™, the Facebook Platform®, Amazon Web Services™, the Google Maps API™, the Google Analytics™ web analytics service, the Google AdSense™ and Google AdWords™ advertising services, and more!



Easy PHP Websites with the Zend Framework

W. Jason Gilmore

Bestselling author of *Beginning PHP and MySQL, Third Edition*

Easy PHP Websites

with the Zend Framework

W. Jason Gilmore

Easy PHP Websites with the Zend Framework

Copyright © 2009 W. Jason Gilmore

Published by

W.J. Gilmore, LLC

1373 Grandview Avenue, Suite 214

Columbus, Ohio 43212

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, scanning, recording or otherwise, except as permitted under the 1976 United States Copyright Act, without the prior written permission of the copyright owner and the publisher.

Trademarks: Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to no benefit of the trademark owner, with no intention of infringement of the trademark. Zend is a trademark of Zend Technologies Ltd. Amazon Web Services™ is a trademark of Amazon.com, Inc. Facebook® is a registered trademark of Facebook Inc. PayPal® is a registered trademark of PayPal, Inc. Google Maps API™, GoogleAdSense™, and Google AdWords™ are trademarks of Google Inc. All other trademarks are the property of their respective owners. W.J. Gilmore, LLC is not associated with any product or vendor mentioned in this book.

Limit of Liability: The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor W.J. Gilmore, LLC shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Source Code: The source code for this book is available at <http://www.easyphtwebsites.com/>.

Dedicated to my family

Table of Contents

| | |
|--|------------|
| CHAPTER 1. Configuring Your Development Environment | 1 |
| CHAPTER 2. Introducing PHP | 23 |
| CHAPTER 3. Interacting With Your Users | 51 |
| CHAPTER 4. Introducing MySQL | 75 |
| CHAPTER 5. Introducing the Zend Framework | 105 |
| CHAPTER 6. Talking to the Database with Zend_Db | 137 |
| CHAPTER 7. Processing Forms and Sending Email | 161 |
| CHAPTER 8. Managing Your User Community | 183 |
| CHAPTER 9. Integrating Google Maps | 211 |
| CHAPTER 10. Introducing the Amazon Associates Web Service | 231 |
| CHAPTER 11. Enhancing the User Interface with Ajax | 245 |
| CHAPTER 12. Extend Your Website with RSS and Facebook | 263 |
| CHAPTER 13. Monitor Traffic and Manage Ads with Google | 287 |
| CHAPTER 14. Accepting Online Payments with PayPal | 305 |
| CHAPTER 15. Introducing Zend_Tool | 315 |

Expanded Table of Contents

| | |
|--|------------|
| CHAPTER 1. Configuring Your Development Environment | 1 |
| Step #1. Installing Apache, MySQL, and PHP on Windows | 2 |
| Step #2. Testing Your Installation | 13 |
| Step #3: Choosing a Code Editor | 14 |
| Step #4. Exploring Other Useful Software | 17 |
| Step #5. Choosing a Web Hosting Provider | 19 |
| | |
| CHAPTER 2. Introducing PHP | 23 |
| Step #1. Creating Your First PHP-Enabled Web Page | 23 |
| Step #2. Publishing Spreadsheet Data to the Web | 38 |
| Displaying Data in a Table | 41 |
| Step #3. Managing Your Site Design Using Templates | 47 |
| | |
| CHAPTER 3. Interacting With Your Users | 51 |
| Step #1. Creating a Contact Form | 52 |
| Step #2. Validating User Input | 56 |
| Step #3. Repopulating Form Data | 65 |
| Step #4. Sending Form Data via E-mail | 67 |
| Step #5. More on Securing User Input | 72 |
| | |
| CHAPTER 4. Introducing MySQL | 75 |
| Step #1. What Is a Relational Database? | 76 |
| Step #2. Introducing MySQL | 77 |
| Step #3. Introducing phpMyAdmin | 83 |
| Step #4. Moving Your Data to MySQL | 86 |
| Step #5. Connecting Your Website to MySQL | 88 |
| Step #6. Restricting Access | 103 |
| | |
| CHAPTER 5. Introducing the Zend Framework | 105 |
| Step #1. What Is a Web Framework? | 106 |
| Step #2. Introducing the Zend Framework | 112 |
| Step #3. Installing the Zend Framework | 115 |
| Step #4. Testing Your Installation | 121 |
| Step #6. Creating the Website Layout | 125 |

| | |
|--|------------|
| Step #7. Creating a Configuration File | 130 |
| Step #8. The init() Method | 133 |
| Step #9. Creating Action Helpers | 133 |
| CHAPTER 6. Talking to the Database with Zend_Db | 137 |
| Step #1. Introducing Object-relational Mapping | 138 |
| Step #2. Introducing Zend_Db | 139 |
| Step #3. Creating Your First Model | 140 |
| Step #4. Querying Your Models | 141 |
| Step #5. Creating a Row Model | 145 |
| Step #6. Inserting, Updating, and Deleting Data | 146 |
| Step #7. Modeling Table Relationships | 148 |
| Step #8. JOINing Your Data | 151 |
| Step #9. Paginating Results with Zend_Paginator | 155 |
| Step #10. Creating and Managing Views | 157 |
| CHAPTER 7. Processing Forms and Sending Email | 161 |
| Step #1. Zend Framework Forms Fundamentals | 162 |
| Step #2. Creating the GameNomad Contact Form | 163 |
| Step #3. Validating Form Data | 166 |
| Step #5. Filtering Form Data | 177 |
| Step #6. Preventing Spamming Using CAPTCHAs | 178 |
| CHAPTER 8. Managing Your User Community | 183 |
| Step #1. Creating the Users Table and Model | 184 |
| Step #2. Registering Users | 188 |
| Step #4. Displaying User Profiles | 203 |
| CHAPTER 9. Integrating Google Maps | 211 |
| Step #1. Introducing the Google Maps API | 212 |
| Step #2. Adding Mapping Services to Your Website | 223 |
| CHAPTER 10. Introducing the Amazon Associates Web Service | 231 |
| Step #1. Introducing the Amazon Associates Web Service | 232 |
| Step #2. Introducing the Zend_Service_Amazon Component | 234 |

| | |
|---|------------|
| Step #3. Searching for Products | 239 |
| Step #4. Customer Reviews | 241 |
| CHAPTER 11. Enhancing the User Interface with Ajax | 245 |
| Step #1. Introducing JavaScript | 246 |
| Step #2. Introducing Ajax | 261 |
| CHAPTER 12. Extend Your Website with RSS and Facebook | 263 |
| Step #1. Building and Publishing RSS Feeds | 264 |
| Step #2. Introducing the Facebook Platform | 269 |
| Step #3. Building Your First Facebook Application | 274 |
| Step #4. Introducing the Facebook PHP Client Library | 276 |
| Step #5. Creating a Facebook Platform Controller | 277 |
| Step #6. Creating a Facebook Profile Tab | 280 |
| Step #7. Sending a Facebook User Notification | 282 |
| Step #8. Adding Facebook Status Updates | 283 |
| Step #9. Deploying Your Facebook Application | 285 |
| CHAPTER 13. Monitor Traffic and Manage Ads with Google | 287 |
| Step #1: Monitoring Traffic with Google Analytics | 287 |
| Step #2. Advertising with Google AdWords | 294 |
| Step #3: Earning Money with Google AdSense | 301 |
| CHAPTER 14. Accepting Online Payments with PayPal | 305 |
| Step #1. Integrating PayPal Website Payments Standard | 305 |
| Step #2. Exploring Third-Party E-Commerce Solutions | 309 |
| CHAPTER 15. Introducing Zend_Tool | 315 |
| Step #1. Configuring Zend_Tool | 315 |
| Step #2. Creating and Managing Projects with Zend_Tool | 316 |

About the Author

Jason Gilmore is founder of W.J. Gilmore LLC (<http://www.wjgilmore.com/>), a publishing and consulting firm based out of Columbus, Ohio. Formerly Apress' open source editor, Jason fostered the development of more than 60 books, along the way helping to transform their open source line into one of the industry's most respected publishing programs.

Jason is the author of several books, including the bestselling *Beginning PHP and MySQL: From Novice to Professional* (currently in its third edition), *Beginning PHP and PostgreSQL: From Novice to Professional*, and *Beginning PHP and Oracle: From Novice to Professional*. He has over 100 articles to his credit within prominent publications such as Developer.com, Linux Magazine, and TechTarget.

Jason is a cofounder of CodeMash (<http://www.codemash.org/>), a nonprofit organization tasked with hosting an annual namesake developer's conference, and was a member of the 2008 MySQL Conference speaker selection board.

About the Technical Reviewer

Chris J. Davis is a Senior Web Developer and Mac enthusiast living in North Texas. He is an international speaker on the intersection of technology and community as well as a founder of the Habari Project, a next generation publishing platform. Also a published author, Chris coauthored *Blog Design Solutions* from Friends of Ed.

Recently Chris has been invited to become a committer on the Apache Software Foundation Infrastructure team, which he enthusiastically accepted. You can find more out about Chris by visiting his website (<http://chrisj.davis.org>) or by following him on Twitter (<http://twitter.com/chrisj.davis>).

Acknowledgements

Ernest Hemingway once offered a particularly sober assessment of his trade, saying, "There is nothing to writing. All you do is sit down at a typewriter and bleed." This being my fifth book, and the first published under my eponymous press, W.J. Gilmore LLC, I can sympathize with his appraisal. Thankfully, psychological first aid is readily available from the family, friends, and colleagues who always make this process much less painful.

Chris Davis did an excellent job as a tech reviewer, catching numerous mistakes made as I fought with both code and prose.

Carlene DeFiore helped out immensely, reviewing chapters, helping with promotional strategy, and keeping me well fed.

Countless other individuals played key advisory roles in helping this book to finally see the light. In alphabetical order, they include David Futato, Jonathan Hassell, Stu Johnson, and Matt Zenko. I'd also like to thank the great people at the Grandview Heights Fifth Third Bank and U.S. Post Office for helping get the business off the ground.

I'd perhaps never have gotten into the publishing business without the help of Apress cofounder Gary Cornell, who first contacted me back in 2000 to write a book for what was then a very small but daring computer book upstart (one which has since grown into a rather large but still daring publisher). He played a pivotal role in my development as an author and later an editor for Apress. Thank you Gary.

Last but certainly not least, I'd like to thank you dear reader, for having enough faith to trade your hard-earned money for this book.

Introduction

Picture spending an evening hanging out with a group of gregarious savants from around the globe. Over hors d'oeuvres you debate the merits of globalism, and split time during dinner extolling the ferocity of the 2008 Pittsburgh Steelers defense and listening intently to two physicists argue over the plausibility of time travel. The evening concludes with no less than four renditions of Beethoven's Symphony no. 7, including electronica and hip-hop versions. No doubt, an evening such as this would leave even the most tempered of socialites admitting to being among rarified company. This also happens to be an evening any one of us could spend while surfing the World Wide Web.

We're living in the midst of the most prolific democratization of knowledge in the history of the world. In the twenty years since Tim Berners-Lee's authoring of the paper which led to the creation of the World Wide Web, we've progressed from a civilization dependent upon the often meager resources of small town libraries and yellow journalism of local media outlets to one which has practically all of the information in the history of the world available at the cost of just a few keystrokes. Never mind that most users tend to use this profound invention to keep up with Hollywood gossip.

Of course, the Web is enormously successful because it's a two way street. That is, it's just as easy to read up on a history of the Napoleonic campaigns as it is to create your own website on the topic. And with the proliferation of powerful programming languages and frameworks such as PHP and the Zend Framework, and the ability to plug into enormous databases via services such as Amazon Associates and Google Maps API, your ambition is limited only by your creativity and understanding of how these tools work. While you're on your own to address the former requisite, I hope this book goes a long way towards satisfying the latter by showing you how to push these tools to their very limits.

Intended Audience

Whether you're a serial entrepreneur, a hobbyist wishing to use the Web to introduce others to your pastime, or a project manager simply wishing to understand more about exactly what it is your IT-minded colleagues do during the day, this book is for you. I make no presumptions regarding the reader's level of programming knowledge, beginning at a logical starting point and gradually introducing new concepts accompanied by plenty of examples.

Likewise, if you're a seasoned PHP programmer wishing to learn more about the Zend Framework to build powerful websites in conjunction with popular technologies such as the Amazon Associates Web Service, Google Maps API, RSS, Facebook, and PayPal, you may wish to simply skip the four introductory chapters and jump right into the more advanced chapters.

About the Book

When I wrote the bestselling *Beginning PHP and MySQL: From Novice to Professional*, now in its third edition (Apress, 2008), my goal was to produce a book that somehow measured up to the unparalleled breadth of features offered by these two powerful technologies. At 1,044 pages, many would say my objective was met, and indeed at present it remains the most voluminous book written on the topic in the country, if not the world. If nothing else, it can hold a door open in even the most wicked

of windstorms.

When I set out to write *Easy PHP Websites with the Zend Framework*, my strategy was decidedly different. Rather than attempt to provide readers with a survey of every possible feature available to PHP, the Zend Framework, and the array of other popular technologies discussed in the book, I made a conscious effort to focus on key topics which I thought would be of most benefit to the time- and resource-challenged reader who was looking to maximize his every second spent working towards the goal of building a powerful website. Accordingly, each chapter is devoted to accomplishing a very specific, desired task, such as processing payments with PayPal, building a Facebook application, integrating Google Maps into a website, and building a product catalog using the Amazon Associates Web Service. You'll also gain from practical instruction on topics pertinent to the logistical-side of Web development, such as creating an ideal development environment, what to keep in mind when choosing a web hosting provider, and how to take advantage of services such as Google Analytics, Google AdWords, and Google AdSense.

In an attempt to keep the instruction topical and applicable to current trends, the book guides you through the hypothetical creation of a social networking website for video game enthusiasts. You'll start simple, learning how to convert a spreadsheet used to track your video game collection into a format capable of being viewed via the Web. From there the project builds in ambition, migrating the collection to a MySQL database, and subsequently turning to the popular Zend Framework in order to more effectively manage user accounts, process data submitted via Web forms, and send e-mail. The final six chapters show you how to integrate the Zend Framework with popular third-party services such as Google Maps, Amazon.com, Facebook, and PayPal to build an even more compelling website. Of course, you're free to pick-and-choose among these topics, using the chapters what you find most appealing and disregarding the rest.

Incidentally, the video game theme wasn't altogether contrived. My own interest in video games led to the creation of GameNomad (<http://www.gamenomad.com/>), a social networking website which gives gamers and their like-minded friends tools for managing their respective game collections. Quite a bit of the code and many examples are based on the website, and in fact you'll find quite a bit of the code used to power the GameNomad website within the source code download.

Book Contents

Easy PHP Websites with the Zend Framework is broken into fourteen chapters, and supplemented with several videos available to readers via the website <http://www.easypHPwebsites.com/>.

Chapter 1. Configuring Your Development Environment

In this introductory chapter you'll learn how to create the ultimate development environment. In addition to showing you how to install Apache, PHP and MySQL on Windows XP and Vista, you'll learn about useful development tools such as Eclipse PDT, Freemind, and the Firefox Developer Add-on. We'll also discuss the different types of hosting providers, and what to keep in mind when selecting a provider.

Chapter 2. Introducing PHP

With the development environment in place, it's time to start coding! In this chapter you'll be introduced to PHP's fundamental syntax, setting the stage for publishing your first dynamic web page.

Rather than guide you through a bunch of contrived examples, you'll instead learn how to do something useful: migrating an Excel spreadsheet to the Web! You'll also learn how to effectively manage your website design by using a simple but effective approach.

Chapter 3. Interacting with Your Users

The beauty of the web is that it's a two-way street; not only can you disseminate information to interested visitors, but so can your visitors provide feedback and interact in a variety of ways. The most commonplace way to do so is via a web form. In this chapter we'll cover form fundamentals, teaching you not only how to retrieve data submitted through a form, but also validate the data to ensure it meets your expectations. You'll also learn how to send this form data directly to your inbox by sending an e-mail directly from a PHP script.

Chapter 4. Introducing MySQL

As your website ambitions grow, so will your need to devise a way to effectively manage data. The most commonplace way for doing so is using a relational database. In this chapter I'll introduce you to MySQL, one of the most popular such databases in the world.

Chapter 5. Introducing the Zend Framework

Building a successful website is a product of much more than mere programming. You need to deal with an enormous number of factors, including configuration issues such as creating user-friendly URLs and managing site configuration data, security issues such as validating user input and preventing other commonplace attacks, user interface issues such as managing design templates and adding Ajax features, and integrating third-party services. Because all web developers face similar issues, web frameworks were created to handle many of these gory details for you. Once such framework is the popular PHP-driven Zend Framework, and this chapter introduces this impressive solution.

Chapter 6. Talking to the Database with Zend_Db

Chapter 4 showed you how to write queries which PHP which made it possible to interact with a MySQL database. Although unquestionably useful, if you're like me there was something oddly unnatural about mixing PHP and SQL code together within a single script. In this chapter I'll show you how to have your cake and eat it too by using the Zend_Db component to interact with MySQL using purely PHP code.

Chapter 7. Processing Forms and Sending Email

This chapter extends what was learned in Chapter 3, using the Zend Framework to process user-submitted data. You'll also learn about the Zend Framework's powerful built-in data validation and filtering features, how to send e-mail using the Zend_Mail component, and how to foil spammer activity using a mechanism known as a CAPTCHA.

Chapter 8. Managing Your User Community

These days, a successful web-based venture is often the direct result of your ability to create a successful online community. In this chapter you'll learn how to build fundamental community mechanisms such as user registration, user login and logout, password recovery, user profile display, and connecting users via a simple social networking mechanism.

Chapter 9. Integrating Google Maps

Google's release of an interface to the popular Google Maps mapping solution opened the floodgates for thousands of location-based websites. In this chapter you'll learn how to use this API to integrate

Google's mapping features into your own website. You'll also learn how to use the Google Geocoder to convert user's mailing addresses into latitudinal and longitudinal coordinates.

Chapter 10. Introducing Amazon Associates Web Service

The Amazon Associates Web Service (AWS) offers developers around the globe with a great way to earn revenue by advertising Amazon's product catalog in new and interesting ways. In this chapter you'll learn how to take advantage of this program to create your own custom catalog. You'll also learn how to retrieve Amazon reviews for a particular product, and even how to use a neat Zend Framework view helper feature to tie a visually appealing starred effect to each review.

Chapter 11. Enhancing the User Interface with Ajax

Enhancing the user experience by enhancing the interface is all the rage these days, a process most commonly done using JavaScript and a programming technique known as Ajax. In this chapter I'll introduce you to both JavaScript and Ajax, alongside the popular Prototype JavaScript framework and Scriptaculous libraries.

Chapter 12. Extend Your Website with RSS and Facebook

Today's users expect to be able to consume information in a format and at a time convenient to them, whether it be via email, a news aggregator such as Google Reader, or even via another website altogether such as Facebook. Much of the information assembled in this digestible manner comes by way of RSS feeds, a convenient markup format similar to HTML which is conducive to publishing website updates and other information in digest format. In this chapter you'll learn how to publish your own feeds using the Zend Framework's Zend_Feed component.

You'll also learn how to create a Facebook application using the Facebook Platform. Creating a Facebook application is easier than you think, and has the potential to introduce your website to millions of new users.

Chapter 13. Monitor Traffic and Managing Ads with Google

Putting countless hours into your website is somewhat fruitless if you lack the ability to closely track visitor traffic and demographics. Thankfully, doing so is a breeze using the Google Analytics web analytics service. In this chapter you'll learn how to configure this free service for use on your own website. You'll also learn how to spread the word about your website using the world's most popular online advertising network, namely the Google AdWords advertising program. While you're at it, why not earn some advertising revenue by publishing ads on your website using the Google AdSense advertising program, another topic discussed in this chapter.

Chapter 14. Accepting Online Payments with PayPal

In this chapter, we'll discuss e-commerce transactions using PayPal Checkout, which is one of the world's most popular services for accepting online payments. For those readers who require a somewhat more sophisticated e-commerce solution, we'll close the chapter with an overview of three popular e-commerce solutions.

Chapter 15. Introducing Zend_Tool

In the final chapter of the book, you'll learn about the new Zend_Tool component, a command-line utility which will make creating new Zend Framework-driven applications a breeze.

Downloading the Source Code

The source code for this book is available via the website <http://www.easyphpwebsites.com/>.

Contact the Author

Over the past decade I've responded to thousands of reader e-mails, and eagerly look forward to responding to yours. E-mail me at jason@easyphpwebsites.com with your questions and comments regarding this book, or anything else that strikes your fancy.

CHAPTER 1

Configuring Your Development Environment

Chances are you'll be relying on a web hosting provider to manage your website, however for reason of convenience and efficiency you'll want to develop the website on a personal workstation or laptop. In this chapter I'll show you how to install Apache, PHP and MySQL on the Windows XP and Windows Vista operating systems. You'll also be introduced to several integrated development environments (IDEs) and other utilities which will prove invaluable when building your web applications, as well as become acquainted with the issues you'll need to keep in mind when selecting a web hosting provider.

VIDEO. Introducing PHP and MySQL

Just what makes the PHP language and MySQL database so popular? Why are they so commonly used together to build websites both large and small? This video introduces you to PHP and MySQL, focusing on the features which have turned open source projects with modest beginnings into two of the world's game-changing technologies. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Chapter Steps

The goals of this chapter are accomplished in five steps:

- **Step #1. Installing Apache, MySQL and PHP on Windows:** In this step you'll learn how to install Apache, PHP and MySQL both the easy way (using an automated installer called XAMPP) and the hard way (manually) on the Windows XP or Vista operating systems. Special attention is paid to troubleshooting confusing installation and configuration issues on the Windows platform. Don't worry if your laptop or computer is a few years old, because the software can run quite ably with minimal resources.
- **Step #2. Testing Your Installation:** In this step you'll test your installation by creating your first PHP script. Although the next chapter is devoted to acquainting you with PHP syntax, in this chapter you'll learn how to embed PHP into a web page and make it available for request via the web server.
- **Step #3. Choosing a Code Editor:** Creating PHP programs doesn't require special development tools; in fact you could write working PHP code with a text editor as simple as Notepad if you'd like. However, there are numerous code editors suited specifically for PHP development which will help you to create web applications much more quickly and efficiently. In this step I'll introduce you to several of these editors
- **Step #4. Exploring Other Useful Software:** In addition to a code editor, you'll want to take advantage of several other great development-related tools. In this step I'll introduce you to some of my favorites.
- **Step #5. Choosing a Web Hosting Provider:** Eventually you'll want to make your website

available to others. Most individuals and organizations rely on a web hosting provider to host their website. In this final step you'll learn about what factors you should keep in mind when choosing a hosting provider.

NOTE. Linux-minded readers might be wondering why installation instructions for their favorite distribution aren't provided in this chapter. The simple answer is that there are simply too many distributions to cover, making any attempt to do so a largely fruitless exercise. Also, because each major Linux distribution tends to maintain their own software repositories, chances are you already have Apache and PHP installed!

Step #1. Installing Apache, MySQL, and PHP on Windows

In this section you'll learn about two different approaches to installing Apache, MySQL, and PHP (hereafter referred to as AMP when collectively discussing all three) on Windows. The first presents the easiest solution, involving an automated installer called XAMPP. The second solution guides you through the manual installation of AMP in three separate steps. I recommend the latter approach because it will no doubt give you a much better technical understanding of how the AMP platform interoperates. Regardless, I won't hold it against you for choosing the easy way out, and think you'll be able to go a long way towards building and managing powerful websites even without this additional installation and configuration knowledge.

Installing Apache, PHP, and MySQL: The Easy Way

Over the years, I've received perhaps hundreds, if not thousands of reader emails. The overwhelming majority of these readers wrote with support questions specific to the problems they were encountering while installing Apache, PHP, or MySQL, or getting all three to properly communicate with one another. It isn't that installing them is particularly difficult, but rather that if something goes wrong, it can be difficult to troubleshoot the problem.

In response to the frustrating issues encountered by newcomers to installing this sort of software, developers Kay Vogelgesang and Kai Seidler got together to create XAMPP, an automated solution which installs Apache, PHP, MySQL for you. It removes all of the potential frustration occasionally encountered when installing one or all of these applications, and is available for all major platforms, including Windows (98, NT, 2000, 2003, XP, and Vista), Linux (known to work on Debian, mandrake, RedHat, and SuSE), Mac OS X, and Solaris.

Installing XAMPP Lite

To install XAMPP, navigate to <http://www.apachefriends.org/en/xampp.html> and click upon the appropriate platform-specific link. In addition to Apache, MySQL, and PHP you'll see each package comes with a variety of additional software, including an FTP server, the Webalizer traffic analysis program, and phpMyAdmin (introduced in the later section, "Introducing phpMyAdmin"). If you're running Windows, you have the option of foregoing the installation of this additional software by downloading XAMPP Lite, which installs just Apache, MySQL, Perl, PHP, and phpMyAdmin. In this section I'll show you how to install XAMPP Lite.

Once you've downloaded XAMPP Lite, double-click the downloaded file's icon to begin the extraction process. The extraction path will default to the same directory as the installer's download location. I suggest changing this location to your C:\ drive, as shown in Figure 1-1. A directory named xampplite being created along that path, with Apache, PHP and MySQL along with various other items being placed in this directory.



Figure 1-1. Extracting XAMPP Lite

Next you should navigate to the newly created directory and double-click the `setup_xampp.bat` icon. Executing this file will initiate the installation and configuration process. Upon completion, the installation window will be similar to that shown in Figure 1-2.

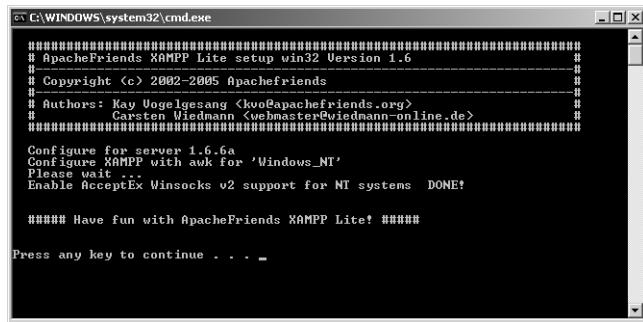


Figure 1-2. Notification of successful XAMPP installation

Upgrading XAMPP Lite

The various components comprising XAMPP are constantly being updated for reason of fixing bugs, adding and improving features, and resolving security issues. Accordingly, you'll likely occasionally want to upgrade XAMPP to take advantage of these improvements. Doing so is surprisingly easy; just delete the existing XAMPP installation directory and reinstall using the aforementioned instructions! Don't forget to back up your existing documents before doing so, in case something goes awry!

In conclusion, keep in mind XAMPP is intended for development purposes only, and should not under any circumstances be used to run a production website! This is because XAMPP is configured to provide users with almost no obstacles in the way of experimenting with any feature of the installed software. However the removal of these obstacles comes with similarly lowered barriers for malicious users seeking to exploit your system or data.

With XAMPP installed, you can skip ahead to "Step #3. Choosing a Code Editor"!

Installing Apache, MySQL, and PHP: The Hard Way

Ok, so installing AMP manually isn't exactly "the hard way". It just takes a bit more time and patience than otherwise required when using XAMPP. With this additional investment comes a better understanding of how Apache, MySQL and PHP work together. Provided you carefully follow the instructions provided in this section, the process should work flawlessly.

Installing Apache

To install Apache, navigate to <http://httpd.apache.org/download.cgi>. You'll be redirected to the Apache Software Foundation page listing the latest available versions. You're looking for the version deemed "best available". However, you don't want to download the Win32 Source version; instead, click on the `Other files` link, then click upon `binaries`, then `win32`, and finally download the release having the highest version number and having a title of `Win32 Binary without crypto (no mod_ssl) (MSI Installer)`. Select a location of your choosing (I usually place installers on the desktop so I can easily sweep them into the Recycle Bin afterwards), and once downloaded, double-click the icon to begin the installation process.

Preliminary Installation Procedures

You'll first be greeted with a screen welcoming you to the Apache Installation Wizard. Click `Next >` to continue. Next up is a screen prompting you to read and accept the Apache license. Take some time to familiarize yourself with the license, accept the terms, and click `Next >` to continue. Next you'll be prompted to peruse the `README` file, which contains numerous valuable notes regarding running Apache on Windows. Take a moment to read this text, and click `Next >` to continue.

Network Information

At this point you'll be prompted to provide three items of information (Figure 1-3): the server's network domain, the server name, and the administrator's e-mail address. Newcomers tend to get hung up at this stage, however the solution is simple: just input `localhost` for both the network domain and server name, and leave the administrator's e-mail address as-is.

The term `localhost` is an alias for the address `127.0.0.1`, which happens to be the address used to identify the local computer. Therefore once Apache is running, any calls to `http://localhost/` will prompt the local Apache installation to respond.

Regarding the Administrator's e-mail address, in certain configurations this address is provided to users in the case of a server error, however because I presume this Apache installation will be used for development purposes, you and your team will be the only users interacting with the server, and it's likely you'll know who to contact in such cases! That said, fill in any e-mail address you please.

At the conclusion of this screen, you're asked whether you'd like to install Apache as a service, meaning it will start and stop automatically when Windows is booted and shutdown, respectively. This will also configure Apache to listen on port 80, the default for web servers. Alternatively, you can install Apache in a fashion that will require you to start and stop it manually, and run it on a non-standard

port. Because you'll likely be using Apache on a regular basis, I recommend leaving the default selection and installing it as a service.

In summary, unless you know what you're doing, input `localhost` for the Network Domain and Server Name, and leave the Administrator's E-mail Address as-is. Also, leave the default Service selection as-is, and click `Next >` to continue.

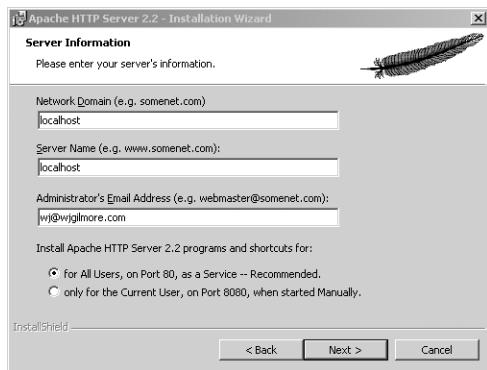


Figure 1-3. Providing the Network Domain, Server Name, Administrator's E-mail Address

Choosing Apache's Installation Procedure

Next up you'll be prompted to choose Apache's installation procedure (Figure 1-4). The first, `Typical`, simply uses a predefined set of defaults regarding the installation path and what Apache components will be installed. The second, `Custom`, allows you to decide these settings. Because we want to change Apache's installation path, click `Custom` and then click `Next >` to continue.

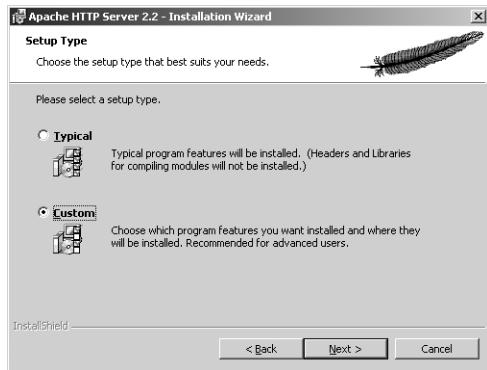


Figure 1-4. Choosing Apache's installation type

Modifying Apache's Default Installation Parameters

We're almost done. On this screen (Figure 1-5) you have the option of determining what Apache features are installed, and tweaking Apache's installation location. Leaving the defaults settings untouched is fine, so let's move on to the installation location.

The default installation location resides deep within your Program Files directory. However, because you'll often want to navigate into Apache's directory in order to view the contents of your web site and to modify Apache's configuration, I suggest changing this to C:\apache. If you're running Windows Vista, you won't be allowed to install applications into C:\, however you can work around this annoying obstacle by first manually creating a directory in C:\, and then choosing that directory using the Change... button. Once the new installation location is chosen, click Next > to continue.

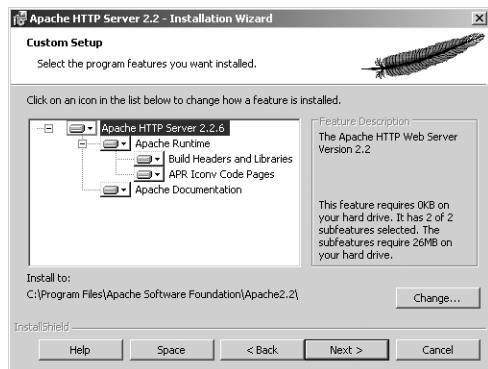


Figure 1-5. Modifying Apache's installation parameters

Apache now has all of the information it requires, and is ready to install. On the final screen, click Install to begin the process. The installer will copy the necessary files to your laptop, and configure the Apache service. You'll next be notified of a successful installation.

Testing Your Installation

Once installed, Apache will automatically start, and begin listening for connection requests. Start your web browser, and navigate to <http://localhost/>. You should see a simple web page containing the statement "It works!".

While Apache is now configured to serve HTML pages, it doesn't yet understand what to do with PHP scripts. Once PHP is installed, we'll return to Apache and make the necessary configuration adjustments to enable this ability.

Installing MySQL

Moving along with the installation process, we'll next install MySQL. Like Apache, MySQL offers a great installation wizard for Windows users. To begin the installation process, navigate to <http://dev.mysql.com/downloads/mysql/>, where you'll be able to select your platform and be taken to the downloads section.

On this page you'll see three versions are available; choose the Windows Essentials (x86) version. This version provides everything you'll need to use MySQL without unnecessarily weighing the server down with superfluous features. On the next page you'll be asked whether you'd like to register for the MySQL web site. Presuming you'd rather not for the moment, click the "No thanks, just take me to the downloads!" link.

Finally, you're prompted to choose a download server. The MySQL web site will identify your current geographical location based on your IP, giving you a select set of locations. Go ahead and choose the server residing closest to your current location, and the download process will begin.

Once downloaded, double-click the installer icon to begin the installation process. Like Apache, you'll be guided through a series of screen prompts. In this section I'll guide you through each.

The first screen is simply a setup wizard welcome screen. Click Next to move on to the next screen.

Choosing the MySQL Setup Type

On this screen (Figure 1-6) you'll choose the setup type which best fits your needs, of which three are available. You might be tempted to select *Complete*, however it's packaged with several features such as the embedded server (used for kiosks and similar devices), the documentation, and a few other items you're unlikely to use during your initial explorations. *Typical* isn't recommended either though, because it will automatically install MySQL in your Program Files path, which can be a highly inconvenient location when the time comes to write administration scripts such as those used to backup the database. To change this path, I suggest choosing *Custom* and clicking the Next button.



Figure 1-6. Choosing from MySQL's Three Setup Procedures

Modifying MySQL's Default Installation Parameters

Because you chose the Custom Setup Type, you'll next be prompted to identify which MySQL components should be installed, and determine its installation path (Figure 1-7). Of the two tasks, the only one you need to deal with is the latter. Click the Change... button and set the path to C:\mysql. Doing so places MySQL in an easily accessible location both when using the command-prompt and Windows explorer

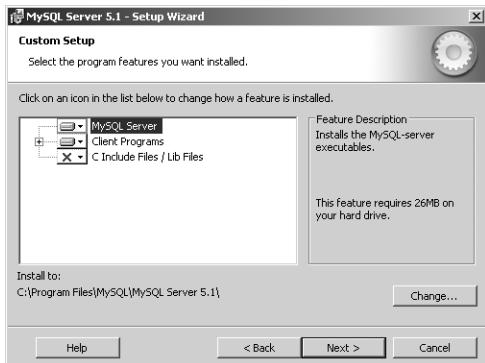


Figure 1-7. Modifying MySQL's default installation parameters

Click **Next** to continue, and MySQL is ready to install. You'll be greeted with a final installation screen informing you of the setup type and the installation directory. Click **Install** to install MySQL.

Once installation is complete, you'll be greeted with a series of informational screens highlighting MySQL offerings. Take a moment to read through these, clicking **Next** to advance to the next screen. Finally, you'll be informed the installation wizard has finished, however a checked option titled "Configure the MySQL Server now" informs you of the pending commencement of the MySQL server configuration wizard. Leave that option checked and click the **Finish** button to continue.

Configuring MySQL

The MySQL Server Configuration Wizard grants you the ability to tweak MySQL's default configuration settings. Using this wizard you can control whether MySQL will be installed as a service, optimize it for specific purposes such as data analysis or e-commerce, and throttle the number of allowable simultaneous connections. For general development purposes we can leave most of these options untouched, however we are going to take advantage of this wizard's ability to change one very important setting: the MySQL root user password. Along the way you'll have the opportunity to review exactly what sorts of configuration options are at your disposal.

The MySQL Server Configuration Wizard begins with a welcome screen. Click **Next** to move on to the first step.

Choosing a Configuration Wizard

At this step you're prompted to choose the Detailed Configuration Wizard or the Standard Configuration Wizard (Figure 1-8). For our purposes the slimmed-down Standard Configuration Wizard is suffice, so go ahead and select this option and click **Next** to continue.

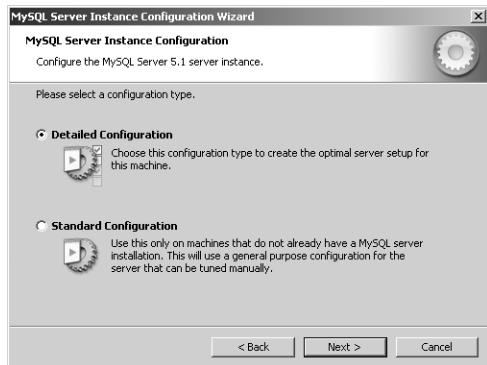


Figure 1-8. Choosing a configuration wizard

Configuring the MySQL Server Instance

Next you're prompted to decide whether MySQL should be installed as a service, and if so, what the service should be named. As was discussed with Apache, I strongly recommend installing MySQL as a service, because it will result in MySQL being automatically started and stopped upon operating system boot and shutdown. You can leave the service name as-is.



Figure 1-9. Configuring the MySQL server instance

You're also prompted to include MySQL's Bin directory in the Windows path. If you're planning on exclusively using the web-based phpMyAdmin MySQL administrator (introduced in Chapter 3), you can leave this option unchecked. Alternatively, if at some point in the future you think you'd like to experiment with the many MySQL utilities bundled with the software (likely you will), check this option.

Setting the root Password

Next you'll be prompted to set the root password (Figure 1-10). Even if your machine is intended solely for development purposes, you should nonetheless exercise sound security practices and set a password, because by default one is not set. I suggest choosing a password consisting of at least six characters, and having a mix of letters and numbers.

Do not under any circumstances enable root access from remote machines, as this could open up your MySQL database to other machines within your network. Also, do not create an anonymous account. Once you've chosen and verified a password, click **Next** to continue.



Figure 1-10. Setting the root password

NOTE. MySQL's root user is kind of like Windows' administrator account or Linux root user. It is capable of modifying practically every configurable characteristic of MySQL, and can create, modify, and delete all databases, tables, and data. Therefore, you should exercise extreme caution when using this account. If you're in the dark regarding how a MySQL account is even used; not to worry as this will all become clear in later chapters.

Applying the Configuration Settings

Finally, you're presented with a roadmap of what the wizard is going to do to apply the specified configuration settings. Click **Execute** to start this process. Once complete, MySQL will inform you of the successful application. Click **Finish** to exit the configuration wizard. Congratulations, MySQL is now installed!

Installing PHP

In the final step of the AMP installation process, we'll install PHP. Like Apache and MySQL, an installer is available, making the process rather simple. To begin, navigate to <http://www.php.net/downloads.php> and click on the link titled **PHP X.X.X installer**, where X.X.X is a placeholder for the latest stable version number. From here you'll be presented with a list of servers from which you can download the installer. Choose the location closest to you to begin the download process.

Once downloaded, double-click the installer to begin the installation process. You'll be greeted with a screen welcoming you to the PHP Setup Wizard. Click **Next** to move to the next screen.

The next screen prompts you to read and accept PHP's end user licensing agreement. Take a moment to read the agreement, accept the terms, and click **Next** to continue.

Specifying the Installation Location

Next you'll be prompted to specify where PHP should be installed (Figure 1-11). I suggest changing the default (`C:\Program Files\PHP\`) to `C:\php`, as it will sit alongside the Apache and MySQL installation directories, presuming you went with the previously discussed Apache and MySQL installation directory suggestions.

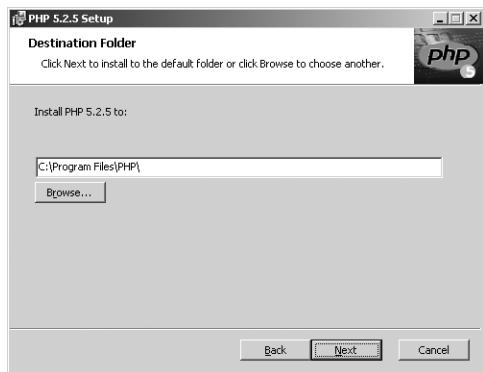


Figure 1-11. Specifying PHP's installation location

Choosing a Web Server

Next you'll be asked which web server you plan on using to serve PHP scripts (Figure 1-12). Because you installed Apache 2.2 or greater, go ahead and choose Apache 2.2.x Module, and click Next to continue.

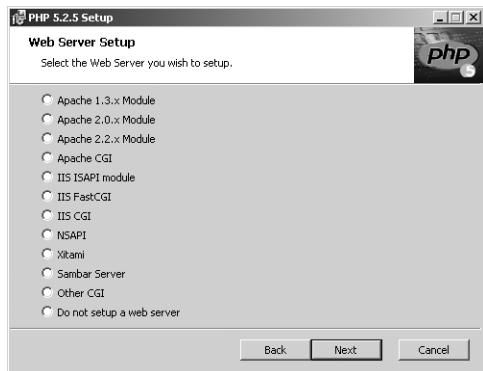


Figure 1-12. Choosing a Web Server

Identifying the Apache Configuration Directory

Apache's configuration file (`httpd.conf`) must be modified in order to be able to recognize and serve PHP-driven pages. While it's trivial to do this manually, the installer will take care of it for you (Figure 1-13)! To initiate the changes, browse to the directory in which you installed Apache by clicking the 'Browse...' button and navigating to that location. Click OK, and then click Next to proceed to the

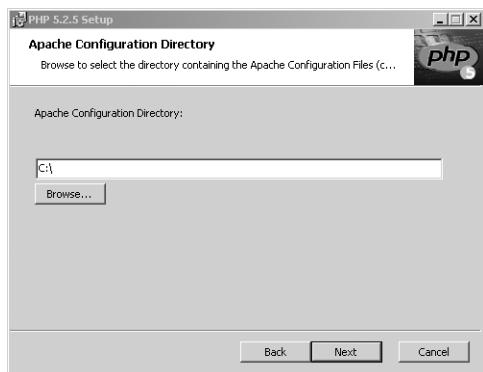


Figure 1-13. Identifying the Apache configuration directory

Installing PHP Extensions

One of the reasons for PHP's widespread popularity is its enormous set of features. Indeed, it seems PHP is capable of performing every conceivable task, from creating PDFs to interacting with PayPal to performing advanced mathematical operations. However, for reasons of maintainability and optimization, not all of these features are enabled by default. To enable a specific feature, you need to modify PHP's configuration file (`php.ini`) to load the appropriate extension, of which there are hundreds.

While it's easy to manually modify `php.ini` to enable extensions, we'll take advantage of the wizard's interface to enable one now (Figure 1-14). Expand the Extensions menu and enable MySQLi, which will enable PHP's ability to interact with MySQL. Additionally, expand the Extras menu and enable PEAR. The PHP Extension Application Repository, better known as PEAR, is used to install third-party extensions, several of which we'll use in later chapters.

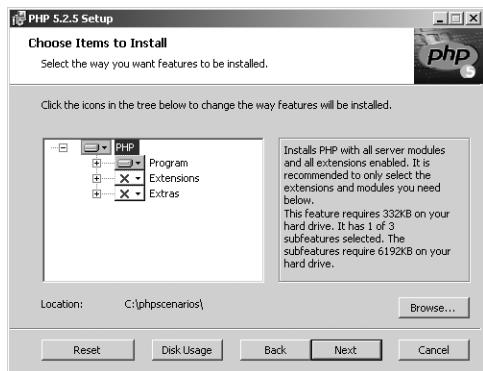


Figure 1-14. Selecting PHP extensions for installation

Optionally, under Extras, choose PHP Manual to install the manual. Alternatively, you can head over to <http://www.php.net/download-docs.php> to download a foreign language version of the docu-

mentation (over 20 languages are available).

Click **Next** to continue, at which point the PHP installation process will begin! Once complete, move on to Step #2.

Step #2. Testing Your Installation

Verifying the installation process went as planned is simple. Open up Notepad or another text editor and create a file named `phpinfo.php`, saving the file within Apache's `htdocs` directory. Within this file insert the following code:

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

In the next chapter I'll explain exactly what this strange syntax means, so don't worry if it's all a mystery at this point. Next, open your browser and navigate to `http://localhost/phpinfo.php`. You should see output similar to that shown in Figure 1-5. If you did, congratulations, PHP and Apache are working together. To verify PHP and MySQL are properly configured, scroll down this page until you see a section titled `mysqli`. If you see this section, PHP's MySQLi extension is properly installed.

NOTE. If you're using Notepad, when saving this file be sure to set **Save as type** setting to **All Files** within the Save As dialog. This will prevent the `.txt` extension from being appended to the filename, which would prevent Apache from being able to properly recognize the file as containing PHP code.



| PHP Version 5.2.3 | |
|--|--|
| System | Windows NT WJG-LAPTOP 5.1 build 2600 |
| Build Date | May 31 2007 09:38:39 |
| Configure Command | <code>ccscript /nologo configure.js --enable-snapshot-build --with-gd=shared</code> |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | enabled |
| Configuration File (php.ini) Path | C:\WINDOWS |
| Loaded Configuration File | C:\php52\php.ini |
| PHP API | 20041225 |
| PHP Extension | 20060613 |
| Zend Extension | 220060519 |
| Debug Build | no |
| Thread Safety | enabled |
| Zend Memory Manager | enabled |
| IPv6 Support | enabled |
| Registered PHP Streams | php, file, data, http, ftp, compress.zlib |
| Registered Stream Socket Transports | tcp, udp |
| Registered Stream Filters | convert.iconv*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.* |

Figure 1-15. Reviewing `phpinfo.php`'s results

Congratulations! AMP is correctly installed! This chapter concludes with an introduction to several prominent code editors and other development tools which will make your job building websites much easier. I'll also offer some advice regarding what to keep in mind when selecting a web hosting

Step #3: Choosing a Code Editor

While a simple program such as Notepad is available for editing text, chances are you prefer to use a word processing application to create all manner of documents, from term papers to resumes, to address labels. The reasons for doing so are many, but it's largely because word processors offer numerous conveniences which allow you to focus on writing rather than the many other important yet distracting tasks required to create a quality document. For instance, among other things word processors tend to take care of matters such as formatting, spell-checking, and offer numerous helper features such as the ability to insert and manage images, hyperlinks, and charts.

Likewise, while you could use a simple text editor to write PHP code, your productivity will undoubtedly rise if you instead opted for a code editor suited specifically to the task. Code editors come with a number of powerful features which will help you write code far faster and more efficiently than simple text editors. In this section I'll introduce you a few of the most popular PHP editors, including my favorite, PHP Developer Tools (PDT).

PHP Developer Tools (PDT)

PHP Developer Tools (PDT - available for download at <http://www.eclipse.org/pdt/>) is arguably the most powerful IDE available to PHP developers, an assertion made even more impressive by its availability free-of-charge for both personal and commercial users. Supported by the three major platforms (Windows, Linux, and Mac OS X), PDT consistently ranks among the most popular download of the many projects managed by the Eclipse open source community (<http://www.eclipse.org/>).

PDT offers PHP developers an impressive number of features intended to maximize your productivity, among them:

- **Real-time error detection:** As syntax errors inevitably creep into your code, PDT will automatically sense and highlight them, denoting the line and typically offering a description of the problem.
- **Code completion and assistance:** Keeping track of the paired quotes, parentheses, brackets, and braces can quickly become tedious, not to mention dealing with the near impossibility of remembering the parameter types and positions of PHP's 1,000+ functions. PDT eliminates the need to keep tabs on such matters by auto-sensing your intent to insert these sorts of paired delimiters and adding the second delimiter for you. It will also sense your intent to insert a function, and offer popup guidance regarding the function's parameters (we'll talk about functions in Chapter 2).
- **Debugging capabilities:** While the aforementioned real-time error detection capabilities help you to easily resolve simple yet elusive syntax errors, coding errors (or bugs) are often much more nuanced than something as straightforward as a missing semicolon or bracket. To help you locate these issues, PDT comes equipped with a debugger which helps you to analyze PHP scripts and track the results on a per-line basis.

- **Project management tools and other add-ons:** Because PDT is built atop the Eclipse framework, PDT users are also able to take advantage of the countless other Eclipse plugins, including notably those intended to facilitate project management. For instance, if you're familiar with source control solutions such as Subversion and CVS, you can integrate Subclipse (a Subversion plugin) directly into PDT, allowing you to take advantage of Subversion from within the PDT interface.

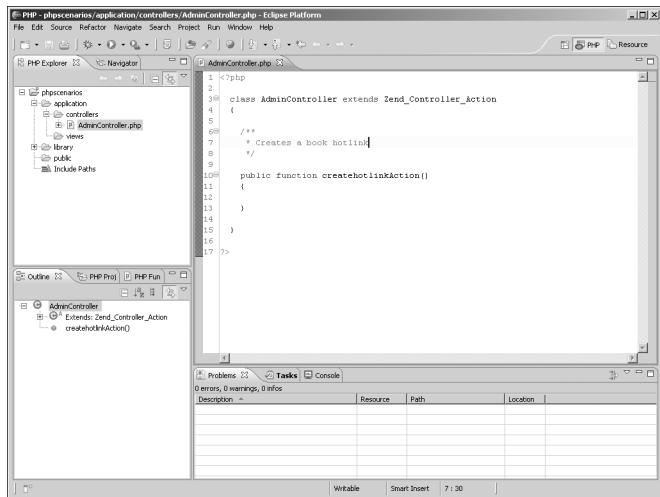


Figure 1-16. Exploring the PDT Interface

TextMate

If you're running Mac OS X and PDT doesn't appeal to you, consider TextMate, considered by many OS X users to be the best OS X editor available. Available from MacroMates (<http://www.macromates.com>) for just \$51 (according to the latest exchange rate, as Macromates is a UK company), and offering a 30-day free trial, TextMate has gained a cult following among its users due to its powerful IDE features, source control integration, and other useful code management features. With support for more than 50 languages, TextMate might be an appealing option if you're an OS X developer in search of a unified code development tool.

Zend Studio

A product of leading PHP-specific products and services provider Zend Technologies, Zend Studio is the most powerful out-of-the-box PHP IDE on the market. Offering built-in coding support for not only PHP, but also for HTML and JavaScript, in addition to powerful debugging and optimization tools, developers are given the advantage of using a unified interface for building both the website's logic and presentational aspects. Additionally, Zend's oversight of the Zend Framework is resulting in ever-improving synchronicity between Zend Studio and the Framework, meaning your organization is likely to reap significant rewards from adopting Zend Studio should you plan on depending heavily on Zend Framework.

At \$399, Zend Studio isn't cheap, but it does come with a one year support and upgrade subscription. If you're looking for a powerful development solution offering the immediacy of an official support line, Zend Studio may well be worth the investment. You can learn more about Zend Studio here: <http://www.zend.com/en/products/studio/>.

Vim

Vim, short for "Vi Improved", is a descendent of the powerful Unix editor "Vi". Available for all modern operating systems, and even a few you haven't seen in years such as the Amiga and MS-DOS, Vim is a great option for users searching for a text editor offering a high degree of configurability while not distracting you with toolbars and other graphical elements.

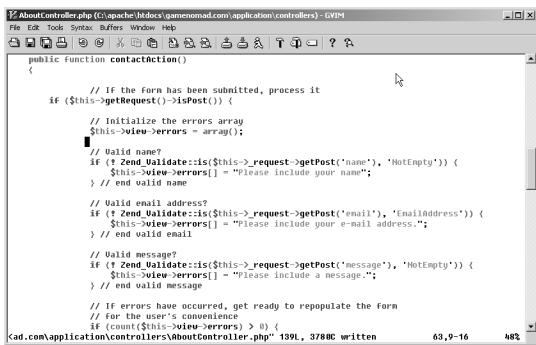


Figure 1-17. Vim's Windows version is decidedly more graphical than its Linux-based counterpart

If you're using Linux or Mac OS X, then it's almost a certainty that Vim is already installed. Just run `vim` from the command line to find out. Otherwise, head over to <http://www.vim.org/> to download the version appropriate for your operating system.

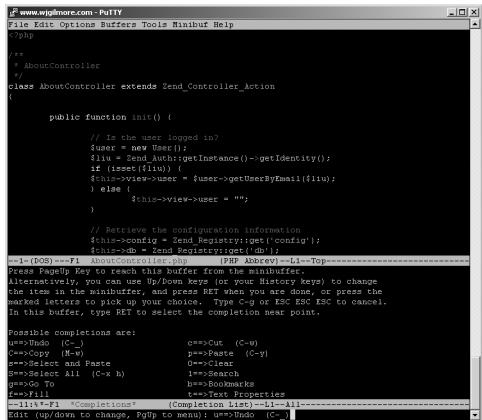
At a minimum, you'll probably want to download the PHP Syntax file, available on the aforementioned Vim website. If you're new to Vim, I also suggest checking out noted PHP community member Andrei Zmievski's excellent tutorial, "VIM for (PHP) Programmers", available at Andrei's website (<http://www.gravitonic.com/>).

Emacs

Emacs is another popular open source text and code editor, dating back to 1976. But don't let its age fool you, if anything its longevity is indicative of the power it offers developers. In fact, devout Emacs users will tell you Emacs is so much more than a typical editor, not only offering support for dozens of programming languages, but capable of acting as your e-mail client, web browser, calculator, calendar, and much more.

Like Vim, the Emacs text editor has long been a stable of the Unix/Linux platform, meaning chances are it's already installed if you're running one of these platforms. Otherwise, if you're running Windows and want to give Emacs a try, head over to <http://ftp.gnu.org/gnu/emacs/windows/>. You'll

want to download the file titled `emacs-XX.X-bin-i386.zip`, where the X's represent the latest version number.



```

www.uglyboore.com - PutTY
File Edit Options Buffers Tools Minibuf Help
C:\php

*/ 
* AboutController
*/
class AboutController extends Zend_Controller_Action
{
    public function init()
    {
        // Is the user logged in?
        $user = new User();
        $fliv = Zend_Auth::getInstance()->getIdentity();
        if ($fliv)
            $this->view->user = $user->getUserByEmail($fliv);
        else
            $this->view->user = "";
    }

    // Retrieve the configuration information
    $this->view->config = Zend_Registry::get('config');
    $this->view->db = Zend_Db::factory($this->config);
--1-(BOS)-->P AboutController.php [PHP About]--L1-Top
Press PageUp Key to reach this buffer from the minibuffer.
Alternatively, you can type Up/Down keys to move between buffers, or press the Tab key to change the current buffer, and press RET when you are done, or press the masked letters to pick up your choice. Type C-g or ESC ESC ESC to cancel.
In this buffer, type RET to select the completion near point.

Possible completions are:
u=>Find D (C-u)          c=>Cut (C-w)
c=>Copy D (M-w)           p=>Paste (C-y)
s=>Select and Paste      t=>Text Properties
g=>Select All (C-x h)     i=>Search
q=>Go To                  b=>Bookmarks
f=>Fill E                t=>Text Properties
All Completions:           Completion C-c C-z All-
Edit (up/down to change, PgUp to menu; u=Undo (C-u)

```

Figure 1-18. Editing a PHP Script in Emacs

With several hundred downloads per week, Aaron Hawley's "PHP Mode for Emacs" plugin (<http://php-mode.sourceforge.net/>) seems to be the most popular of the several PHP plugins available today. You should also check out the useful PHP Mode page at [emacswiki.org](http://www.emacswiki.org/emacs/PhpMode), available at <http://www.emacswiki.org/emacs/PhpMode>.

NOTE. Like Vim, Emacs can take some getting used to. Its seemingly endless array of task-oriented keystroke combinations can be a chore to remember, however once you get used to this dramatically different approach to code editing, chances are you won't want to go back to other solutions. In fact, best-selling science fiction author Neal Stephenson has much to say on this topic, having written a book titled "In the Beginning was the Command Line", which espouses the virtues of a clean editing interface. You can download the book for free from here: <http://www.cryptonomicon.com/beginning.html>.

Step #4. Exploring Other Useful Software

Today's web site developer is blessed with an enormous array of useful utilities which serve to make the development and design process much more efficient than ever before. In this section I'll introduce you to a few of my favorite utilities in the hopes they'll be as beneficial to you as they have to me over the years. In later chapters I'll introduce you to other useful task-specific software.

FileZilla

Although more sophisticated methods exist for transferring code and other assets such as images between your development machine and your web server, FTP remains far-and-away the easiest and most popular way to do so. Using FTP software, you can transfer these files using a drag-and-drop paradigm not dissimilar to what you would encounter when copying files from one Windows folder to another.

Because FTP is so popular, there are dozens, if not hundreds of available FTP clients. My personal favorite is an open source client named FileZilla (<http://filezilla-project.org/>). FileZilla offers all of the features you'll desire in an FTP client, including the ability to manage multiple servers, connect securely using encryption, and easily recover from failed transfer sessions.

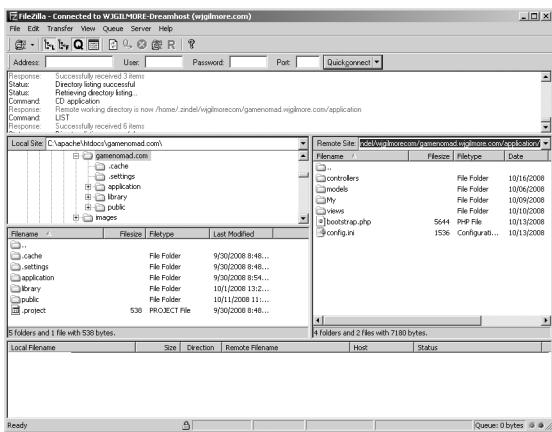


Figure 1-19. The FileZilla FTP client

Freemind

Project management experts have long advocated the strategy of brainstorming project details using a variety of exercises such as scribbling down ideas on sticky notes and organizing them on the office wall or cubicle. In today's paperless, distributed office, users desire an updated approach to this otherwise timeless strategy. One such solution is FreeMind, a free-association tool which helps users organize and revise project-related thoughts and ideas.

Like FileZilla, FreeMind is open source, and available for download from SourceForge (<http://freemind.sourceforge.net/>). Unlike FileZilla, it's available for a variety of platforms, including Windows, Linux, and Mac OS X.

I'm an avid user of FreeMind, and in fact used it to sketch out my preliminary thoughts regarding this book project. You'll see the project map (known in Freemind parlance as a *mind map*) in Figure 1-20.

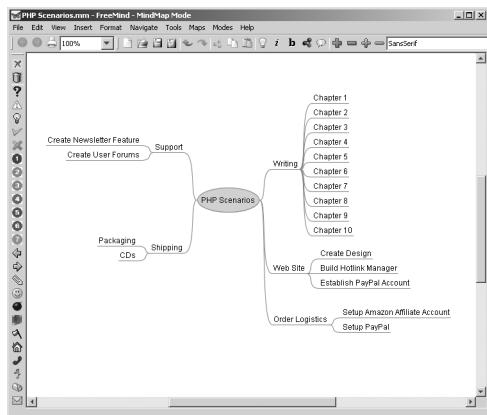


Figure 1-20. Mapping a project with Freemind

The Firefox Web Developer Add-on

It seems every profession has one dispensable tool that without it, greatly hinders one's ability to properly do their job. For instance, an accountant would be hard-pressed to complete a tax return without a calculator, and a plumber would have a hard time repairing a leak without a pipe wrench. Likewise, the web developer has the Firefox Web Developer Add-on.

Offering a web-based interface for quickly and easily examining a seemingly countless array of web page information, the Firefox Web Developer serves an invaluable debugging role. Among other features you can determine the size in pixels of a specific page element, display the dimensions of any images found on the page, outline tables, and quickly access a page's JavaScript and CSS. Figure 1-21 presents just one of this extension's hundreds of features, an option for overlaying form details atop the rendered HTML.



Figure 1-21. Viewing form details with the Firefox Web Developer Add-on

Download the Firefox Web Developer add-on by navigating to <https://addons.mozilla.org/>, searching for the plug-in by name, and clicking the *Add to Firefox* button. Doing so will start the add-on installation process. Once complete, the browser toolbar will be displayed at the top of the browser window. As the name implies, this tool is only available for the Firefox browser, so if you haven't yet installed the browser be sure to do that first.

Step #5. Choosing a Web Hosting Provider

Although there's no need to make the investment right now, at some point you're going to need to seek

out and choose a web hosting provider. This provider will be tasked with reliably hosting your site, in addition to providing you with a variety of tools for effectively accessing and managing the site. Because the provider is effectively the conduit to your users, it is absolutely crucial that you take the time to research and choose appropriately. Indeed, it's so important that I've opted to close this chapter with a special section on the topic in the hopes it will help bring you up-to-speed regarding key hosting terminology and prepare you to ask the right questions when the time comes to begin seeking out a provider suitable to your own needs.

NOTE. You're probably wondering what expenses are involved in using a hosting provider. Over the years I've worked with dozens of hosting providers ranging from the very cost-effective shared-hosting providers (less than \$10/month) to expensive dedicated hosting providers (in excess of \$400/month). While those in the upper echelon of pricing typically offer excellent personalized support and troubleshooting services not otherwise realistically available among the lower cost providers, my experience has nonetheless shown that when chosen carefully, low cost shared providers can offer a tremendously positive and efficient services.

Shared vs. Dedicated vs. VPS Web Hosting

These days hosting solutions are typically classified as being dedicated, shared, or a virtual private server:

- **Dedicated Hosting:** A dedicated hosting solution grants all of a server's resources to a single client. The client is often granted complete control over this server's operation, including installing and upgrading software and even rebooting it as necessary. As you might imagine, dedicated hosting solutions are typically the most expensive.
- **Shared Hosting:** Shared hosting solutions require clients to share a single server alongside several other customers, which can occasionally number into the thousands. If you're running a fairly simple website with little need for significant server resources or bandwidth, shared hosts are often a great solution given the low pricing arrangements.
- **Virtual Private Server Hosting:** Virtual private servers are the relative newcomer, combining the best attributes of both dedicated and shared hosting into a single solution. While clients share a single server, thus lowering the hosting cost, each client is granted an independent partition on that server which not only often comes with the client's ability to autonomously manage and even reboot the server, but also shields each client from being negatively affected by the actions of other clients on that server, both from a resource- and security-specific standpoint.

Unless you require particularly esoteric or resource-intensive environment, chances are shared hosting or virtual private server hosting will serve your purposes just fine. If you're comfortable installing server software and otherwise managing your own server, then you'll find the latter solution much more appealing. Otherwise, if your sole goal is to launch a website, then shared hosting might be the way to go. Whatever you decide, you'll be pleased to know that given the withering competition in this area, the customer is clearly in control of the purchasing situation, with hosting companies often allowing you to migrate from one service tier to the next as your needs change.

Vetting a Hosting Provider for S-U-C-C-E-S-S

Beyond determining the obvious, such as whether the hosting provider actively supports PHP and MySQL, there are several other hosting factors you'll want to keep in mind when searching for an ideal provider. I call this my checklist for S-U-C-C-E-S-S, consisting of seven key items:

- **Service Tiers:** Rather than offer a one-size-fits-all solution, most hosting providers will divide their offerings into service tiers, with each tier offering a select set of server resource, bandwidth, and support options. Don't be afraid to start small as most providers will allow you to easily switch from one tier to the next.
- **User Support:** Hosting providers often offer a wide array of support options, ranging from hands-on troubleshooting, to phone, chat, and e-mail). Be sure to identify the levels supported by your candidate providers, and understand any fees that might be involved for more personalized support levels.
- **Community Support:** Web hosting providers such as Dreamhost (<http://www.dreamhost.com>) are lauded for their expansive community support options, including wikis, forums, and chat rooms. Investigate these options beforehand to determine the level of activity, as your fellow developers can often offer invaluable advice regarding deploying and maintaining web sites on the particular hosting provider's platform.
- **Contract Flexibility:** Many providers will offer a 10% or greater discount in exchange for paying for a year or even a quarter of services in advance. Therefore consider saving some money by investing for the long term, particularly if the provider also offers a no-questions-asked contract termination policy, complete with a pro-rated refund.
- **E-mail Options:** In today's 24/7 business environment, the ability to access e-mail anytime and from any device is crucial. Most, if not all providers offer IMAP capabilities, meaning you'll be able to synchronize your e-mail among several clients (for instance your laptop and PDA), in addition to providing some sort of webmail solution. Be sure to inquire as to which options are available, in order to ensure they'll fit your organization's expectations.
- **Service Migration:** Be sure to ensure the provider offers an easy upgrade/downgrade path to nearby service tiers. Being able to quickly upgrade will be crucial should your website suddenly experience a surge in popularity. Likewise, if you've overestimated service needs, you should be able to easily downgrade to a lower service tier without penalty, thereby saving on monthly expenses.
- **Solid Track Record:** Although the industry quality has improved significantly in recent years, there remain a number of hosting providers with extraordinarily negative customer satisfaction ratings. Therefore be sure to thoroughly investigate the hosting provider's track record before signing on, notably by asking existing clients for frank assessments. Also, if you have a friend or colleague already running a website, don't be afraid to ask them where they're hosting! Not to mention many hosting providers offer "refer-a-friend" discounts for both the referrer and new client, so in the process you just might save yourself a few dollars by signing on.

Taking Advantage of Google

Most web hosting providers combine web and e-mail hosting solutions under a single offering. However, as e-mail has become an increasingly crucial part of doing business, some organizations have started opting for e-mail hosting providers dedicated exclusively to the practice of effectively managing your e-mail-related services. If you foresee unusually high e-mail volume or otherwise desire a dedicated mail hosting solution, doing some investigation may well be worth your time.

One of the least known newcomers in this area is Google, who recently started offering small and large businesses an array of e-mail hosting solutions. In addition to being able to use the typical e-mail clients such as Outlook and Thunderbird, you can access your e-mail over the web using Google's popular web-based e-mail client, and also the web-based mobile client. Incredibly, if you and your colleagues don't mind working in an ad-supported environment, the service is free! Otherwise, if you're looking for a particularly robust mail environment complete with e-mail archiving and a larger storage allocation (25 GB per account vs. 7.25 GB for the free edition), you can contact Google regarding Premier Edition pricing.

E-mail hosting services is just part of Google's push into the business environment; in fact e-mail makes up just one part of the larger package, which includes calendaring, document hosting, and even web hosting. For more information, see <http://www.google.com/a/>.

Conclusion

We covered a tremendous amount of ground in this opening chapter, providing instruction and advice that you'll undoubtedly immediately begin putting into practice as you delve deeper into this book, in addition to identifying topics such as web hosting that you'll return to as necessary.

In the next chapter you'll be introduced to the PHP language, not only creating your first PHP-enabled web page, but also actually creating your first practical website, namely one which publishes information about your video game collection.

CHAPTER 2

Introducing PHP

With your development environment in place, what's next? The most logical next step is to begin learning PHP, but you won't be doing so by way of an exhaustive review of language rules, constructs, and syntax. Instead, we're going to learn the language's fundamentals in a much more interesting way: by taking a first crack at building a website capable of displaying information about our video game collection.

This chapter is particularly applicable because it shows you how to implement a commonplace desire of newcomers to Web development. Many budding Web developers are borne out of a desire to want to make data found in a spreadsheet available for viewing on a Website. Of course, one could simply upload the spreadsheet to a Web server, and require visitors to download a file, but this method is inconvenient and requires interested visitors to have software such as Microsoft Excel installed on their laptop. What if you could instead upload the spreadsheet to the web server and use a PHP script to parse the spreadsheet and integrate the data into a web page complete with a visually-appealing design? This is the focus of the second chapter, which will use this project as a vehicle for teaching you the fundamentals of the PHP language.

Chapter Steps

The goals of this chapter are accomplished in three steps:

- **Step #1. Creating Your First PHP-enabled Web page:** In your first step you'll learn how to integrate PHP code into your web pages. This step is particularly crucial as in it you'll be provided with a short tutorial on PHP's fundamental syntax and features which we'll use repeatedly in subsequent steps and in other chapters found throughout this book.
- **Step #2. Publishing Spreadsheet Data to the Web:** In this step you'll learn how to convert an Excel spreadsheet containing your video game collection into a format capable of being read by a PHP script. This script, which we'll also create in this step, will parse the spreadsheet data and display it to the browser in a user-friendly format.
- **Step #3. Managing Your Site Design Using Templates:** While simply displaying a directory serves a practical purpose, chances are you're going to want to spruce up the page a bit by incorporating a few design elements. In order to maintain design consistency and enforce maintainability as the website grows you'll likely want to convert these design elements into templates and then dynamically integrate them into your website. This step shows you how.

Step #1. Creating Your First PHP-Enabled Web Page

PHP is a particularly attractive programming language because useful scripts can be written in as little as one line. Remember how in the last chapter we used the following script to verify Apache and PHP were installed correctly:

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

Believe it or not, this is a valid PHP script! The `<?php` and `?>` characters are called opening and closing PHP tags, used to delimit the PHP code so the PHP engine knows what it should execute when the web server hands over the page. Found between the tags is a single line of PHP code, in this case a PHP function named `phpinfo()`. *Functions* are a common feature of mainstream programming languages, and are nothing more than a reusable piece of code used to perform a task which you might need to perform repeatedly within an application. Functions are typically assigned easily recognizable names, allowing you to simply call the name in order to execute the corresponding code. Later in this section I'll show you not only how to learn more about the many other PHP functions at your disposal, but also create your own.

You might also recall we named this file `phpinfo.php`. You could have actually named it anything you please, provided the `.php` extension was used. For example, `jason.php`, `games.php`, and `45jabba.php` are all perfectly valid names. The `.php` extension is important because this tells the web server that PHP code is likely found inside, and it should pass the file to the PHP engine for review before serving the requested document back to the user's browser.

Incidentally, because PHP doesn't take line breaks nor indentation into account, you could also write the aforementioned script like this:

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

You use the very same approach to create your own scripts. Try creating the following page and saving it as `index.php`, placing the document within your Web server's document root (the server document root was defined in Chapter 1):

```
<?php  
    echo '<h1>Welcome to My Game Collection!</h1>';  
?>
```

Now open your browser and call this script using the following URL (presuming you're executing this on your local machine, rather than on a hosted server; if the latter you'll need to supply the appropriate domain name and path):

```
http://localhost
```

Presuming you typed in the script exactly as shown above, a web page consisting simply of the sentence *Welcome to My Game Collection!* will be rendered to the browser, as shown in Figure 2-1.

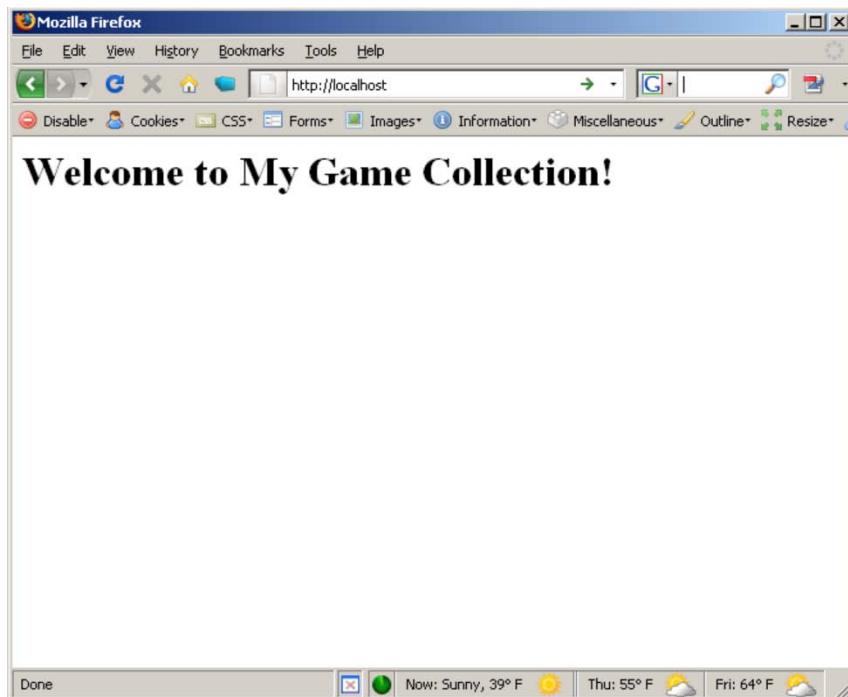


Figure 2-1. Your first PHP-enabled page

The `echo()` statement is one of several options PHP provides for outputting data. You'll often see other solutions widely used, such as the `print()` and `printf()` statements, however as you'll soon see, `echo()` offers a perfectly practical solution for outputting all sorts of data.

VIDEO. Creating Your First PHP Page

One of PHP's most compelling characteristics is the ease in which you can get started using the language. But figuring out how to create that first script can sometimes be confusing. This video shows you how to create your first script, where to place the script so Apache can find it, and how to call the script from within your browser. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Adding Dynamic Data to Your Web Page

But you could just as easily have created a `games.html` page to perform this task, right? Indeed you'll typically want to use PHP only in circumstances where dynamic data is going to be displayed, such as the date. Let's revise `index.php`, this time using PHP's `date()` function to output the current date:

```
01 <h3>Welcome to My Game Collection!</h3>
02 <?php
03   $date = 'March 11, 2009';
04   echo "Today's date is {$date}.";
05 ?>
```

Reload your browser and you'll see output similar to what's shown in Figure 2-2.

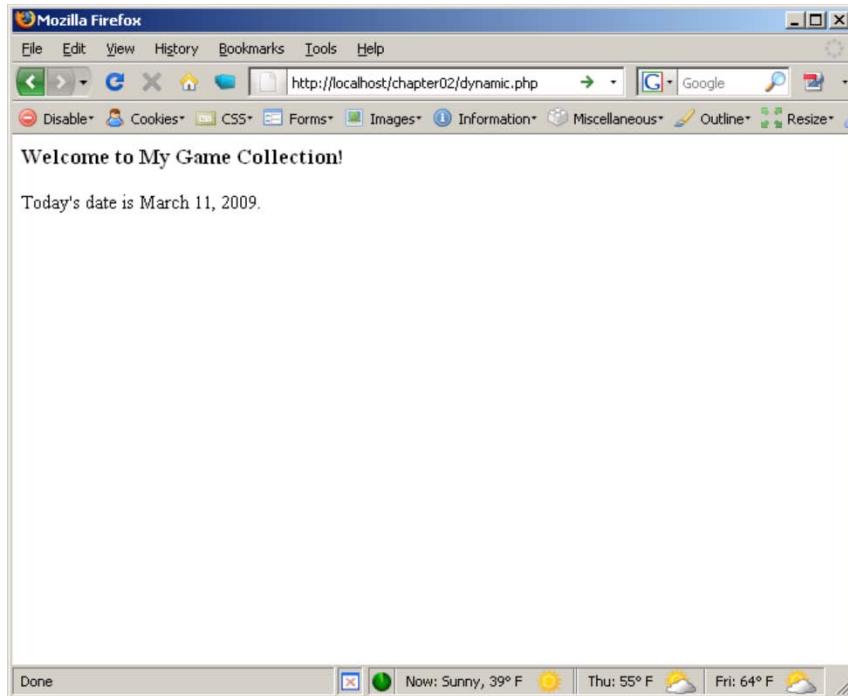


Figure 2-2. Rendering dynamic data to a web page

This example demonstrates several important concepts:

- PHP code can be embedded directly alongside static HTML. In fact, you're free to escape to and from PHP as many times as you please, a practice which you'll see in later chapters is actually quite common.
- Line 03 defines a variable named `$date`, and assigns the string `March 11, 2009` to it. A *variable* is a feature found in many mainstream programming languages which allows the programmer to conveniently reference a potentially dynamic piece of data. We'll use variables so many times in the coming examples found in this chapter and others that any lack of understanding you have regarding their usefulness will soon disappear.
- Line 04 outputs the newly created `$date` value to the browser, alongside a static bit of text (referred to in a computer science as a *string* type) that puts the date into context. This example demonstrates how static and dynamic data can be intermingled within the `echo()` statement. By enclosing the variable within curly brackets, we can be sure PHP understands it's to convert the variable to its corresponding value. While PHP will also typically recognize the variable even when not surrounded by curly brackets, its ability to do so will diminish as more complex types of dynamic data is used, therefore putting this approach into practice as early as possible is the best way to go. Again, if you don't completely understand the thinking behind this approach, not to worry as it will all become apparent as the examples progress in complexity throughout the book.

The observant reader will notice a slight variation between this and the previous example in terms of the use of quotes. Namely, why did I use double quotes to output the string this time, whereas in the previous example I used single quotes? As this is something which has confounded beginning PHP developers for years, it's best to clarify on the matter immediately. As a rule of thumb, you should use single quotes when there's no need to interpret the quoted contents. You should use double quotes when the data found between the quotes needs to be interpreted by PHP, for instance if a variable is found within. The only caveat here is if your single-quoted string contains a single quote, you'll need to escape it using a backslash. There are certainly ways to ignore this approach, but for the most part applying this strategy is the best way to eliminate any confusion as to what the intent of any delimited string.

Working With Different Types of Data

Programming is an exacting sort of vocation, often requiring you to be very specific regarding your intentions lest unexpected side effects or even errors occur. To help clarify your intentions, most mainstream programming languages, PHP included, support numerous data types such as strings, integers, floating point numbers, and even the Boolean (true/false) type.

However, unlike programming languages such as C++, PHP is quite lax when it comes to formally assigning a type to data. For example, in C++ you must explicitly assign a type to a variable at declaration time, like so:

```
string date = 'March 11, 2009';
```

However as you saw in earlier examples, PHP requires no such specification:

```
$date = 'March 11, 2009';
```

Instead, PHP will automatically attempt to determine a variable's type based on its contents. For example, it knows the following variable is of type Integer simply based on how you assigned it and on its contents:

```
$gameCount = 45;
```

It treats the following variable as type Float (the Computer Science term meaning decimal values) because of its contents:

```
$averagePrice = 12.45;
```

However, it will treat both of the following variables as type String because you enclosed the numbers in quotations:

```
$gameCount = '45';
$averagePrice = '12.45';
```

Why does this matter? Because although you can forego the rigorous type definition requirements imposed by other languages, you'll also need to be more cautious when working with variables as unexpected side-effects could occur. For example, what do you think would happen if you tried to add

```
echo '45.4' + 12;
```

It will output 57.4! PHP is able to divine your intention by presuming the plus sign is indicative of your intention to add the two values together, despite one of them being a string.

In a later section you'll learn about another data type called array, and in later chapters you'll learn about a truly powerful type known as the object.

The Power of printf()

Earlier in this section I mentioned the `printf()` statement, another common function for outputting data. The `printf()` function is very useful not only because it can not only result in greater code readability, but also because it offers a number of useful formatting features. The following list identifies a few of the more commonly used type specifiers:

- `%d`: Integers such as 45, -8, and 687
- `%f`: Floating point numbers such as 73.59, 0.98, and -45.67
- `%s`: Strings such as 'PHP', 'I love programming', and 'The class of 1993 ruled!'

In previous examples you saw how the string type specifier (`%s`) was used to work with a single string-based variable. But how would you use `printf()` to output multiple string variables, or multiple variables of different types? A few examples follow:

```
printf("%d games cataloged as of %s!", $total, $date);
# Sample output: 45 games cataloged as of November 13, 2008!
```

```
printf("Expect to pay on average $%f at this video game store!", $averageCost)
# Sample output: Expect to pay on average $22.99 at this video game store!
```

Other specifiers exist for managing other formatting characteristics of the variable data, including among others whether the data should be left- or right-justified, the precision of floating-point numbers. Consult the PHP documentation for a complete breakdown of what's available.

Working with Groups of Data (Arrays)

Much of your time as a programmer will be spent working with groups of like data; U.S. states, employee e-mail addresses, salaries, or pizzeria names, you name it, chances are you'll at one point need to analyze, output, or otherwise manipulate data groups such as these. PHP provides a data structure known as an *array* which makes it easy to manage data groups. For example, you can create an array consisting of three games like so:

```
$games = array('Halo 3', 'Super Mario Bros.', 'NBA 2K7');
```

For purposes of readability you're free to place each element on its own line, like so:

```
$games = array('Halo 3',
    'Super Mario Bros',
    'NBA 2K7');
```

Each array entry, also known as an element, can then be accessed according to its defined positional offset in the array. Regarding the `$games` array, you might logically think the offset of 'Halo 3' is 1, however arrays have a funny behavior of treating the first element as the 0 offset, meaning you'd actually have to reference the first element like so:

```
echo {$games[0]};
```

To retrieve 'NBA 2K7', you'd use an offset of 2, like so:

```
echo {$games[2]};
```

Likewise, if you decided to swap 'NBA 2K7' for 'Saint's Row', you could modify the array like so:

```
$games[2] = 'Saint\ 's Row';
```

You can output an array's contents using the function `print_r()`, like so:

```
print_r($games);
```

Executing this command will dump the array's contents to the browser:

```
Array ( [0] => Halo 3 [1] => Super Mario Bros. [2] => Saint's Row )
```

While not pretty, the `print_r()` function can be very useful for testing purposes. In the later section "Looping Constructs" you'll learn how to wield much more control over displaying arrays to the browser.

Associative Arrays

The array elements discussed in the previous examples are accessible according to a numerical offset, but what if you wanted to tie a meaningful index value to a corresponding value? For instance, what if you wanted to manage a list of U.S. states abbreviations and their corresponding names? This sort of array is known as an associative array, and it's easily accomplished in PHP. For example:

```
$states = array('CA' => 'California', 'OH' => 'Ohio', 'NY' => 'New York');
```

You could then retrieve the name of any state in the array based on its abbreviation, like so:

```
echo {$states['OH']};
```

Multi-dimensional Arrays

The arrays you've seen so far are single-dimensional; that is, there's only one key/value pair per element. But what if you wanted to store another array within an array element? This is known as a

multi-dimensional array, and has long been supported by PHP. For example, suppose you wanted to track the platform for each game in your collection within a multi-dimensional array:

```
$games = array (
    array('title' => 'Halo 3', 'platform' => 'XBOX 360'),
    array('title' => 'NBA 2K7', 'platform' => 'Sony Playstation 3'),
    array('title' => 'Saint\'s Row', 'platform' => 'XBOX 360')
);
```

You could then retrieve for instance Halo's platform using the following reference:

```
echo {$games[0]['platform']};
```

Commenting Your Code

Whether for your own guidance, or for the benefit of other programmers who may work with you in the future, you should always take care to comment your code in detail. PHP offers two ways to do this.

One-line Comments

Sometimes you just want to insert a quick note into the code, perhaps briefly describing the line that follows, or adding a personal reminder to complete some task. This is easily done with a one-line comment, and there are two methods available for doing so. Both of the following lines are valid comments:

```
# Print game title to the browser
// Print game title to the browser
```

The variations are available simply to accommodate the commenting preferences of programmers hailing from other languages, notably Perl and C++, respectively. So what's the difference? Absolutely nothing, since the PHP parser will ignore both. What's important is you choose one of the variations and stick with it.

Multi-line Comments

If you'd like to add a somewhat more lengthy comment which encompasses several lines, PHP offers a somewhat more suitable method than those intended for single-line comments. Known as the multi-line, or C-style variation, it delimits comments within /* and */, allowing you to stretch comments across numerous lines, like so:

```
/** 
 * Script: games.php
 * Author: W.J. Gilmore, LLC
 * Description: Formats the game directory spreadsheet
 */
```

Taking Advantage of Functions

Earlier in this chapter you learned what a function is: a bit of code packaged in such a way that it can be repeatedly executed by referring to its defined name. For example, in this chapter you've already encountered the `printf()` function. As you might imagine, PHP offers many more functions capable of performing a seemingly countless number of tasks, ranging from rounding fractions down to counting the number of characters in a string, to redirecting a visitor to another website. To obtain a better idea of just how many functions are at your disposal, take some time to peruse PHP's manual.

In this section you'll become better acquainted with functions by formally introducing a function you'll repeatedly return to when building PHP-powered websites.

Introducing the date() Function

The `date()` function is very powerful, allowing you to format not only the current date, but also the current time in a wide variety of ways. For example, presuming today's date was July 4, 2009, among many other formats you could easily output the current date and time using any of the following formats:

```
July 4, 2009  
07/04/2009  
07/04/09  
Saturday  
Sat, 04 July 2008
```

The `date()` function can also output the current time in a wide variety of formats such as:

```
08:15 pm  
20:15  
11:40 am EST  
11:40 am -0500
```

Although not mandatory, many functions accept one or more input parameters which help determine how the function behaves. Indeed, the `date()` function accepts two, a required parameter which tells it how to return the date and/or time, and an optional parameter which tells it to format a specific date and time in accordance with a supplied timestamp (more about what a timestamp is in a moment). Additionally, a function will always return a value, even if that value is nothing (or *void*, as they say in computer science-ese). In the `date()` function's case, it returns a string, consisting of the desired formatted date and time.

NOTE. Typically you'll learn about a function by consulting the PHP manual. Incidentally, if you already know the function's name you can navigate directly to it by using the following URL: <http://www.php.net/function>, where `function` is a placeholder for the name of the function you'd like to consult. Therefore to consult the `date()` function's page, navigate to <http://www.php.net/date>.

When consulting the PHP manual to learn more about a specific function, keep in mind that each function's page clearly identifies the return value, and which input parameters are optional by enclos-

ing them within square brackets. For instance if you navigate over to <http://www.php.net/date> you'll see the `date()` function defined like this:

```
string date( string $format [, int $timestamp] )
```

So what do you assign the `$format` input parameter in order to specify what sort of date/time it should return? This is done by assigning format specifiers to the parameter in a certain order. Format specifiers are simply a variety of different characters, with each having a specific meaning. For example, to have `date()` return just the current day of the week, you would assign the lowercase `l` to `$format`, like so:

```
echo date('l');
```

Presuming today is Tuesday, it will return:

```
Tuesday
```

If you're just looking for the current month, you use the `F` character:

```
echo date('F');
```

Let's consider a more involved example. Suppose you want to return the current date using this format: March 11, 2008:

```
echo date('F j, Y');
```

Finally, let's return the current date and time using this format: 11-13-09 2:47 pm:

```
echo date('m-d-y g:i a');
```

So how do you keep all of these parameters straight in your head? The simple answer is, you don't. The PHP manual is always just a few keystrokes away, and you can even download a copy to your computer by navigating to <http://www.php.net/docs.php>.

Hopefully this brief section has provided you with some insight into not only how to use functions, but also into one of PHP's most popular features, `date()`. Try building upon what you've learned here by experimenting with some of my other favorite functions, including `strtoupper()`, `str_word_count()`, and `pi()`. Remember you can easily look these up by appending the function name to <http://www.php.net/>!

Defining Your Own Functions

While the PHP language is bundled with an amazing array of diverse functions, sooner or later you'll want a function capable of carrying out a task very specific to your own needs. For example, suppose some future version of your gaming web site started selling copies of used video games, and because your business is run out of Ohio you need to calculate sales tax for Ohio-based purchasers. This sort of task might be used at several locations throughout your website, so creating a function capable of calculating the sales tax seems like a good idea. The following listing demonstrates how such a func-

tion is created and subsequently called:

```
01 <?php
02
03     function calculateSalesTax($price, $tax) {
04         return ($price * $tax) + $price;
05     }
06
07     echo "Total cost: {calculateSalesTax(9.99, .0575)}";
08
09 ?>
```

Let's review this script:

- Line 03 begins the function definition, assigning a name and identifying any input parameters.
- Within the function body (line 04) you'll see the total cost is calculated, with the `return` keyword used to return that value to the caller.

Of course, you're not constrained to immediately output the returned value. For instance, if you wanted to assign the return value to another variable, perhaps in order to perform additional processing, you could call `calculateSalesTax()` like this:

```
$total = calculateSalesTax(9.99, .0575);
```

Conditional Statements

Most computer programs are like flowcharts, with their execution dependent upon a select set of conditions. In order to determine which route the program will take, *conditional statements* are often used. These statements are used to validate a given condition, such as what rating applies to which definition. For example, suppose you wanted to offer a user-friendly message based on the video game's rating:

```
if ($rating == 'T') {
    echo 'Rated T for Teens!';
}
```

Notice I used two equal signs here. This is because we're comparing `$rating` to 'T' rather than assigning `$rating` the value of 'T'.

Of course, you might want to offer an alternative message to the visitor in case the rating isn't available. This is easily accomplished with the if-else statement:

```
if ($rating == 'T') {
    echo 'Rated T for Teens!';
} else {
    echo 'No rating available!';
}
```

As a final example, what if you wanted to check a value against more than two conditions? For instance, you could use PHP's `if-elseif-else` conditional statement to create a custom suggestion based on a game's rating and whether it has support for multiple players:

```
if ($rating == 'T' AND $multiplayer == 'Y'){
    echo 'Suitable for playing with your friends after school!';
} elseif ($rating == 'M' AND $multiplayer == 'Y') {
    echo 'Bring your coworkers together with a night of gaming!';
} else {
    echo 'Sorry, no suggestion available for this game!';
}
```

Looping Statements

Even the simplest of web applications will likely require you to iterate over a set of data. For example, in the earlier introduction to arrays you learned how to selectively output array elements, but what if you wanted to output all of them? Explicitly identifying each array index in an `echo()` statement wouldn't make much sense, so instead you'll logically want an easier solutions for doing so. PHP's looping statements offer that solution.

In this section I'll introduce you to the three most popular looping statements, although you're encouraged to consult the PHP manual to review other less commonly used alternatives.

The while Loop

The `while` loop iterates until a certain condition is met. For example, the script shown next (`while1.php` in your code download) will output "I love video games!" ten times.

```
01 <?php
02
03     // Set a looping counter
04     $count = 1;
05
06     // Output 'I love video games!' until the counter reaches 10
07     while ($count <= 10) {
08
09         echo 'I love video games! <br />';
10
11         // Increment the counter
12         $count++;
13
14     }
15 ?>
```

Let's review this script:

- Line 04 sets a starting point for subsequently comparing `$count` to 10. Because the `while` loop condition (line 07) specifies it will execute until `$count` equals 10, the loop will execute a total of ten times.

- Line 12 increments the counter using the increment operator (`++`). This is just a shortcut for incrementing an integer by 1. For instance, you could alternatively use the longhand version of this line: `$count = $count + 1`. Similarly, a decrement operator (`--`) exists for decrementing an integer value by 1.

As you might have guessed, executing this script produces output identical to that shown in Figure 2-3.

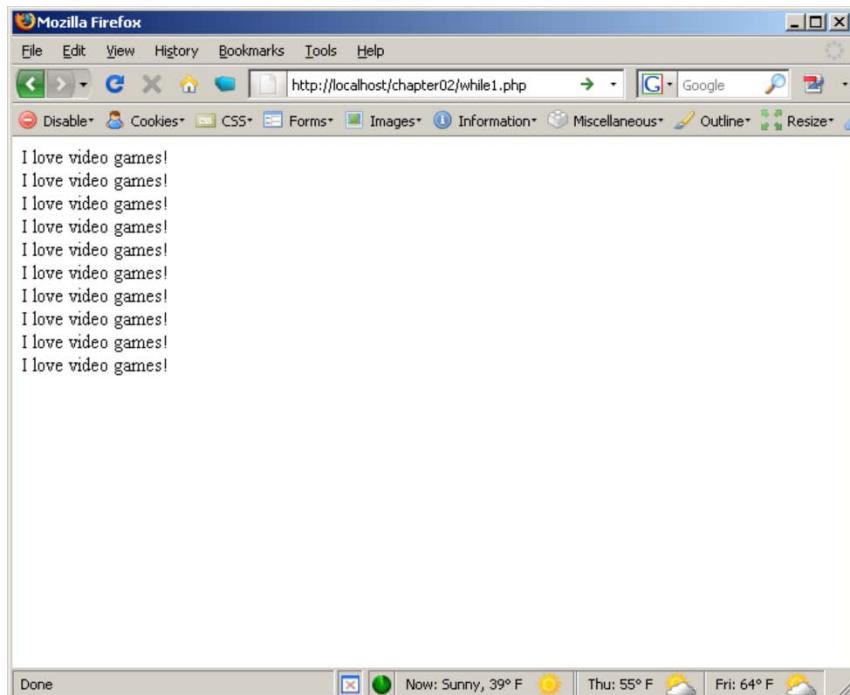


Figure 2-3. Using a while loop

Let's consider a more practical example. Recalling the `$games` single-dimensional array created in the introductory section on arrays, suppose you wanted to output each game name to the browser. The counter would logically be equivalent to the size of the array, which you can determine using the `count()` function. The next example demonstrates how to use `count()` alongside an introduction to using the while loop to output the `$games` array:

```

01 <?php
02
03     // Create the array
04     $games = array('Arkanoid Live!', 'Tekken 6', 'Duke Nukem 3D');
05
06     // Start the counter at 0 in order to retrieve the first element
07     $count = 0;
08
09     // Loop through the array until the end is reached
10     while ($count < count($games)) {
11         echo "{$games[$count]}<br />";

```

```
12     $count++;
13 }
14 ?>
```

Let's review this script (`while2.php` in your code download):

- Line 10's conditional statement might leave you scratching your head. Why are we only looping while `$count` is less than the size of `$games`, rather than until it's equal to the array size? Remember that PHP considers array offsets to begin at 0, meaning when `$count` is equal to `count($games) - 1`, we've reached the end of the array.

Executing this script produces the output shown in Figure 2-4.

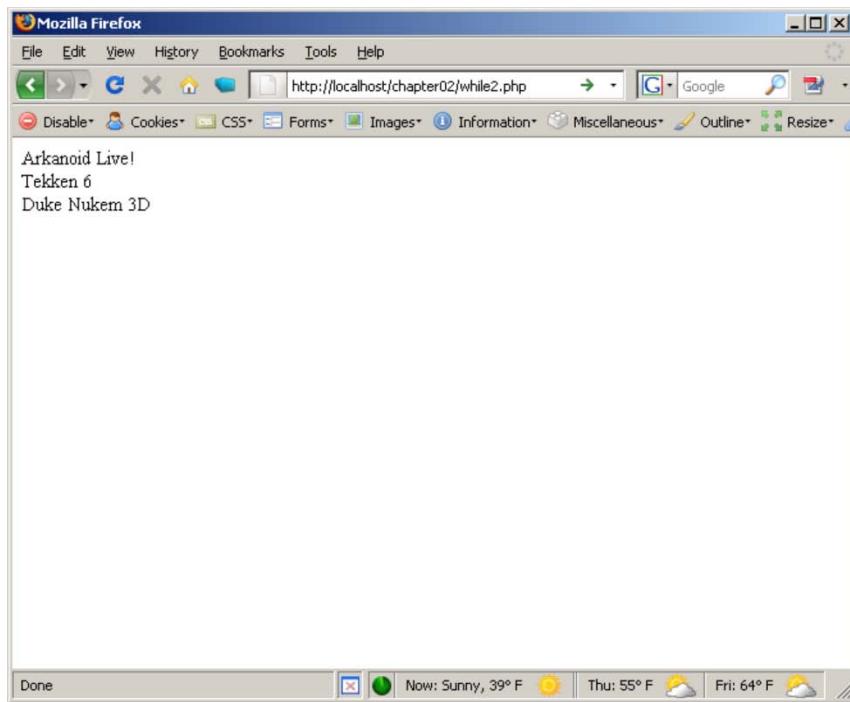


Figure 2-4. Displaying array elements using the `while` loop

The for Loop

The `for` loop accomplishes the same task as `while`, looping over a group of statements until a certain condition is met. The only difference is the counter is managed alongside the conditional statement rather than within the looping body. Whether you choose `while` or `for` is simply a matter of preference.

As an example, the following script (`for.php`) outputs the `$games` array in the same fashion as was demonstrated in the previous example:

```
01 <?php
03
04     // Create the array
05     $games = array('Arkanoid Live!', 'Tekken 6', 'Duke Nukem 3D');
07
08     // Loop through the array until the end is reached
09     for ($count = 0; $count < count($games); $count++) {
10         echo "{$games[$count]} <br />";
11     }
12
13 ?>
```

Let's review the script:

- The `for` loop ultimately produces identical results to a similarly constrained while loop, only that the initial value, condition, *and* iterator are packaged into the loop arguments. So in this case `$count` is first set to 0 (this first argument executes only once). Subsequently with each iteration the second argument is executed, followed by the execution of the third. If the second evaluates to true, the loop body will execute.

The output of this script is identical to that shown in Figure 2-4.

The foreach Loop

You learned how the `for` loop consolidates a bit of code otherwise required when using a `while` loop by packaging the iterator and conditional into a single line. However when iterating over an array there exists yet another looping mechanism which can further cut down on this code.

The `foreach` looping statement will iterate over an array, automatically retrieving the next element each time it loops. For example, the following `foreach` loop reproduces the same output as that shown in Figure 2-4 when passed the same array:

```
01 <?php
02
03     foreach ($games as $game) {
04         echo "{$game}<br />";
05     }
06
07 ?>
```

The `foreach` loop also offers a secondary syntax useful for peeling keys and their corresponding values from associative arrays. For instance you might recall the `$states` associative array from earlier in this chapter:

```
$states = array('CA' => 'California', 'OH' => 'Ohio', 'NY' => 'New York');
```

The `foreach` statement can iterate through this array, retrieving each key and its value, using the following syntax:

```
01 <?php
02     foreach($states AS $key => $value) {
03         echo "{$value} ({$key})<br />";
04     }
05 ?>
```

Executing this example produces the output shown in Figure 2-5.

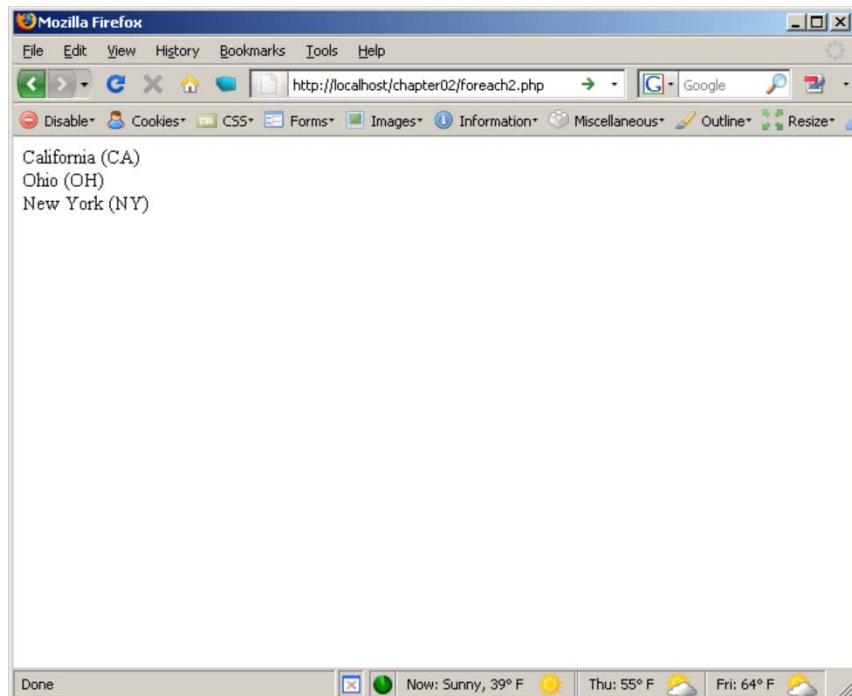


Figure 2-5. Using the `foreach` Loop

Step #2. Publishing Spreadsheet Data to the Web

Suppose your idea for building a website showcasing your video game collection sprung from the frustrations encountered when attempting to keep friends informed about your games and which are currently being borrowed. So far you've been managing the game collection in a simple spreadsheet, with each row containing information about a specific game. For each game you've cataloged the title, platform, price, whether a friend is borrowing it, and a few words regarding your general opinion about the game. A sample of what the spreadsheet looks like is found in Figure 2-6.

| Title | Platform | Price | Borrowed? | Comments |
|--------------------|------------------|---------|-------------------|-----------------------------|
| Halo 3 | Xbox 360 | \$59.99 | No | Love the online multiplayer |
| Spy Hunter | Nintendo Gameboy | \$12.99 | No | |
| FIFA 2006 | Nintendo Gameboy | \$12.99 | No | |
| Saint's Row | Xbox 360 | \$59.99 | No | Awesome game! |
| Nascar 2008 | Xbox 360 | \$49.99 | Scott (10/1/2008) | Hard to drive cars |
| Call of Duty 4 | Xbox 360 | \$59.99 | No | My favorite game! |
| NBA 2K7 | Xbox 360 | \$59.99 | Sean (9/15/2008) | |
| Super Mario Bros 3 | Nintendo Gameboy | \$19.99 | No | |
| Gears of War | Xbox 360 | \$59.99 | No | Would like to sell or trade |

Figure 2-6. The game spreadsheet

As you've distributed the spreadsheet to friends over a period of months, various copies have become outdated and it's causing quite a bit of confusion. Mitch wants to borrow *NBA 2K7* but his version of the spreadsheet indicates Sean has it, although Sean actually returned it weeks ago. Scott wants to buy *Gears of War* from you but you already sold it. And nobody seems to know you've been playing *Madden 2009* day and night for weeks now. Multiple versions of the spreadsheet are causing a giant headache, and so you've decided to create a website which all of your friends can simply refer to for updates. All you need to do is figure out how to make the spreadsheet available online.

Converting the Spreadsheet

To make your spreadsheet data available online, you'll need to first convert it to CSV, or *Comma-Separated Values* format. CSV isn't formally a standard format, although it's supported by many applications and languages as a generally agreed upon way to share data, PHP included. As Microsoft Excel remains far-and-away the most popular spreadsheet creation and analysis application, I'll show you how to convert an Excel spreadsheet to CSV format.

Saving an Excel spreadsheet in CSV format is very simple; just open up the spreadsheet in Excel and then navigate to File -> Save As. The Save As dialog will appear as shown in Figure 2-7. Navigate to a desired save destination, assign the file an easily-recognizable name, and from the Save As Type drop-down menu choose CSV (comma delimited). Finally, press the Save button to create the CSV file.

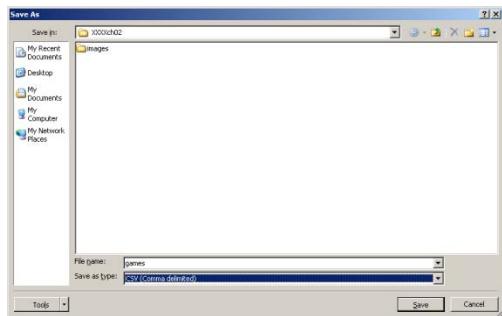


Figure 2-7. Saving the spreadsheet as a CSV file

If you open up this file in a text editor, you'll see that each spreadsheet field is separated by a comma. To prevent any confusion which might arise due to commas found in any of the fields, Excel delimit-

ed those fields with quotation marks. For example, a few lines found in the `games.csv` file are shown here:

```
Title,Platform,Price,Borrowed?,Comments
Halo 3,Xbox 360,$59.99,No,Love the online multiplayer
Spy Hunter,Nintendo Gameboy,$12.99,No,
FIFA 2006,Nintendo Gameboy,$12.00,No,
Saint's Row,Xbox 360,$59.99,No,Need to finish it
Nascar 2008,Xbox 360,$49.99,Scott (10/1/2008),Hard to drive cars
Call of Duty 4,Xbox 360,$59.99,No,My favorite game!
NBA 2K7,Xbox 360,$59.99,Sean (9/15/2008),
Super Mario Bros 3,Nintendo Gameboy,$19.99,No,
Gears of war,Xbox 360,$59.99,No,Would like to sell or trade
```

Later in this section I'll show you how to disregard these quotation marks when parsing the CSV file.

TIP. Chances are you're going to eventually include a comma in the text found in one of the columns, which will present a problem when you later write a script to parse the CSV file. The most effective solution is to change the comma delimiter Excel uses to something you'll never use in the spreadsheet, for instance a vertical bar |. While Excel doesn't offer any way to change this, you can still change the delimiter by navigating to Start -> Control Panel -> Regional and Language Options. From here the process varies depending upon your version of Windows. If you're using Windows XP, click the Regional Options tab, then Customize, and change the separator found in the List separator box. If you're using Windows Vista, click the Regional Options tab, and then click Customize this format, and change the separator found in the List separator box. For the purposes of this exercise I'll forego using commas in any of the fields so we can just use the default setting.

Reading the CSV File

Next we'll write a short PHP script capable of reading the CSV file's contents, and formatting and displaying the data to the browser. To begin, place the newly saved CSV file within your application's home directory so the script can easily access it.

NOTE. If you don't want visitors to be able to download your CSV file, you can instead place the file within PHP's include path, specified within PHP's `php.ini` configuration file by the `include_path` directive.

Next we'll create the script which will be used to parse the CSV file. Believe it or not, much of the task has already been implemented within a standard PHP function known as `file()`. The `file()` function will parse a file, dumping its contents into an array. In doing so, it treats each line as a new array element, with each line ending when it encounters the hidden newline (`\n`) character. For the first iteration of our script (`game_parse.php`), we'll dump `games.csv` into an array, and then iterate over each line of the array, displaying each element to the browser:

```
<?php
    // Dump the games.csv file into an array
    $games = file('games.csv');

    // Parse the array
    foreach($games as $game) {
        echo "{$game}<br />";
    }

?>
```

Executing `game_parse.php` produces the output shown in Figure 2-8:

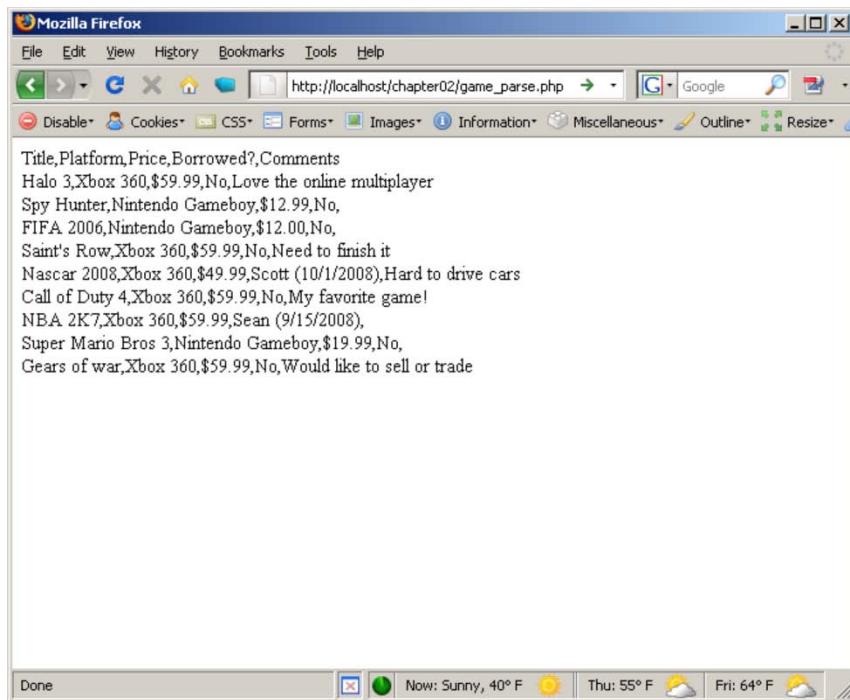


Figure 2-8. Parsing the games.csv file

Now we're getting somewhere! However we're looking to put together a rather professional presentation, not to mention one that provides a greater level of control over each array element. To accomplish this goal, let's revise the `game_parse.php` script to organize the data in an HTML table.

Displaying Data in a Table

The grid has long been one of the most popular approaches to presenting data in an easily readable fashion. For instance, calendars, flight schedules, and spreadsheets are just a few examples of the grids found all around you. Given the prevalence of this format, it might not come as a surprise that the grid was one of the first features available to the HTML markup language.

To create a well-organized table, you only need to use a few HTML tags. For instance, the following HTML snippet produces the table found in Figure 2-9.

```
<style type="text/css">

#games {
    border: 1px solid black;
}

#games th {
    text-align: left;
}

</style>

<table id="games">
    <tr>
        <th>Title</th>
        <th>Platform</th>
        <th>Price</th>
        <th>Borrowed?</th>
        <th>Comments</th>
    </tr>
    <tr>
        <td>Halo 3</td>
        <td>Xbox 360</td>
        <td>$59.99</td>
        <td>No</td>
        <td>Love the online multiplayer</td>
    </tr>
    <tr>
        <td>Spy Hunter</td>
        <td>Nintendo Gameboy</td>
        <td>$12.99</td>
        <td>No</td>
        <td></td>
    </tr>
</table>
```

| Title | Platform | Price | Borrowed? | Comments |
|------------|------------------|---------|-----------|-----------------------------|
| Halo 3 | Xbox 360 | \$59.99 | No | Love the online multiplayer |
| Spy Hunter | Nintendo Gameboy | \$12.99 | No | |

Figure 2-9. A sample HTML table

In the case of the games spreadsheet, we want the table to expand and shrink based on the number of rows in the spreadsheet, therefore the table must be built dynamically. In the next section I'm going to show you how to use a great third-party solution for doing so. First however I'd like to introduce you to a community-driven code sharing service known as PEAR, which we'll use to obtain the third-party

code for generating dynamic HTML tables.

Introducing PEAR

The open source community has a long history of creating convenient solutions for sharing code. This code typically offers well-designed solutions to common problems faced by a majority of programmers within that community. For instance, no matter what sort of PHP-driven web application you'd like to build, chances are you're going to have to deal with things such as table layouts, form processing, and validation of user input. You may also need to perform seemingly more esoteric tasks such as converting numbers to Roman numerals, validating the syntax of IP addresses, or creating user passwords. Wouldn't it be great if you could simply take advantage of readily available solutions for these sorts of problems rather than go through the trouble of creating your own?

Indeed, the PHP community has long had a community-driven solution in place for sharing code capable of accomplishing all of the aforementioned tasks and more. Known as PEAR, or the *PHP Extension and Application Repository*, there are at the time of this writing 460 such solutions freely available for your use! You can browse a list of available packages at <http://pear.php.net/>.

Unfortunately, many PHP books tend to shy away from showing readers how to find and use this code until much later into the introduction. Not so here! Because this book is all about showing you how to use PHP in the most efficient and straightforward manner possible, I'd like to buck this trend and instead introduce you to PHP's particular solution as early as practical in the book, which is now.

Using PEAR

If you followed the installation instructions found in Chapter 1, then PEAR is already installed meaning you can begin using it immediately. You can verify it's installed by opening up a shell (command prompt in Windows) and executing the following command:

```
%>pear
```

This will produce a listing of available commands and other usage information. Feel free to experiment with this command to get a feel for what PEAR can do. For instance, execute the following command to learn what PEAR packages are already installed on your machine (several are installed by default):

```
%>pear list
```

I'm not going to introduce each of the available PEAR commands, as the listing produced earlier should be fairly self explanatory. Just keep in mind you'll use this command to install, update, and remove packages. In fact you'll put this command into practice by installing the `HTML_Table` package in the next section.

PEAR packages are all accessed through the object-oriented programming (OOP) paradigm. Most PHP books tend to give OOP the white glove treatment, because of the notion that OOP is really hard for newcomers to programming to understand. I dispute this convention, and in fact think you'll find it quite easy to understand! OOP is centered around the idea of a *class*, which you can think of as a

cookie-cutter or a blueprint for a particular entity we might find in the real world, such as an automobile, person, or HTML table. This class defines the characteristics (known as *attributes*) and behaviors (known as *methods*) of the entity. For example, an HTML table class might have an attribute which identifies the table's CSS ID, and a method that adds a header to the table.

But remember the class is a blueprint; to use it you need to create an instance of that blueprint, known as an *object*. This makes it possible for you to use multiple instances of the class on the same web-page for instance, which would be necessary if you wanted to add multiple tables to a page in the case of an HTML table class.

There's much more to OOP than what I've introduced in the previous two paragraphs, however for the moment it's irrelevant. Just remember the four key terms: class, object, attribute, and method, and what follows in the remainder of this chapter will be quite clear.

VIDEO. Introducing Object-Oriented Programming

The object-oriented programming paradigm views applications much in the same way humans view the world: in terms of objects and their interactions. In mimicking our natural world-view, developers can better conceptualize, design, and maintain applications. This video introduces object-oriented programming, and guides you through PHP's object-oriented features. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Introducing HTML_Table

`HTML_Table` is a PEAR solution for easily creating dynamic HTML tables. You can use it to manage all characteristics of the table, such as setting cell contents, retrieving the total number of rows created, and assigning CSS identifiers to the table, headers, and cells. `HTML_Table` comes with more than 35 methods, however six are of particular importance because you'll use them in almost every table you generate using this package:

- `HTML_Table()`: This method creates a new table, and must be called before you can use any other `HTML_Table` methods.
- `setHeaderContents()`: This method assigns data to the `<th>` elements which are used to title each table column.
- `setRowAttributes()`: This method assigns attributes to the `<td>` elements found in a table row.
- `setCellContents()`: This method assigns data to a table cell (`<td>`).
- `altRowAttributes()`: This method assigns attributes to alternate rows in the table, typically for reason of enhancing readability.
- `toHtml()`: This method outputs the table structure in HTML format.

You can learn more about `HTML_Table`, and can review a breakdown of all available methods at http://pear.php.net/package/HTML_Table.

Installing HTML_Table

To install `HTML_Table`, fire up PEAR in the same fashion as you did earlier, and execute the following command:

```
%>pear install -o HTML_Table
```

Some installation-related output will scroll by, followed by a message confirming successful installation. That's it! You're ready to begin using `HTML_Table`.

Using HTML_Table

As mentioned, we're going to use `HTML_Table` to format the game collection in a much more visually appealing fashion than what was achieved with the earlier approach. The script we'll cover in this chapter will produce the table shown in Figure 2-10.

| Title | Platform | Price | Borrowed? | Comments |
|--------------------|------------------|---------|-------------------|-----------------------------|
| Halo 3 | Xbox 360 | \$59.99 | No | Love the online multiplayer |
| Spy Hunter | Nintendo Gameboy | \$12.99 | No | |
| FIFA 2006 | Nintendo Gameboy | \$12.99 | No | |
| Saint's Row | Xbox 360 | \$59.99 | No | Awesome game! |
| Nascar 2008 | Xbox 360 | \$49.99 | Scott (10/1/2008) | Hard to drive cars |
| Call of Duty 4 | Xbox 360 | \$59.99 | No | My favorite game! |
| NBA 2K7 | Xbox 360 | \$59.99 | Sean (9/15/2008) | |
| Super Mario Bros 3 | Nintendo Gameboy | \$19.99 | No | |
| Gears of War | Xbox 360 | \$59.99 | No | Would like to sell or trade |

Figure 2-10. Formatting the game collection using a table

The next script (`games_table.php`) presents the code. Following the listing is a breakdown of the script.

```
01 <?php
02
03     // Make HTML_Table available to the script
04     require 'HTML/Table.php';
05
06     // Dump games.csv into an array
07     $games = file('games.csv');
08
09     // Set Table CSS class
10     $attrs = array('class' => 'game');
11
12     // Create a new instance of the HTML_Table class
13     $table = new HTML_Table($attrs);
14
15     // Output the header
```

```
16 $header = explode(',', $games[0]);
17 for($header_count = 0; $header_count < count($header); $header_count++) {
18     $table->setHeaderContents(0, $header_count, $header[$header_count]);
19 }
20
21 // Set Header CSS class
22 $table->setRowAttributes(0, array('class' => 'header'));
23
24 // Parse the array
25 for($row_count = 1; $row_count < count($games); $row_count++) {
26
27     // Convert each line into an array
28     $game = explode(',', $games[$row_count]);
29
30     // Set Row CSS class
31     $table->setRowAttributes($row_count, array('class' => 'row'));
32
33     // Output the next row
34     for($col_count = 0; $col_count < count($game); $col_count++) {
35         $table->setCellContents($row_count, $col_count, $game[$col_count]);
36     }
37
38 }
39
40 // Set attributes for alternating rows
41 $altRow = array('class' => 'altrow');
42 $table->altRowAttributes(1, null, $altRow);
43
44 // Output the HTML table to the browser
45 echo $table->toHtml();
46
47 ?>
```

Let's review some of this script's key lines:

- Line 04 makes the `HTML_Table` package available to the script by "including it" into the document. We'll talk more about the details of the `require()` command in the next section, including how the `require()` command was able to locate the package without having to define the complete path.
- Line 07 dumps the `games.csv` file to an array named `$games`.
- Line 10 creates an array of attributes that will be associated with the `<table>` tag. Typically you'll use this to assign a CSS class tag so CSS style can control attributes such as the table's width and color.
- Line 13 creates a new instance of the `HTML_Table` class, and simultaneously assigns the aforementioned CSS class tag to it.

- Line 16 uses the PHP function `explode()` to parse just the first line of the array. This results in retrieving the spreadsheet header names which we'll use for the table column headers.
- Lines 17-19 iterate through the `$header` array, placing each header title into the table header using the `setHeaderContents()` method. Note the first parameter of this method is set to 0 because the table row offset for the very first line is zero.
- Line 22 sets the CSS class for the header.
- Line 25 initiates the iteration of the remaining lines in the `$games` array. In this `for` loop we'll repeatedly peel off the next row in the array (line 25), parse it into an array called `$game` (line 28), set the CSS class for each row (line 31), and then output each element into a `<td></td>` tag.
- Lines 41-42 set another CSS ID for alternating rows in the table, beginning with row 1.
- Line 45 outputs the table to the browser.

Step #3. Managing Your Site Design Using Templates

Chances are you have ambitious plans when it comes to the game directory, meaning the website could soon reach 10, 20, and perhaps even hundreds of pages. But how can you easily manage the design of each page, presuming each will be based around an identical design theme? For instance, even changing the copyright year found at the bottom of each page could become quite a chore if the website consisted of 500 pages.

To remedy this problem, you can create page headers and footers, and then include them within your scripts. By isolating these templates to a single file, changing header or footer content is as simple as opening and editing a single document. First up let's review the game directory's header (`header.php`):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>My Game Collection</title>
    <link rel="stylesheet" type="text/css" href="/chapter02/css/styles.css" />
  </head>
  <body>
    <div id="logo">
      
    </div>
    <br />

    <div id="container">
```

Next you'll find the game directory footer (`footer.php`):

```

</div>

<div id="footer">
    If you want to borrow a game, contact me!
</div>
</body>
</html>

```

By inserting the header.php and footer.php scripts into their appropriate locations with the script displaying the table, we'll be able to achieve a result similar to that shown in Figure 2-11.

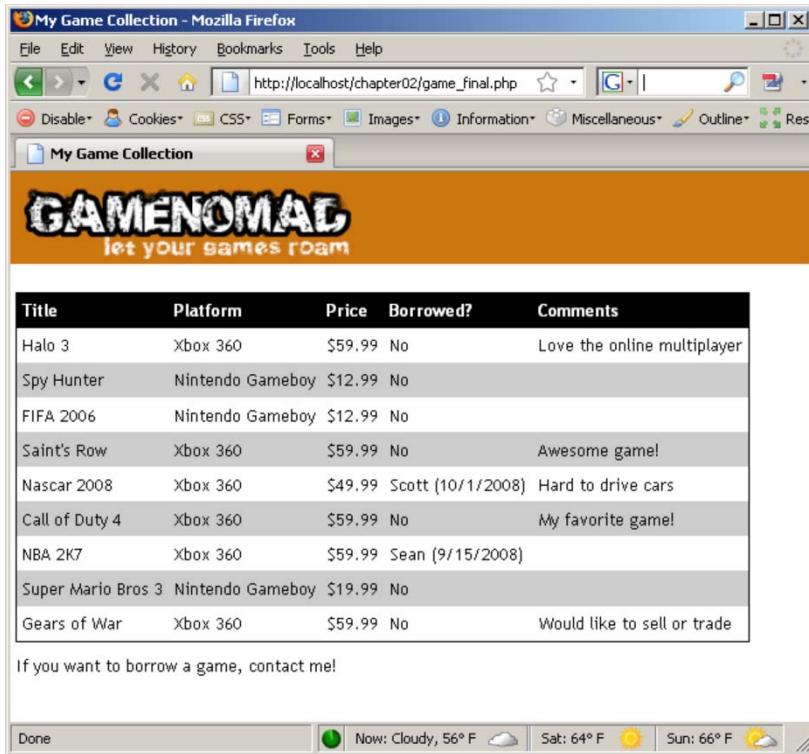


Figure 2-11. The game directory page assembled (game_final.php)

To integrate the header and footer into your script, you'll use the `require()` statement. This statement will retrieve the contents of the specified file, and integrate it into the calling page. If any PHP logic is found in the file, that logic will be evaluated. The following script (`game_final.php`) demonstrates how Figure 2-11 was created.

```

<?php
    // Insert page header
    require('templates/header.php')

    // The contents of games_table.php would be inserted here,
    // but for reasons of space are clipped from the book.

```

```
// Insert page footer  
require('templates/footer.php');  
?>
```

The `require()` function doesn't require a full path name provided you modify the `includes_path` directive found in your `php.ini` file. For instance, for purposes of this example I set my `includes_path` directive like so:

```
includes_path = ".;c:\apache\htdocs\easyphpwebsites\code\chapter02"
```

On Linux, this path might look like this:

```
includes_path = "./usr/local/apache/htdocs/easyphpwebsites\code\chapter02"
```

From here, the `require()` statement will look for a `templates` directory as specified in the provided path in the script, and from there retrieve the `header.php` and `footer.php` files. If you don't have the `php.ini` file at your disposal, then you're certainly free to specify the complete path.

As you'll learn in later chapters, far more powerful ways exist for managing large websites, in fact no such "pages" will exist; rather each will be dynamically created according to user request. However, if you're interested in building a simple website consisting of just a few pages, using the technique described in this section will be ideal.

Conclusion

Congratulations! In this brief chapter you've progressed from being a PHP neophyte to having successfully built your first dynamic Web application! In the next chapter you'll give your friends an easy way to contact you through a Web form.

CHAPTER 3

Interacting With Your Users

By now the spreadsheet is online, and friends are really digging this new and convenient way to keep track of your game collection. Encouraged by their interest, you've started thinking about ways to expand the website. For starters, you think it would be cool if they could use the website to ask your opinion about certain games, or even request to borrow a few games for the coming weekend.

While the easiest solution might involve simply publishing an e-mail address to the site, you've heard doing so is a surefire way to get deluged by unwanted e-mail from spammers. The most effective alternative is a Web form, which an inquiring visitor can fill out, providing information such as his name, e-mail address, and a brief message. In this chapter we'll cover form fundamentals, teaching you not only how to retrieve data submitted through a form, but also how to validate the data to ensure it meets your expectations. You'll also learn how to send this form data directly to your inbox by sending an e-mail directly from a PHP script.

Chapter Steps

The goals of this chapter are accomplished in five steps:

- **Step #1. Creating a Contact Form:** Creating an HTML form is easy enough, but what happens to the data once it's submitted? In the first step of this chapter I'll show you how to pass form information to a PHP script for further processing by creating a contact form which visitors can use to get in touch with you.
- **Step #2. Validating User Input:** Because most web sites are open to visitors from around the world, malicious individuals included, it is crucial to thoroughly examine and validate all user input. In this step I'll show you how to validate input by creating your own data validation class.
- **Step #3: Repopulating Form Data:** In cases when users have been informed of submission errors (such as an invalid e-mail address), you want to let them know about the error while not inconveniencing them by forcing them to complete the form anew. In this step I'll show you how to repopulate forms with user data to save them the hassle.
- **Step #4. Sending Form Data via E-mail:** By now you know how to efficiently capture and process form data, but what's next? For relatively low traffic contact forms, the easiest way to view the data is by sending it via e-mail to some designated address. In this step I'll show you how to transmit forms data to your inbox using PHP's `mail()` function, as well as through a more popular solution known as the Mail PEAR package.
- **Step #5: More on Securing User Input:** While Step #2 showed you how to validate form data to ensure it conforms to certain constraints, validation is really only part of the bigger picture when it comes to securing data arriving from an untrusted source (in the case of validation, the user). In this concluding step I'll discuss some crucial steps you should, no must, take when processing data arriving from untrusted sources.

Step #1. Creating a Contact Form

As the website increases in popularity, your stature as a gaming aficionado continues to grow. Not surprisingly, visitors have started emailing you asking your opinion about new and soon-to-be-released games, as well as lobbying you to add their favorite game to your vaunted list. However, publishing your e-mail address on the website has resulted in it being netted by spammers' automated programs, and your inbox is being deluged with Viagra offers and notifications of lottery winnings.

The most effective alternative solution is a web-based contact form. By now you've probably used these sorts of forms dozens of times to contact companies and owners of various websites. In this section I'll show you how to build such a form and process data sent through it using a PHP script. We'll create the form shown in Figure 3-1, subsequently using a PHP script to just display the submitted contents back to the Web page. In Step #3 you'll learn how to send these contents to your e-mail inbox.

Your name:

Your e-mail:

Your phone number:
 () -

Preferred contact method:

What is your favorite video game:

Subscribe to the gaming newsletter? Yes

Figure 3-1. Invite your visitors to use a contact form

The Contact Form HTML

We'll begin by creating the contact form HTML. I'll presume you're at least somewhat familiar with HTML syntax at this point, and so won't break down the code, although following Listing 3-1 I'll discuss a few key form characteristics you should keep in mind when creating PHP-driven forms.

Listing 3-1. The contact form HTML (listing3-1.php)

```
<form id="form" method="post" action="listing3-1.php">

    <p>
        Your name: <br />
        <input name="name" id="name" type="text" size="35" value="" />
    </p>
```

```
<p>
    Your e-mail: <br />
    <input name="email" id="email" type="text" size="35" value="" />
</p>

<p>
    Your phone number: <br />
    (<input name="phone1" id="phone1" type="text" size="3" value="" />)
    <input name="phone2" id="phone2" type="text" size="3" value="" />-
    <input name="phone3" id="phone3" type="text" size="4" value="" />
</p>

<p>
    Preferred contact method:<br />
    <select name="contact_method">
        <option value="email">E-mail</option>
        <option value="telephone">Telephone</option>
    </select>
</p>

<p>
    What is your favorite video game:<br />
    <textarea name="message" cols="35" rows="5"></textarea>
</p>

<p>
    Subscribe to the gaming newsletter?
    <input name="newsletter" id="newsletter" type="checkbox" value="yes" />Yes
</p>

<p>
    <input name="submit" id="submit" type="submit" value="Send!" />
</p>

</form>
```

Even if you're new to PHP, I'd imagine you're pretty familiar with HTML forms, however the first line might still be a mystery if you're new to forms processing. To recap, the line is:

```
<form id="form" method="post" action="listing3-1.php">
```

This line is particularly important because it defines two key characteristics of this form. First, it specifies that the form data should be submitted using what's known as the POST method. I'll spare you a deep technical explanation regarding form methods, however it's important you understand that the POST method should always be used for forms-related requests which *add or change* the world's "state", so to speak. For instance, submitting this form will introduce new data into the world, meaning the proper method to use is POST. On the contrary, forms intended to retrieve information, for instance from a search engine, should use the GET method. Forms submitted using the GET method

will result in the data being passed by way of the URL. For instance, if you head on over to Amazon.com and search for a book, you'll see the search keywords passed along on the URL.

The distinction is important because forms are often used to perform important tasks such as processing a credit card. Browser developers presume such forms will adhere to the specifications and be submitted using the POST method, thereby warning the user if he attempts to reload the page, potentially causing the critical action to be performed anew (in this case, charging the credit card a second time). If GET was mistakenly used for this purpose, the browser would logically not warn the user, allowing the page to be reloaded and the credit card potentially charged again (I say potentially because the developer should logically have built safeguards into the application to account for such accidents). Given the important distinction between these two methods, keep the following rules-of-thumb in mind when building web forms:

- Use GET when the form results in an action being taken that no matter how many times it's submitted anew, will not result in a state-changing event. For instance, searching a database repeatedly will not affect the database's contents, making a search form a prime candidate for the GET method.
- Use POST when the form submission results in a state-changing event, such as a comment being posted to a blog, a credit card being charged, or a new user being registered.

This opening line also defines another important form characteristic: the URL to which the form data will be sent. Of course, there's nothing to prevent you from submitting the data right back to the very same form, a strategy that is particularly convenient when working with a fairly simple form such as this. I'll show you how to do this next.

VIDEO. Creating Your First Web Form

Web forms are used in every conceivable type of website, acting as the channel for contacting the sales team, posting a comment, or listing an item on a site such as craigslist. This video shows you how to use PHP to retrieve data submitted through a form, and demonstrates why it's so important to properly validate user-supplied data.

Sending Form Data to a PHP Script

If you call the contact form from the browser, fill it out, and press the submit button, nothing of consequence happens. This is because although we've specified the form destination to be `listing3-1.php`, the PHP code used to process the data hasn't yet been created. In this section you'll create this script, and learn how to parse the data submitted by the user via the contact form.

Any PHP script defined as the form's submission destination can access the form data using an associative array called `$_POST`. The array values can be accessed by referencing the appropriate key, with the key being defined by the form's input tag names. For instance, to access the submitting user's e-mail address you would reference `$_POST` like this:

```
$_POST['email'];
```

Accessing form fields such as checkboxes and drop-down lists operate in identical fashion. For in-

stance, to access the user's preferred contact method you would reference `$_POST` like this:

```
$_POST['contact_method']
```

This would logically contain one of two values: `email` or `telephone`.

Because `$_POST` is an array, you can easily output all of the elements using `print_r()` (The `print_r()` function was first introduced in Chapter 2). In fact, in Listing 3-2 we'll create the skeleton for what will ultimately become the `index.php` script by using `print_r()` to simply dump the submitted form data to the browser.

Listing 3-2. Outputting submitted form data to the browser

```
01 <?php
02
03     if ($_POST)
04         print_r($_POST);
05
06 ?>
```

Obviously there isn't a practical application to the code found in Listing 3-2, however it does prove the important point that this data is indeed accessible by the PHP script. Completing and submitting the form will produce output similar to this (formatted for readability):

```
Array (
    [name] => Jason Gilmore
    [email] => jason@wjgilmore.com
    [phone1] => 614
    [phone2] => 555
    [phone3] => 1212
    [contact_method] => email
    [message] => We've got to get together and play Call of Duty!
    [newsletter] => yes
    [submit] => Send!
)
```

Take special notice of the text attached to the key `message`, because you'll see a single quote included in the value. For versions prior to PHP 5.2, PHP will automatically *escape* single quotes (along with double quotes, backslashes, and NULL characters), meaning a backslash is placed before each quote. This was done in an effort to prevent users from submitting potentially dangerous input. In Step #2 I'll talk more about why these sorts of special characters are dangerous, explain PHP's reasoning for escaping them, and show you what you should do as an alternative if running PHP version 5.2 or newer (which I highly recommend).

Of course, outputting the array in this manner is impractical, useful only for quickly viewing form contents. Let's revise the code (Listing 3-3) to selectively output each form item.

Listing 3-3. Outputting form data in a somewhat more coherent manner

```
<?php
if ($_POST) {
    echo "Name: {$_POST['name']}<br />";
    echo "E-mail: {$_POST['name']}<br />";
    echo "Tel: ({$_POST['phone1']}) {$_POST['phone2']}-{$_POST['phone3']}<br />";
    echo "Preferred contact method: {$_POST['contact_method']}<br />";
    echo "Message: {$_POST['message']}<br />";
    echo "Subscribe to newsletter?: {$_POST['newsletter']}";
}
?>
```

Submitting the revised form script will produce the following output:

```
Name: Jason Gilmore
E-mail: jason@wjgilmore.com
Tel: (614) 555-1212
Preferred contact method: email
Message: We've got to get together and play Call of Duty!
Newsletter subscription decision: yes
```

Dealing with Empty Values

If when experimenting with Listing 3-3 you happened to impatiently complete only part of the form before submitting it, chances are your output was accompanied by an error message looking like this:

```
Notice: Undefined index: newsletter in C:\apache\htdocs\chapter03\listing3-3.php
on line 8
```

This is because the identifiers for certain form field types will not be passed at all in the case they're left blank (namely radio buttons and check boxes), resulting in the lack of a corresponding variable within the `$_POST` array. You can determine whether all form variables exist by first checking them with the `isset()` function, like so:

```
if (isset($_POST['newsletter'])) {
    echo "Newsletter subscription decision: {$_POST['newsletter']}<br />";
}
```

But data validation requires more than checking for empty values. In fact it's so important that the entire second step in this chapter is devoted to the topic.

Step #2. Validating User Input

If there were one mantra developers should utter before firing up their IDEs each day, it's "never trust user input". While most users will interact with your web site with no malicious intent, rest assured it will regularly be probed, prodded, and subject to outright attack by unsavory individuals seeking profit, glory, or some twisted version of entertainment. But by rigorously validating user input at ev-

ery opportunity, you can greatly minimize the possibility your website and data will be compromised.

Of course, thorough data validation also has the positive side-effect of ensuring your well-intentioned users have also provided both adequate and proper input. Otherwise, it would probably come as a surprise to both you and your users if your e-commerce website accepted orders without requiring a credit card number, or by accepting one which proves to be completely invalid!

Because data validation will play such an important role in all of your applications, it makes sense to create a set of functions can be repeatedly used within the validation process. Creating this set of functions, which we'll organize in a file commonly referred to as a *library*, is not only a valuable exercise in terms of showing you how to create and reuse a library, but also in terms of demonstrating PHP's power in terms of parsing data.

Creating Your First Library

Creating a PHP library doesn't require any special programming or technical considerations; it's just a term used for a PHP script which contains a set of functions typically created to perform a set of related tasks. For instance, a validation library might contain four functions: one for determining whether a value is empty, another for validating an e-mail address, yet another for determining the validity of a date, and a final function for validating a phone number. In this section we'll create these four functions, and organize them within a library named `gamenomad_validation.php`.

Checking for Empty Values

The first function we'll create is the easiest. The `isBlank()` function accepts a value, determining whether the value is blank. If so, it returns `TRUE`, otherwise it returns `FALSE`. The code is presented next, followed by a short summary.

```
01 /**
02  * Determines whether a value is blank
03  * @param string $input value to validate
04  * @return boolean
05 */
06
07 function isBlank($input)
08 {
09     if ($input == '') {
10         return TRUE;
11     } else {
12         return FALSE;
13     }
14 }
```

A summary of the `isBlank()` function follows:

- This is a fairly straightforward function, accepting one input parameter (`$input`), and using PHP's equality operator (`==`) to determine whether `$input` is blank. If so, it returns `TRUE` to

- Lines 01-05 of this example also demonstrate another valuable concept: documentation. Whether working solo or in teams, it is crucial you take the time to thoroughly document your code for later reference. This particular example demonstrates a rigorous form of documentation using phpDocumentor (<http://www.phpdoc.org/>) syntax. While you're not required to document your PHP scripts, I strongly suggest looking into this solution as early as possible.

Validating Phone Numbers

Next let's create a slightly more complex function for validating U.S. phone numbers. The code is presented next, followed by a breakdown of the relevant lines:

```
01 /**
02 * Validates a U.S. phone number for proper format
03 * @param integer $areaCode the phone number's area code
04 * @param integer $prefix the phone number's prefix
05 * @param integer $lineNumber the phone number's line number
06 * @return boolean
07 */
08
09 function isValidPhone($areaCode, $prefix, $lineNumber)
10 {
11
12     $validAreaCode = (strlen((int)$areaCode) == 3);
13     $validPrefix = (strlen((int)$prefix) == 3);
14     $validLineNumber = (strlen((int)$lineNumber) == 4);
15
16     if ($validAreaCode && $validPrefix && $validLineNumber) {
17         return TRUE;
18     } else {
19         return FALSE;
20     }
21
22 }
```

Let's review the code:

- Lines 01-07 continue the best practice of documenting the code.
- Line 09 accepts three parameters, namely `$areaCode`, `$prefix`, and `$lineNumber`. These are the technical terms assigned to the three parts of a U.S. phone number. For instance, given the number (614) 555-1212, 614 forms the *area code*, 555 the *prefix*, and 1212 the *line number*.
- Lines 12-14 perform two important verifications. First, by prefixing a variable with `(int)`, we're using a technique known as *typecasting*. Typecasting will attempt to convert, or cast, that value as the specified type (in this case, `integer`), and in the process could render invalid any values not fitting the specifications defining an `integer`. For instance, typecasting 614

would have no effect because the value is indeed an integer. However, typecasting `n14` would produce the value `14`, effectively removing the `n` from the front of the value. But typecasting won't merely remove invalid characters as the previous example implies. For instance, typecasting `6n4` as an integer produces solely `6`.

- Following each typecast, the `strlen()` function found in lines 12-14 determine the length of what remains following the typecast. The area code, prefix, and line number should consist of three, three, and four digits respectively, and if so, the fact we're using the equality operator (`==`) will result in the assignment of an appropriate Boolean value to the `$validAreaCode`, `$validPrefix`, and `$validLineNumber` variables, respectively.
- Line 16 determines whether all three phone number components satisfied the tests in lines 12-14 by chaining their resulting Boolean values together to form a logical operation. In this case, because the `&&` operator is being used we're requiring all three chained values to be assigned `TRUE` in order for the operation to process as valid. This follows the general rules as defined by Boolean math, which you almost certainly learned in middle school. If the operation is valid, `TRUE` is returned, otherwise `FALSE` is returned.

Validating Dates

Of course, not all custom functions need to consist solely of custom code. After all, PHP has thousands of functions and hundreds of libraries to draw upon, so why not rely upon these well-tested functions to perform some of the more complex tasks? A great example of just such a complex task is date validation. For instance, supposing we're using a date format of MM/DD/YYYY (example: 07/04/1776) you can't just presume a date is valid if each component is a valid integer. For instance, while 04/12/78 is valid, 04/31/78 is not. Therefore a more sophisticated mechanism is required for not only ensuring each component is a valid integer, but also that it meets the constraints as defined by the Gregorian calendar (the world's most widely used calendaring system). While we could write a rather complex piece of code to perform this task, a PHP function named `checkdate()` is already available. We'll use this function within `isValidDate()` to validate any dates submitted through a form:

```

01 /**
02  * Determines whether a value is a valid date as defined
03  * by PHP's checkdate() function
04  * @param string $date date to validate
05  * @return boolean
06 */
07 function isValidDate($date)
08 {
09     list($month, $day, $year) = explode("/", $input);
10     return checkdate($month, $day, $year);
11 }
```

Let's review this code:

- Line 09 uses the `explode()` function to divide the `$input` parameter into three component (`$year`, `$month`, `$day`). The `explode()` function uses the specified separator (in this case a forward slash) to determine the cutoff point for each component. Because `explode()`'s de-

fault behavior is to return these parts in an array, we can use a nifty trick to capture the three desired values by immediately parsing the returned array using the `list()` function.

- Line 10 passes the date components into the `checkdate()` function, which determines whether they satisfy the constraints as defined by the Gregorian calendar.

Changing the Separator

The `explode()` function is useful because we can use it to easily parse out the month, day and year components from a date. But what if the separator is a hyphen? Or something more esoteric, like an asterisk? To ensure maximum flexibility, you could use an optional function input parameter to assign a default separator, while simultaneously giving you the opportunity to override the default separator if you know another will be used. Let's revise `isValidDate()` to include this default input parameter:

```
01 /**
02  * Determines whether a value is a valid date as defined
03  * by PHP's checkdate() function
04  * @param string $input date to validate
05  * @param string $separator date component separator
06  * @return boolean
07 */
08 function isValidDate($input, $separator='/')
09 {
10     list($month, $day, $year) = explode($separator, $input);
11     return checkdate($month, $day, $year);
12 }
```

This revised `isValidDate()` function differs little from its predecessor, save for two important points:

- An optional second input parameter named `$separator` has been defined. If this parameter is not passed when invoking the function, it will be presumed you wanted to use a forward slash as the separator.
- Instead of exclusively using a forward slash as the separator, `explode()` now accepts the `$separator` function as its first argument.

With this optional parameter defined, the `isValidDate()` function can now be called using two formats. To use the default forward slash separator, you would call it like so:

```
if (isValidDate('07/04/2008')) {
    echo "This date is valid!";
}
```

Alternatively, if you preferred an alternative separator is used, you can identify the separator like so:

```
if (isValidDate('07-04-2008', '-')) {
    echo 'This date is valid!';
}
```

Validating E-mail Addresses

For the fourth and final validation example, we'll create the most complex validator of the bunch. Validating an e-mail address is a fairly complex task, because of the complex syntactical rules which describe a valid e-mail. For instance, you probably recognize `games@example.com`, `2008gamer@example.net` and `i_love_games@example.org` as all valid e-mail addresses, but did you know `%+gamer%+@example.com`, `g-----@games.example.fm`, `you_like_"gaming"@example.info` and even `jason!loves@games@server2.example.museum` are equally valid? Other valid domains include `.museum`, `.info`, `.ca`, `.tv`, `.fm`, and `.biz`, among many others. With seemingly countless variations, how could you possibly account for all of them?

The answer lies in a powerful programmatic idiom known as the *regular expression*. You can think of a regular expression as a formalized description which can then be used to determine textual strings meet a well-defined set of characteristics. PHP supports two kinds of regular expression syntax, one called Perl-compatible and the other called POSIX Extended. We'll concentrate on the former, using it exclusively here and when appropriate throughout the book.

Introducing PHP's Perl-Compatible Regular Expression (PCRE) Syntax

Regular expression syntax is notoriously confusing to read, even among seasoned programming veterans, so if even you're new to the topic hopefully its reputation hasn't preceded reading this section. Even if it has, I'm convinced the gentle introduction to the topic in this section will leave your fears subsided. Let's begin with one of the simplest possible examples. Suppose you modified the contact form to also collect the user's zip code. U.S. zip codes come in two formats, the typical format consisting of five digits, and a second, more precise format consisting of five digits followed by a hyphen, followed by four more digits. We'll begin by validating the basic format, which makes sure exactly five digits have been provided:

```
01 if (preg_match('/^([0-9]{5})$/','43210')) {  
02     echo 'Valid zip code!';  
03 }
```

This simple example demonstrates several new concepts:

- The `preg_match()` function is used when you're trying to determine whether a string matches a regular expression, returning `TRUE` if so and `FALSE` otherwise.
- The `preg_match()`'s first input parameter is the regular expression. All regular expressions are surrounded by a pair of delimiters, which are typically forward slashes, although there's nothing to prevent you from using asterisks for instance. The second parameter is the string you're attempting to validate.
- The regular expression is the part of this example which is likely completely new to you. It's simple to dissect though. The carat (^) tells the regular expression that what's to follow must be found at the very beginning of the string. Therefore, while `43210` would validate, `Z43210` would not.
- The `[0-9]` is what's known as a *character class*, and it defines a range of acceptable char-

acters, in this case any integer between 0-9. Your options are practically unlimited when it comes to character classes. For instance, [a-e], [0-5], and [d-gD-G] are all acceptable classes.

- The character class is followed by a repetition operator which defines the number of times the characters in this class are allowed to appear, in this case five. Like character classes, repetition operators are also rather flexible. For instance, you can specify a minimum and maximum range {2,5}, or a minimum allowable number {2,}.
- Finally, the dollar sign (\$) tells the regular expression that what precedes it must appear at the very end of the string. This means that while 43210 would validate, 43210Z would not.

TIP. If you're solely interested in searching for a specific string, use PHP's `strpos()` function instead. It provides much faster performance than anything a regular expression could offer.

This example works fine for the five digit zip code, but what if you wanted to give the user the option of also including the additional four digit sequence? Because the goal is to allow the user to optionally give this additional information, we don't want to be too terribly overbearing in terms of the format, other than ensuring exactly five or exactly nine digits are provided. Therefore the revised regular expression should accept formats such as: 43210, 43210-1196, and 432101196. To make this happen, we need to figure out how to account for an optional hyphen, as well as an optional extra four digits.

You can declare a character or sequence to be optional using the asterisk (*) *metacharacter*. Metacharacters are characters considered to be of special significance within the expression. For instance, you've already learned about two metacharacters, namely the ^ and \$. Using the asterisk, we can specify that both the hyphen and the ensuing four digit sequence be optional (strictly speaking, the asterisk metacharacter will match zero or more of the previous character/sequence, thereby making it optional. You can use the plus sign metacharacter (+) to match one or more of the previous character/sequence, thereby making at least one instance required):

```
01 if (preg_match('/^([0-9]{5})(-)*([0-9]{4})*$/', '43210') {
02     echo 'Valid zip code!';
03 }
```

Believe it or not, you've learned everything you'll need to validate an e-mail address! Next, we'll create the regular expression, along the way introducing just a few more fundamentals which will complete this whirlwind introduction to regular expressions.

Validating an e-mail Address

As was discussed earlier in this section, validating an e-mail address is a fairly complex procedure. However, like most problems you can eventually work your way to the correct syntax by breaking the regular expression into distinct parts. Below you'll find the regular expression, followed by an explanation of each line:

```
01 if (preg_match(
02     '/^([_a-z0-9-]+)
03     (\.[_a-z0-9-]+)*
```

```

04      @([a-z0-9-]+)
05      (\.[a-z0-9-]+)*
06      (\.[a-z]{2,6})$/,
07      'jason@example.com')) {
08  echo 'Valid e-mail address!';
09 }
```

This is quite a mouthful. Let's break each line down:

- Line 02 and 03 define the part of the e-mail address commonly referred to as the username. The first sequence (^[_a-z0-9-+]+)), located on line 02, states that the address must begin with one or more underscores, lowercase letters, digits, hyphens, or plus signs. Technically speaking, the specification formally defining e-mail address syntax specifies this part can consist of a few other characters, such as the !, #, and \$, however most, if not all e-mail service providers disallow such characters, and so I don't see any reason for excluding them from the validation. Located on line 03, the second sequence ((\.[_a-z0-9-]+)*) is in place really to allow for a dot to be placed anywhere within the username except for the beginning of the username.
- Line 04 states that following the username, the at-symbol (@) must appear. This is followed by a sequence of one or more characters, with allowable characters including lowercase a-z, 0-9, and a hyphen.
- Line 05 specifies that all of the characters found in Line 04 are allowed, in addition to the period.
- Line 06 defines allowable top-level domain (TLD) addresses. A range of two to six characters are allowed, meaning TLDs such as .us, .com, .info, and .museum will all be accepted. The dollar-sign directly following this sequence means the TLD must appear last.

Congratulations, you've worked your way through a fairly complex regular expression, in the process creating the most significant piece of the `isValidEmail()` function. Let's finish the task by integrating it into the function:

```

01 /**
02  * Determines whether an e-mail address is valid
03  * @param string $input email to validate
04  * @return boolean
05 */
06 function isValidEmail($email)
07 {
08     if (preg_match(
09         '/^([_a-z0-9-]+)(\+)*(\.[_a-z0-9-]+)*
10         @([a-z0-9-]+)
11         (\.[a-z0-9-]+)*
12         (\.[a-z]{2,6})$',
13         $email)) {
14         return TRUE;
15     } else {
```

```
16     return FALSE;
17 }
18 }
```

Using the Validation Library

Using the library is simple; just include the `gamenomad_validator.php` file and call the functions as appropriate. Let's revise the relevant part of `contact.php` to validate each piece of user input:

```
01 if ($_POST) {
02
03     // Create array for storing error messages
04     $messages = array();
05
06     // Make sure a name was provided
07     if (isBlank($_POST['name'])) {
08         $messages[] = 'Please provide your name.';
09     }
10
11     // Validate the e-mail address
12     if (!isValidEmail($_POST['email'])) {
13         $messages[] = 'Please provide a valid e-mail address.';
14     }
15
16     // Validate the phone number
17     if (!isValidPhone($_POST['phone1'],$_POST['phone2'],$_POST['phone3'])) {
18         $messages[] = 'Please provide a valid phone number.';
19     }
20
21     // Make sure a contact method was provided
22     if (isBlank($_POST['contact_method'])) {
23         $messages[] = 'Please tell me your preferred contact method.';
24     }
25
26     // If errors exist, enumerate them. Otherwise, thank the user for
27     // getting in touch
28     if (count($messages) > 0) {
29         echo '<ul>';
30         foreach($messages AS $message) {
31             echo '<li>' . $message . '</li>';
32         }
33         echo '</ul>';
34     } else {
35         echo '<p>Thank you for contacting me!</p>';
36     }
37
38 }
```

Figure 3-2 shows what happens should the user forego providing his name and phone number.

The screenshot shows a web form with several fields and validation messages. At the top, a yellow-bordered box contains two error messages: "Please provide your name." and "Please provide a valid phone number." Below this, there are input fields for "Your name" (empty), "Your e-mail" (jason@example.com), and "Your phone number" (empty). A dropdown menu for "Preferred contact method" is set to "E-mail". A text area for "What is your favorite video game?" contains the text "Super Mario Brothers 3!". Below the form is a question "Subscribe to the gaming newsletter? Yes" and a "Send!" button.

- Please provide your name.
- Please provide a valid phone number.

Your name:

Your e-mail:

Your phone number:
 () -

Preferred contact method:

What is your favorite video game:

Subscribe to the gaming newsletter? Yes

Figure 3-2. Validating form data

Figure 3-2 demonstrates that the errors display as expected, but if you're particularly observant you noticed those form fields which have been completed by the user are also filled in following the form submission! How was this accomplished is the subject of the next step.

Step #3. Repopulating Form Data

When users erringly submit an invalid phone number or e-mail address, the last thing they're expecting is to be penalized by the mistake, with the penalty coming in the form of having to fill the form out anew even if just one invalid field was submitted or if a field was overlooked. Forcing the user to do so could very well cause him to abandon registration or the transaction out of frustration, costing you valuable traffic and revenue.

Of course, you'll also populate forms in instances when users need to update existing data. For instance, Figure 3-3 displays GameNomad's user profile editing screen, which draws on information the user entered when registering or previously updating his profile.

Manage Your Profile

Use this interface to manage your profile.

wjgilmore (Jason Gilmore)

Your City:

Your State:

Your Province (if outside U.S.A):

Your Zip Code:

Your Country:

Greatest Game Ever and Why:

Halo 3 because it rocks

Save Changes

Figure 3-3. GameNomad's profile management interface

Updating text fields and text areas is very easy, just echo the POSTed form variable back into the field, like so:

```
<input name="name" type="text" size="35" value="<?php echo $_POST['name']; ?>" />
```

Of course, if you're populating a form for reason of updating information, you won't use data hailing from the POST array; instead you'll assign the data (typically drawn from a database, as you'll learn in Chapter 4) to an appropriate variable and use that instead:

```
<input name="name" type="text" size="35" value="<?php echo $name; ?>" />
```

Depending upon how PHP's error reporting level is configured, you could receive error notifications of level notice stating that those variables not explicitly assigned (as would be the case if the user forgot to complete a field). To avoid these errors, you could either change PHP's error reporting level to ignore notice-level errors, disable error reporting, or preferably set the variable even if it is empty. One syntactically compact way to do this is using the *ternary operator*. The ternary operator sports some rather strange syntax but its power is in succinctness. Here's an example:

```
$name = isset($_POST['name']) ? $_POST['name'] : '';
```

Before you start gnashing your teeth over the bizarre syntax, take a moment to consider a generalized form of the ternary operator's syntax:

```
$variable = condition ? TRUE : FALSE
```

In other words, set `$variable` to the value specified by the `TRUE` placeholder if the condition eval-

ates to true, otherwise set \$variable to the value specified by the FALSE placeholder. Therefore in the case of the previous example, if `$_POST['name']` has been set, we'll set `$name` to `$_POST['name']`, otherwise we'll set it to a blank value.

The ternary operator is equally useful for selecting the correct option in a select box, or checking an appropriate checkbox. For instance, the following example will select the `email` option should `$contact_method` be set to `email`.

```
<option value="email" <?php echo $contact_method == 'email' ? 'selected' : '';
?>>E-mail</option>
```

Step #4. Sending Form Data via E-mail

Once the form has been submitted and its contents validated, it's time to send the data to your inbox. But how is a message transferred from your website to your e-mail address? This answer is easier than you think, given it's such a commonplace task. PHP has long offered a function called `mail()`, which can send a simple e-mail to a recipient:

```
01 <?php
02
03     $recipient = 'recipient@example.com';
04     $subject = 'Hello stranger!';
05     $from = 'jason@example.com\r\n';
06     $message = 'Hope all is well!';
07
08     mail($recipient, $subject, "From:$from", "$message");
09
10 ?>
```

While this short script should be fairly self-explanatory, configuring PHP to send e-mail is often a source of significant confusion. This confusion stems from the differing operating system-specific approaches required to ensure this feature works properly.

If you're executing a script involving `mail()` on Unix/Linux/OS X, this script will likely work because these operating systems almost invariably come with a mail transfer agent (MTA) such as Sendmail installed. However, because the Windows operating system does not offer a similar feature, you'll need to configure PHP to allow it to talk to another mail server.

Configuring `mail()` on Windows

Unlike most, if not all, Linux and Unix platforms, the Windows 2000, XP, ME, and Vista operating systems do not come equipped with a mail server, meaning you'll need to rely on a third-party SMTP server to send e-mails. To configure PHP to use this SMTP server, open your `php.ini` file and search for the following lines:

```
[mail function]
; For Win32 only.
SMTP = localhost
```

```
smtp_port = 25
; For Win32 only.
;sendmail_from = me@example.com
```

To use a remote mail server, you'll need to change the SMTP directive to the remote mail server's domain name or IP address. For instance, the GameNomad mail server is `mail.gamenomad.com`. If your mail server uses an SMTP port of something other than the standard port 25, you'll need to change the `smtp_port` directive as well. If you plan on sending all mail using a single address, you can optionally set the `sendmail_from` directive, however you'll likely want to set the address at the time you send the e-mail, which will be demonstrated in the next section.

Of course, you can't just randomly choose a mail server and send e-mail through it. Most servers (Google's GMail service, for instance) require you authenticate the sender before sending e-mail, and unfortunately PHP has yet to offer a way to authenticate the sender, leaving Windows users' without a particularly attractive solution when it comes to sending e-mail from a PHP-driven website. There is however an easy alternative, discussed in the later section, "Introducing Mail".

Using PHP's `mail()` Function

As demonstrated earlier in this step, the `mail()` function presents PHP developers with a native solution for sending e-mail through a PHP script. If you're not subject to the Windows-specific configuration drawbacks as described in the previous section, the previous example showed that using this function to send e-mail couldn't be easier. You might however be wondering how to tweak the function to include other useful attributes such as the reply-to address (`Reply-To`), carbon copy (`Cc`) and blind carbon copy (`Bcc`) recipients. For instance, if you wanted to CC another recipient, the `mail()` function's third parameter would look like this:

```
From:jason@example.com\r\nCc:theboss@example.net\r\n
```

Note how each attribute concludes with the string `\r\n`. This is a special character sequence which standards-compliant mail servers rely on to recognize each attribute separately. Neglecting to include this string could result in unintended consequences so be sure to remember it!

Creating E-mail Templates

Finally, the above example used a very short message simply for purpose of illustration, however in practicality you'll likely send something much longer. I prefer to maintain these e-mail templates in a separate file, and then use the `require()` statement to pull them into the script for mailing purposes as necessary. Because the e-mail message typically includes customized data set before mailing the message, I store the message in a PHP variable using what's known as `heredoc` syntax. The heredoc syntax is simply an easy way to delimit large amounts of text for variable assignment. Here's an example:

```
01 <?php
02 $message = <<<message
03 Dear {$user},
```

```
04
05 Thank you for registering! Your e-mail address has been confirmed. To begin
06 managing your games, login to your account at http://www.gamenomad.com/.
07
08 Thank you,
09 The GameNomad Team
10 message;
11 ?>
```

This example demonstrates several important points about heredoc usage:

- Line 02 defines the variable which will store the message text. The <<<message acts as the opening text delimiter, much as the double quotes do when assigning text to a variable as you've seen in previous examples.
- Line 03 demonstrates how a variable can appear within the text for later evaluation. When the file is included within a script, this \$user variable will be evaluated in the scope of the script including the file, meaning if a variable named \$user appears in the script, the corresponding value will be assigned to the \$user variable appearing in the mail message.
- Line 10 displays the ending delimiter. Notice this is named identically to the opening delimiter label, *and* that it appears in the very first column of the script (not taking the line numbers into account of course).

Save the e-mail template as a PHP file (`mail-registration.php` for instance), and then include it as necessary, like so:

```
<?php

// User information drawn from database or elsewhere
$user = 'Jason';
$email = 'jason@example.com';

// Specify e-mail subject and sender
$subject = 'GameNomad: Thank you for registering';
$sender = 'users@gamenomad.com';

// Include the e-mail template
require '/templates/mail-registration.php';

// Send the message
mail($user, $subject, "From:$sender\r\n", $message);

?>
```

Remember, when using the `include()` or `require()` statement you'll need to make sure the location of the file being included is found in PHP's include path.

Introducing Mail

If you're on Windows and require an e-mail transmission solution, chances are PHP's default features aren't going to be of much help. However there are numerous alternative solutions which make the task a breeze. One such solution is the Mail PEAR package, available from <http://pear.php.net/package/Mail>. Mail is a great solution not only because it makes it easy to identify which of the three supported mailing solutions you'd like to use (the configuration made available to PHP's `mail()` function through the `php.ini` file, Sendmail, or SMTP), but also because it provides a programmatic interface for performing tasks you can't otherwise easily do with the standard PHP installation, such as provide SMTP authentication parameters.

Installing the Mail Package

To install Mail, execute the following commands:

```
%>pear install -o Mail  
%>pear install -o Net_SMTP
```

Some installation-related output will scroll by following each command, followed by a message confirming successful installation. That's it! You're ready to begin using Mail.

Using the Mail Package

You send e-mail using the Mail package in two steps:

1. Identify the supported mailer mechanism. This decision is typically based on the capabilities of your operating system, but its flexibility allows you to choose the solution most suitable to your particular needs.
2. Send the message by specifying the recipients, headers (including attributes such as the subject, cc, and bcc), and the message body.

The following script demonstrates how to send an e-mail through Google's GMail service (which I use to host mail services for W.J. Gilmore, LLC). Following the script is a breakdown of the relevant lines:

```
01 <?php  
02  
03     // Include the Mail package  
04     require 'Mail.php';  
05  
06     // Identify the sender, recipient, mail subject, and body  
07     $sender = 'wj@wjjgilmore.com';  
08     $recipient = 'wjjgilmore@gmail.com';  
09     $subject = 'GameNomad: Thank you for registering';  
10     $body = 'Thank you for registering!';  
11  
12     // Identify the mail server, username, password, and port
```

```
13 $server = 'ssl://smtp.gmail.com';
14 $username = 'wj@wjgilmore.com';
15 $password = 'supersecret';
16 $port = 465;
17
18 // Setup the mail headers
19 $headers = array(
20     'From'      => $sender,
21     'To'        => $recipient,
22     'Subject'   => $subject
23 );
24
25 // Configure the mailer mechanism
26 $smtp = Mail::factory('smtp',
27     array(
28         'host'      => $server,
29         'username'  => $username,
30         'password'  => $password,
31         'auth'       => true,
32         'port'       => $port
33     )
34 );
35
36 // Send the message
37 $mail = $smtp->send($recipient, $headers, $body);
38
39 if (PEAR::isError($mail)) {
40     echo ($mail->getMessage());
41 } else {
42     echo('Success!');
43 }
44
45 ?>
```

The code breakdown follows:

- Line 04 includes the Mail package. Remember you'll need to make sure PHP's `include_path` is set to include the location where your PEAR packages are located.
- Lines 07-10 identify the sender, recipient, subject, and message. Of course, chances are you'll want to use an e-mail template as was described in the earlier section, "Creating E-mail Templates"
- Lines 13-16 identify the mail server, username, password, and port. Because Google requires secure authentication be used, you'll need to make sure PHP's OpenSSL extension is properly configured. This means not only uncommenting the `php_openssl.dll` line within the `php.ini` file, but also copying the `libeay32.dll` file (found in your PHP installation's home directory) into Windows' `system32` directory. If you find these instructions confusing, watch the screencast described below for more guidance.

- Lines 19-23 create the array which will subsequently be used to configure the message
- Lines 26-34 identify the mailer mechanism (in this case, SMTP), and pass in the necessary parameters to properly configure that mechanism
- Lines 37-43 send the e-mail, producing an appropriate message depending upon the outcome.

Step #5. More on Securing User Input

Step #2 of this chapter is devoted to validating user input, or ensuring the input meets certain pre-defined characteristics. For instance, we required the user to provide his name, and so would not process the form until a name was provided. Of course, we could have been much more exacting in our validation procedure, because there's nothing to prevent the user from entering "5933" for instance, which wouldn't be any more useful than had he provided no name at all.

However, this inability to foresee what the user might enter underscores a much larger problem than mere inconvenience. Left unchecked, there's a real possibility malicious user input could not only do serious damage to your website's data and general accessibility, but also result in your website being used as a conduit for attacking other users. For instance, suppose your website gave users the opportunity to post comments about published articles by way of a web form. These comments were then immediately appended to a list at the conclusion of the page. But because you neglected to properly filter user input, some joker posted the following data within a comment:

```
I hate your website! <script language='JavaScript'>document.location='http://www.example.com/'</script>
```

Because the JavaScript `document.location` call causes the webpage to redirect to whatever URL is assigned to `document.location`, this so-called comment will prevent others from not only reading your article, but participating in the conversation by adding their own (presumably legitimate) comments. To say the least, this is not the sort of participation you anticipated.

But the consequences of not properly filtering user input can be far more serious. Because the source code of most websites can be easily obtained simply by viewing the source through a Web browser, what's to stop somebody from recreating your website and hosting it at `http://bogusgamenomad.example.com/`? By doing so and then redirecting users from your site to the bogus site using the above-described `document.location` call, there's a great possibility unsuspecting users could be duped into providing personal information such as credit cards or other valuable data. Even worse, your organization is likely to be the target of user venom in the aftermath.

These examples highlight only a few of the very serious issues which could arise if user input is left unchecked, and in subsequent chapters I'll discuss others as the opportunity arises. However, there are a few built-in PHP functions you can start using right now to stem these issues:

- **Stripping HTML Tags:** Use the `strip_tags()` function to completely remove HTML and PHP tags from user input. An optional parameter gives you the ability to define allowable tags should you deem them permissible.
- **Converting Special Characters:** Use the `htmlspecialchars()` function to convert char-

acters such as < to their HTML entities (in this example, <). This results in the characters being displayed *exactly* as entered, rather than parsed and rendered by the browser as if they were actual HTML.

Let's consider two examples. Suppose the user attempted to enter the above comment, but this time you filtered it through `strip_tags()` before publishing it on the website:

```
<?php  
  
    $input = "I hate your website! <script  
language='JavaScript'>document.location='http://www.example.com/';</script>";  
  
    echo strip_tags($input);  
  
?>
```

Executing this script produces the following output. Notice the `<script>` tags have been removed, meaning the embedded JavaScript can no longer execute!

```
I hate your website! document.location='http://www.example.com/';
```

Now run the same script, but this time pass the input through `htmlspecialchars()` instead of `strip_tags()`. This time, the following string will be output (to the browser, which will in turn convert the HTML entities back to their counterparts, rather than rendering the text as HTML):

```
I hate your website! &lt;script language='JavaScript'&gt;document.  
location='http://www.example.com/';&lt;/script&gt;
```

Notice how the special characters have been converted to their HTML entities? This prevents them from taking on special meaning within the browser, thereby preventing the JavaScript from executing.

As you can see, you can completely remove a serious security problem with minimal code. Do yourself a favor and do so at every opportunity!

Conclusion

In just three chapters, we've covered significant ground in terms of giving you the tools to not only gather user data, but validate this data, and transmit it via e-mail to your inbox. We also discussed a few serious security issues which could arise if user input is left unchecked.

In the next chapter, we'll examine another key topic in the world of web development: databases. Specifically, I'll introduce the MySQL database, showing you how to import your spreadsheet data into MySQL, and convert GameNomad to a fully database-driven website. Onwards!

CHAPTER 4

Introducing MySQL

Many newcomers to web development instinctively try to leverage familiar tools such as Microsoft Excel or Microsoft Access when building their first website. However, their limitations soon become apparent. For starters, data transfer is a one-way street; you or your friends can't easily move data provided via the website into the spreadsheet. What if you just bought a really cool new game and wanted to let your friends know about it immediately using your iPhone? Further, relying on a file-based data storage solution as demonstrated in Chapter 2 inhibits you from easily implementing key features such as search, and enhancing the site with the ability for users to add reviews of games found in the directory.

The solution? Move your application data to a database. Don't worry, it's not as hard as it sounds! In this chapter you'll learn about a powerful, free database called MySQL. You'll also learn how to migrate your game collection to MySQL, and create Web interfaces for not only accessing this data, but also adding, modifying, and deleting it!

Chapter Steps

The goals of this chapter are accomplished in six steps:

- **Step #1. What Is a Relational Database?**: We'll kick things off by introducing you to the concept of a relational database, and learn why it's such a useful tool for efficiently managing data.
- **Step #2. Introducing MySQL**: In this step you'll become acquainted with the MySQL database and learn how MySQL interprets the data stored within it.
- **Step #3. Introducing phpMyAdmin**: In this step you'll learn how to use phpMyAdmin to create the website database and the tables which will host the game collection, along with a user granted sufficient privileges to manage the database.
- **Step #4. Moving Your Data to MySQL**: You've successfully configured MySQL and created the data structures necessary to manage your application's data, but how do you go about actually importing the existing data into MySQL? It's easier than you think, and in this step you'll learn how.
- **Step #5. Connecting Your Website to MySQL**: In this step you'll learn how to connect to the MySQL database from a PHP script, and build the pages which will make it possible to add new games, modify existing game information, and even delete games from your collection directory, all through the web!
- **Step #6. Restricting Access**: Clearly you'll want to restrict access to the game management interfaces, otherwise malicious users might take the opportunity to modify or delete your data. In the final step of this chapter I'll show you an easy way to password-protect any script.

Step #1. What Is a Relational Database?

Even the free spirits among us have to admit we live within the bounds of a very structured environment. It just seems to be the way we're wired; whether it's geology, biology, libraries, or the stock market, mankind seems obsessed with assigning order to disorder.

As developers, it makes sense to model structures within code in a way that most closely resembles the world around us. Indeed, this sentiment has been shared by the majority of the computer science community for the past 30 years, with much of the thinking having been based around a seminal paper on the topic of *relational data modeling* by IBM researcher E.F. Codd.

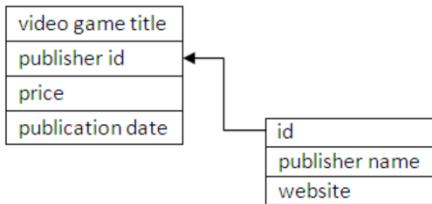
VIDEO. Introducing the Relational Database

Beginning web developers quickly realize learning a programming language is only half of the battle. They'll also need to figure out how to manage the website data, a task most commonly done using a relational database. This video introduces the concept, helping you to understand why relational databases play such a pivotal role in web development. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

In plain terms, Codd's proposed solution revolves around the notion of being able to effectively manage data by striving to eliminate redundancy at every opportunity. This process is known as *normalization*, and as you'll soon see, ensuring your data is properly normalized will make your life as a developer much more appealing. Let's start with a simple example. Suppose some future version of your game collection application included the name of each publisher. Conceptually the data structure (known as a *table* in relational database parlance) might look like this:

| |
|------------------|
| video game title |
| publisher name |
| price |
| publication date |

Based on this approach, logically you'll need to manually enter the publisher's name each time a new game is added to your collection. It's entirely likely you'll come to own several games tied to the same publisher, Midway Games for instance. As you insert new games over a period of time, several variations of the company name begin to appear: MidwayGames, Midway, Midwaygames, and when you're particularly lazy, MG. The result of the many variations is that it's now impossible to effectively filter your collection to view just those games published by Midway Games. Compounding the potential problems, suppose Midway Games one day decided to change their name, thereby forcing you to manually update every row containing the old name. So how can these sorts of problems be avoided? By placing the publishers into their own data structure, and then using some sort of key to tie (or *relate*) each game in the games table to the appropriate record (also known as a *row*) in the publishers table:



Problem solved! While there's a tad more overhead involved in terms of maintaining the second table and building the Web interfaces capable of managing it, you're now guaranteed to avoid any inconsistencies, not to mention greatly decrease the time required to change a publisher name. This approach also enhances our ability to store other information about each publisher, such as the website URL.

In a nutshell, you've just learned the basic premise behind relational databases and the sound design principles which help you to effectively manage data. This is however just one of several normalization rules proposed by Dr. Codd. As your database grows in size and complexity, you'll want to model the structures in accordance with the other rules.

VIDEO. Database Normalization: The First Three Rules

A database is generally considered to be fully normalized if it abides by Codd's first three normalization forms. This video introduces these three forms, complete with several demonstrations showing how a denormalized database (a database lacking all normalization) can be transformed into a fully normalized version. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Step #2. Introducing MySQL

In the first chapter you installed MySQL, quite possibly without even really understanding what MySQL even is, beyond the fact it's a database. If this statement applies to you, this section will help dispel any confusion. Otherwise, if you're already familiar with MySQL's background and relational databases in general, skip ahead to the section "Introducing phpMyAdmin".

MySQL is a database first released over a decade ago by a group of enterprising developers hailing from a small Swedish software company. Originally a side project, MySQL soon commanded an increasingly large share of the developers' time, prompting them to eventually pursue the new project full time, forming a new company named MySQL AB. Fast forward ten years, countless improvements, and millions of dollars in venture capital, and their efforts culminated in the sale of the company to Sun Microsystems for almost \$1 billion in early 2008.

So what was it that caused this side project's popularity to soar to meteoric heights? MySQL's advantages are many, notably including:

- **It's free:** It might surprise you that MySQL is free, and due to its licensing arrangement, will always remain so! Commercial licenses are also available should you desire to use MySQL in a manner which conflicts with its liberal open source licensing terms, or if you'd like to take advantage of official technical support.
- **You can talk to it using dozens of languages:** PHP, Java, C#, C, C++, Perl, Python, and Ruby are just a few of the languages you can use to interact with MySQL.

- **It's flexible:** One of the reasons for MySQL's soaring popularity is its flexibility. Although its primary use is as a database for powering websites, you'll also find MySQL playing a key role within products and services found in industries such as finance, defense, and entertainment.
- **It's as complex (or as simple) as you want it to be:** Like PHP, MySQL's developers have always put simplicity and performance ahead of an everything-but-the-kitchen sink approach to adding features. Although today's MySQL now offers all of the features desired by the majority of database administrators, this no-frills approach has historically been quite attractive to database newcomers, contributing greatly to MySQL's popularity.
- **There are numerous support outlets:** Community support is another huge advantage for MySQL users, with hundreds of discussion groups available online, and countless fellow users ready to answer your questions. If you prefer a more formalized solution, you can purchase varying levels of support from Sun Microsystems or from other vendors.

Later in this chapter you'll learn how to interact directly with the MySQL database using a Web-based MySQL administration application called phpMyAdmin. First however I'd like to introduce you to some general database concepts which will help you to understand how MySQL works.

How MySQL Manages Data

When working with a spreadsheet you think in terms of rows, columns and cells. The same concept applies to databases, except that the database world uses the term *table* rather than the term spreadsheet. Each database typically consists of several tables, and each table consists of multiple *rows* and *columns*.

All modern database solutions, MySQL included, can host several databases, each of which might contain multiple tables. You might use one database to store the data found on your corporate website, another to manage your blog, and still another to manage sales data. Each table column is defined by a name and *data type*. These data types formally define the kind of data stored in the column, playing a very important role in matters such as determining the amount of space required to store the data, and how MySQL interacts with the data.

VIDEO. Introducing the MySQL Client

Command-line junkies will appreciate the MySQL client's powerful yet minimalistic interface, capable of managing practically every aspect of an installation with just a few clicks of the keyboard. But ferreting out the proper commands can take some time. This video seeks to jumpstart your use of this powerful utility by showing you the most commonly used commands. Watch the video at <http://www.easyphpwebsites.com/zfw/videos/>.

MySQL's Data Types

Each table column is intended to store a specific type of data, such as a date, string, or integer value. When creating a table, you formally define the type of data each column is intended to store by specifying a data type. MySQL supports almost 30 such data types including 14 data types for storing strings alone. For instance, if you wanted to create a table containing information about your games and wanted to store the title, you'd define the column using the VARCHAR type, which stores strings of

varying lengths up to 65,535 characters. To store the release date, you'd use the DATE type. Quite a few other supported types exist, including variants intended to maximize MySQL's performance and minimize storage waste. Long snippets of text can be stored using a variety of data types, such as MEDIUMTEXT (maximum 16,777,215 characters), and LONGTEXT (maximum 4,294,867,295 characters). Table 4-1 enumerates some of MySQL's more commonly used data types.

Table 4-1. Common MySQL Data Types

| Data Type | Range |
|------------|---|
| CHAR | 0 to 255 characters |
| VARCHAR | 0 to 65,535 characters |
| MEDIUMTEXT | 0 to 16,777,215 characters |
| TINYINT | -128 to 127 |
| INT | -32,768 to 32,767 |
| FLOAT | -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38 |
| DATE | 1000-01-01 to 9999-12-31 |
| DATETIME | 1000-01-01 00:00:00 to 9999-12-31 23:59:59 |
| TIMESTAMP | 1970-01-01 00:00:01 to 2038-01-09 03:14:07 |

You might be wondering why MySQL is so exacting when it comes to storage definitions. If you know all dates will be maintained using the format YYYY-MM-DD, why not just use CHAR(10), which will set aside exactly ten characters for the date? For that matter, why not just use text-specific data types to store everything from release dates to titles to prices? For starters, MySQL's storage requirements vary widely according to the chosen data type, which over time could result in significant consumption of storage space if your data grows sufficiently large. To put this into perspective, using CHAR(10) to store a date using the aforementioned format will require more than three times the storage space than otherwise needed had you used DATE.

Although seemingly limitless hard drive space can be purchased for less than a thousand dollars, there's a secondary and perhaps even more important reason for judiciously applying MySQL's data types. When querying data, MySQL relies on these data types being properly defined when using them in conjunction with query functions. For instance, MySQL supports over fifty date- and time-related functions which can be used to perform temporal calculations and conversions when querying data. Of course, these functions can only be expected to properly work when used in conjunction with a date or time type. Likewise, MySQL supports numerous functions for manipulating strings and numbers. Later in this section I'll introduce you to some of the powerful tasks you can perform using these functions.

The takeaway? MySQL's broad support of data types isn't without reason. When designing your data structures, it's in your best interests to use these data types to model your data in the most exacting way possible, not only minimizing your storage requirements but also ensuring you're able to use of MySQL's full complement of query functions.

So what might a table used to store games look like? MySQL supports a CREATE TABLE command which can define a new table. The syntax itself isn't so important, since later in this chapter I'll show

you how to use phpMyAdmin to create tables using a graphical interface; for the moment just pay particular attention to the data types being used so you can start getting familiar with their purpose:

```
CREATE TABLE games (
    title VARCHAR,
    release_date DATE,
    price DECIMAL(4,2)
);
```

In this example, the game title can consist of up to 65,535 characters, the release date can occur anywhere between January 1, 1000 and December 31, 9999, and the price can consist of a number as large as \$99.99, because (4,2) is representative of a decimal value consisting of a total of four digits, with two digits representing the number of digits which can follow the decimal point.

While a good start, there's actually much more we can do to formally define the data stored within the `games` table. Most notably, we probably want to make sure these columns are unable to accept blank values, introduced next.

Defining a Column as Not Null

Identifying the data type goes a long way towards defining the specific role of a table column, but with data type attributes you can do much more. Type attributes place additional constraints on the data type, forcing you to be even more specific regarding the column's contents. One commonly used attribute is `NOT NULL`, which forces a value to be provided. Because we want each row in the table to include the title, release date, and price, it makes sense to add this attribute to each column:

```
CREATE TABLE games (
    title VARCHAR NOT NULL,
    release_date DATE NOT NULL,
    price DECIMAL(4,2) NOT NULL
);
```

With this change, any attempts to insert a row lacking data for each column will result in an error.

Defining Signed/Unsigned Integers

Suppose you instituted some sort of starred rating system for your games, ranging from 1-10, 10 being great. Because this system consists of a range of integers (1-10), you would logically use the `TINYINT` data type to represent the column. However, because these values will never be negative, you can further constrain the value range by making it an unsigned integer:

```
CREATE TABLE games (
    title VARCHAR NOT NULL,
    release_date DATE NOT NULL,
    price DECIMAL(4,2) NOT NULL,
    rating TINYINT UNSIGNED NOT NULL
);
```

Because signed integer values are the default, there's no need to explicitly define integers as such. Incidentally, identifying an integer-related column as unsigned will result in a shift of its range to be entirely positive. For instance, a signed TINYINT has a range of -128 to 127, whereas an unsigned TINYINT has a range of 0 to 255.

Assigning Default Values

Suppose you might not necessarily wish to rate a game at the time it's added to your collection. After all, it's entirely likely you will eventually purchase a game but not be able to immediately spend adequate time playing it and so feel uncomfortable assigning a rating at the time it's inserted into the database. However you don't just want to assign a value in the range of 1-10 because it could mislead your friends into thinking you've already rated the game. Therefore you could assign the column a default value of 0:

```
CREATE TABLE games (
    title VARCHAR NOT NULL,
    release_date DATE NOT NULL,
    price DECIMAL(4,2) NOT NULL,
    rating TINYINT UNSIGNED NOT NULL DEFAULT 0
);
```

Creating Primary Keys

As you learned earlier in this chapter, a successful relational database implementation is dependent upon the ability for table rows to reference each other using some unique value. This value which uniquely identifies each row is known as a *primary key*, and it absolutely must be unique for each and every row, no matter how large the table grows. As an example, if you're building a product catalog using of the Amazon Associates Web Service (discussed in Chapter 10), you might consider using the unique codes Amazon.com uses to identify every product, known as the ASIN (Amazon Standard Identification Number):

```
CREATE TABLE games (
    asin CHAR(10) NOT NULL PRIMARY KEY,
    title VARCHAR NOT NULL,
    release_date DATE NOT NULL,
    price DECIMAL(4,2) NOT NULL,
    rating TINYINT UNSIGNED NOT NULL DEFAULT 0
);
```

However, there's actually a much easier way to create primary keys: the automatically incrementing integer. If when adding a new row you increment the previously inserted row's primary key by one, you're guaranteed to never repeat an existing value. This is such a common strategy that MySQL, like all other mainstream databases, supports the ability to identify an integer-based column as an automatically incrementing primary key:

```
CREATE TABLE games (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    asin CHAR(10) NOT NULL,
```

```
title VARCHAR NOT NULL,
release_date DATE NOT NULL,
price DECIMAL(4,2) NOT NULL,
rating TINYINT UNSIGNED NOT NULL DEFAULT 0
);
```

You're free to name the primary key anything you please, however for purposes of consistency I always identify mine using the term `id`. In the next section, why I use this particular term will become more apparent.

NOTE. As you'll see later in this chapter, when inserting a new row into a table which contains an automatically incrementing primary key, you don't actually specify a value for that column. Instead, depending on the format of the query used (MySQL supports several), you either identify it as `NULL`, or ignore the column altogether.

Creating Table Relations

Video games are often released for play on multiple platforms, and as an avid gamer it's entirely likely you could own the same game for both the Xbox 360 and Nintendo DS. Of course, each of these games must be treated separately despite having identical titles, and so you'll need to record each, along with their respective platform, in the `games` table.

The easy solution would be to add this column to the `games` table and proceed to type the platform name in each time you add a game:

```
platform VARCHAR(100) NOT NULL
```

However, over time it's a given you'll wind up with numerous variations of the platform name, for instance `Xbox 360`, `XBox 360`, and `360`. You might even be particularly lazy one day and enter simply `Xbox`, which could lead your friends to believe you purchased an original Xbox rather than the newer 360 model. The result is a database rife with inconsistency, leading not only to user confusion, but also hindering the possibility of being able to filter the collection according to platform.

Instead, the proper way to track platform information is to create a new table consisting of just a primary key and platform name:

```
CREATE TABLE platforms (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL
);
```

Then, you'll refer to the appropriate platform's primary key within the `games` table:

```
CREATE TABLE games (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    asin CHAR(10) NOT NULL,
    platform_id INTEGER UNSIGNED NOT NULL,
    title VARCHAR NOT NULL,
```

```
release_date DATE NOT NULL,  
price DECIMAL(4,2) NOT NULL,  
rating TINYINT UNSIGNED NOT NULL DEFAULT 0  
);
```

Now, instead of typing the platform name each time a game is created, you'll retrieve the list of platforms found in the games table, and use an HTML form control such as the select box to choose the desired platform. You'll learn how to do this later in this chapter. Figure 4-1 depicts the HTML form evolution from one requiring you to manually enter the platform name to one drawing from the platforms table to produce a select box.

The figure consists of two side-by-side screenshots of a web form. On the left, the 'Game platform:' field is a simple text input box. On the right, it is a dropdown menu with several options. A circle containing the text '-VS-' is placed between the two forms to indicate a comparison.

Figure 4-1. Revising the game insertion form to use a platform select box

Step #3. Introducing phpMyAdmin

MySQL is packaged with a number of command-line tools which help you to manage all aspects of your database, from the users to the actual databases, tables, and data. However, many developers prefer to use graphically-oriented applications, allowing them to point and click rather than have to memorize often obscure commands and options. MySQL users are in luck, because a fantastic Web-based MySQL manager named phpMyAdmin has long been freely available.

NOTE. If you're on Windows and would like to experiment with or use MySQL's command-line utilities, consider downloading Console (<http://sourceforge.net/projects/console/>). Console is a great alternative to Windows' cumbersome command prompt, offering a resizable window, multiple tabs, simple text-selection, and many more features. Like PHP and MySQL, Console is released under an open source license, meaning it's free to download and use.

You can experiment with a live phpMyAdmin demo server by navigating to <http://pma.cihar.com/STABLE/>.

VIDEO. Getting Acquainted with phpMyAdmin

phpMyAdmin is the world's most popular web-based MySQL administration utility, useful for managing nearly every aspect of a MySQL installation. This video introduces you to several of the utility's most useful features. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Installing phpMyAdmin

To install phpMyAdmin, head over to <http://www.phpmyadmin.net/> and download the latest stable release. You'll see each release offers numerous variations, however ultimately these variations are grouped into two distinct categories: English and other languages. Presuming you're a native English speaker, choose the `english.zip` (Windows) or `english.tar.gz` (Linux) download.

Once downloaded, extract the files to a directory with a convenient name (I suggest `phpmyadmin`). Open the file named `config.sample.inc.php` found in phpMyAdmin's root directory, and save it as `config.inc.php`. This is phpMyAdmin's sample configuration file, and it contains several configuration directives for controlling phpMyAdmin's behavior. To begin using phpMyAdmin however you only need to modify one directive, namely `$cfg['blowfish_secret']`, which is used in the password encryption process:

```
$cfg['blowfish_secret'] = '';
```

I suggest simply inserting a string of random characters, as you'll never need to refer to this value again. Once complete, head over to your phpMyAdmin login page by navigating to the installation directory. Presuming you're working with a fresh MySQL installation, you can login using the username `root` and an empty password.



Figure 4-2. phpMyAdmin's login page

NOTE. Don't fret if the login page displays an error message regarding the inability to load the mcrypt extension. This extension is only required for 64-bit systems, so chances are the message will be irrelevant. If you'd rather the message did not appear, you can configure this extension, a topic out of the scope of this book but easily done after searching Google for instructions.

Creating a New MySQL User

Once logged in, you'll be presented with a summary screen, including pointers to high-level management tasks such as viewing MySQL's configuration information and currently running processes.

On the left side of the screen you'll see a list of available databases, including at least one named `mysql` (if you're using a version of phpMyAdmin installed by your hosting provider, this won't be the case). This database contains information regarding which users can access the MySQL server, and denoting what tasks they can perform. The root user has default access to all installed databases, and can perform every available task, including shutting down the database server! Because of this database's important role, you should never delete it or directly modify its tables. Given such power, it's not wise to use the root user for general interaction, and so we're going to create a new user which we'll subsequently use within the GameNomad application.

 Add a new User

Login Information

User name: Use text field:
Host: Any host
Password: Use text field:
Re-type:
Generate Password:

Database for user

None
 Create database with same name and grant all privileges
 Grant all privileges on wildcard name (username_%)

Global privileges (Check All / Uncheck All)

Note: MySQL privilege names are expressed in English

Data SELECT INSERT UPDATE DELETE FILE

Structure CREATE ALTER INDEX DROP CREATE TEMPORARY TABLES SHOW VIEW CREATE ROUTINE ALTER ROUTINE EXECUTE CREATE VIEW EVENT TRIGGER

Administration GRANT SUPER PROCESS RELOAD SHUTDOWN SHOW DATABASES LOCK TABLES REFERENCES REPLICATION CLIENT REPLICATION SLAVE CREATE USER

Resource limits

Note: Setting these options to 0 (zero) removes the limit.

MAX QUERIES PER HOUR
MAX UPDATES PER HOUR
MAX CONNECTIONS PER HOUR
MAX USER_CONNECTIONS

Figure 4-3. Creating a new MySQL user

To create a new user, click on the "Privileges" link located on the phpMyAdmin home page. You'll be presented with a tabulated list of users. To create a new user, click the Add a new User link. You'll be first prompted to create a new user by specifying a user name, host (the user's originating location), and a password.

Regarding the username, for this project I suggest using `domainname_prod`, (I tend to create several users to interact with a website database, typically calling them `domainname_dev`, for the development site, and `domainname_prod`, for the production site. While it's out of the scope of this book to discuss logistical matters such as how to manage development and production stages, these are nonetheless matters you should start considering as your project grows in size) as this is the same name we'll use for the database.

Regarding the Host value, I'll presume you'll be running phpMyAdmin from the same computer in which the server is installed, so set this to `Local`. You do however have the ability to set this to a remote IP address or hostname (or even wildcard-based IP addresses or hostnames if you were connecting from one of several clients in a network), should you happen to run your database on one server and your website on another. Chances are you're running everything on a single machine, and so setting this to `Local` will likely suffice.

Finally, create a password, or click the Generate button to have one created for you. If you choose the latter option, be sure to also click the Copy button so the password is copied into the respective password fields. If you choose the former, be sure to choose a password of sufficient strength, preferably consisting of a combination of letters, numbers, and symbols (such as !, %, and ^).

Next select the "Create database with same name and grant all privileges" option. Once the Go button is pressed, this will result in a new database named `domainname_prod` being created, and will grant the `domainname_prod` user with all of the privileges necessary to effectively manage the database. Test the newly created user account by logging out of phpMyAdmin (click the Exit button at the top left of the screen), and logging in anew with the `domainname_prod` account.

Step #4. Moving Your Data to MySQL

You'll need to create the `games` and `platforms` tables used earlier in this chapter, and then migrate your spreadsheet data into the database. While you could manually insert the data into the newly created table, in this section you'll learn how to import the spreadsheet data directly into the table.

Creating the games Table

For the purposes of the exercises found in the remainder of this chapter, the final version of the `games` table looks like this:

```
CREATE TABLE games (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    platform_id TINYINT UNSIGNED NOT NULL,
    title VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    price DECIMAL(4,2) NOT NULL,
    created_on TIMESTAMP NOT NULL
);
```

Let's create this table using phpMyAdmin. To begin, click on the `domainname_prod` link located on the left side of phpMyAdmin to enter that database. You'll be greeted with a prompt similar to the one shown in Figure 4-4. Enter the name `games` and specify six fields (one for each column as defined in the above table definition), subsequently clicking the Go button to continue.

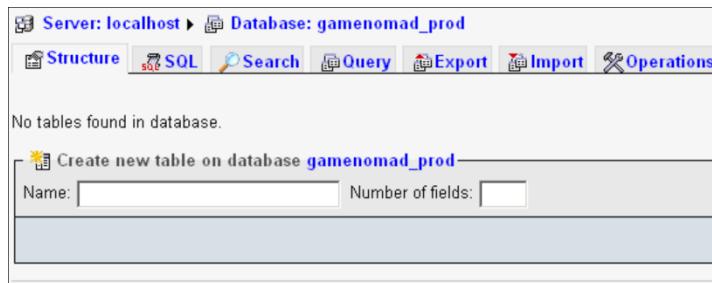


Figure 4-4. Creating a new table in phpMyAdmin

Next you'll be prompted to define the six fields. Figure 4-5 shows you the values and attributes I've assigned to each. Once you're done, click the Save button to complete the table creation process.

| Field | Type | Length/Values ¹ | Collation | Attributes | Null | Default ² | Extra | Comments |
|---------------------------|-----------|----------------------------|-----------|------------|----------|----------------------|-------------------|----------|
| <code>id</code> | INT | | | UNSIGNED | not null | | auto_increment | |
| <code>platform_id</code> | TINYINT | | | UNSIGNED | not null | | | |
| <code>title</code> | VARCHAR | 255 | | | not null | | | |
| <code>release_date</code> | DATE | | | | not null | | | |
| <code>price</code> | DECIMAL | 4,2 | | | not null | | | |
| <code>created_on</code> | TIMESTAMP | | | | not null | | CURRENT_TIMESTAMP | |

Table comments: Stores game information
Storage Engine: InnoDB
Collation:
Save Or Add [1] field(s) Go

Figure 4-5. Creating the games table's fields

Congratulations! You've just created your first MySQL table. Repeat the process to create the `platforms` table, which looks like this:

```
CREATE TABLE platforms (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL
);
```

Once you've added the platforms table, take some time to familiarize yourself with other phpMyAdmin features. It's a great MySQL management solution, and the more familiar you are with it, the better off you'll be.

Step #5. Connecting Your Website to MySQL

While phpMyAdmin presents a great solution for creating users, databases, tables, and performing other administrative functions, our ultimate goal is to create a series of custom interfaces for not only managing the game collection, but allowing our friends to view it. In this final step I'll introduce you to all that's involved in creating these interfaces.

Connecting to the MySQL Server

Before you can begin doing anything with the `domainname_prod` database, a successful connection must first be made. In fact, this connection is repeatedly made each time you attempt to perform a database action, whether it involves selection, updating, deleting, or inserting data. This is because MySQL's security model is twofold: first, it wants to make sure you're allowed to even connect to the database server, confirmed by examining the credentials you provide (namely the username, password, and address from which you are connecting). If it confirms you are indeed authorized to connect, it next determines whether you have adequate privileges to execute the action.

You might recall when creating the `domainname_prod` user that we chose the option "Create database with same name and grant all privileges". This resulted in `domainname_prod` being granted what essentially amounts to root-level privileges for the namesake database, giving you the power to perform all manner of administrative tasks. However, returning to that user creation screen, you'll see you can actually selectively determine exactly what the user can do. If you were creating a much more complex application which dealt with sensitive information such as consumer's private data, you might create several accounts: one account which possessed all privileges, used to manage the database's operation, another account which possessed just the select, insert, update, and delete privileges, used to manage the database's data, and finally, an account possessing just the select privilege, used to view the data over a secure web interface.

TIP. This introduction to MySQL's security model really only scratches the surface of what's possible. I cover this topic in great detail in Chapter 29 of "Beginning PHP and MySQL, Third Edition". You can buy this book at <http://www.easypHPwebsites.com/>.

Presuming you followed my advice when creating the `domainname_prod` user and granted the account all available privileges when working with the `domainname_prod` database, you will be able to both connect to the database and execute any possible action. But how is this connection even carried out? If you recall, in the first chapter we mentioned PHP's MySQLi extension. This extension provides PHP developers with the ability to talk to MySQL from inside PHP scripts, not only in terms of retrieving data for display within the browser, but also offering the ability to insert, update, and delete data among other tasks. A procedural and object-oriented interface are available, although we'll be discussing solely the latter for the remainder of this chapter as I believe it produces more succinct, better organized code.

Creating the Connection

To connect to MySQL, you'll create a new `mysqli` object, passing along the database's hostname, username, password, and database:

```
$db = new mysqli('localhost', 'domainname_prod', 'secret', 'gamenomad_prod');
```

It's always a good idea to verify the connection was successful before attempting to query the database. You can do so by making sure the `mysqli` object exists with a simple `if` statement:

```
$db = new mysqli('localhost', 'domainname_prod', 'secret', 'domainname_prod');

if (! $db) {
    echo 'My game collection is currently offline. Please come back later.';
    exit;
}
```

Although PHP will automatically close open database connections and return other resources to the server once a script completes execution, it's nonetheless good programming practice to explicitly close the database connection once your script is finished with it. To do so, call the `close()` method:

```
$db = new mysqli('localhost', 'domainname_prod', 'secret', 'domainname_prod');

if (! $db) {
    echo 'My game collection is currently offline. Please come back later.';
    exit;
}

// ...Perform various database-related tasks

// Close the connection
$db->close();
```

Creating a Configuration File

Because the MySQL connection must occur prior to executing any database-related tasks, you'll need to instantiate the `mysqli` class within each script. Because the ultimate goal is to eliminate redundancy, it wouldn't make sense to manually embed the connection parameters into each script. Doing so would make maintenance difficult should you later need to change the username or password.

The easy solution is to create a configuration file containing connection parameters and class instantiator, and then use the `require_once()` statement to include this code within each desired page. It's common practice to call these configuration files `config.inc.php`:

```
<?php
$dbHost = 'localhost';
$dbUsername = 'domainname_prod';
$dbPassword = 'secret';
$dbDatabase = 'domainname_prod';
$db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbDatabase);
if (! $db) {
    echo 'My game collection is currently offline. Please come back later.';
    exit;
}
?>
```

Place this file within your website's home directory and subsequently reference it within your scripts using:

```
require_once('config.inc.php');
```

No matter what you call the file, it is important you use the .php extension when naming it, because it will hinder any malicious attempts to view this file's contents. If somebody navigated to `http://www.example.com/config.inc.php` they would see a blank page since the PHP code found in the file would execute. However, if this file was named just config.inc, and the user navigated to `http://www.example.com/config.inc`, the file's contents would be displayed within the browser window!

Creating the Platform Manager

Before we can begin moving games into the `games` table, the `platforms` table entries must be logically added. So let's begin by creating the form and scripts used to add, view, and update this table. The first script, titled `platform-add.php`, handles the insertion of platforms into the `platforms` table. The form, shown in Figure 4-6, consists of nothing more than a text field and submit button.

Figure 4-6. The platform insertion form

However, before moving onto the code, let's take some time to understand how queries are created and sent from PHP to MySQL.

Introducing Queries

Even if you're not familiar with structured query language (SQL), the language used to talk to relational databases such as MySQL, you'll immediately understand the basic syntax. The following query inserts a new gaming platform into the `platforms` table:

```
INSERT INTO platforms VALUES(NULL, 'Xbox 360')
```

The `NULL` value tells MySQL to increment the primary key to one value higher than the last inserted row. Because this is our first row, this value will be 1. MySQL also supports an alternative `INSERT` syntax which looks like this:

```
INSERT INTO platforms SET title = 'Xbox 360'
```

In this case, MySQL recognizes the `id` column is an automatically incrementing primary key, meaning you don't have to explicitly set it, therefore MySQL will handle the increment by default. So how would this data be subsequently retrieved? Using the `SELECT` statement. To retrieve all columns of all rows, you would use this query:

```
SELECT * FROM platforms
```

To retrieve just the titles, you'd specify the column in the query:

```
SELECT title FROM platforms
```

To retrieve a specific row, you'd use the `WHERE` clause. For example, use the following query to retrieve the row having the primary key of 1:

```
SELECT * FROM platforms WHERE id = 1
```

Of course, if you don't specify a unique value in the `WHERE` clause, it's possible you'll retrieve several rows. Consider a situation where several games have a purchase price of \$59.99. You can use the following query to retrieve them:

```
SELECT * FROM games WHERE price = 59.99;
```

You can also perform wildcard searches. Suppose the platforms "Playstation 2" and "Playstation 3" were added to the table. You can use the `LIKE` clause to retrieve all rows with the word "Playstation" in the title:

```
SELECT * FROM platforms WHERE title LIKE "Playstation%"
```

The `%` following "Playstation" acts as a wildcard, telling MySQL to match anything following the term.

You now know how to insert and select rows. How do you update and delete them? Logically, using the `UPDATE` and `DELETE` queries. Suppose you misspelled "Nintendo Wii" as "Nintendo We" when adding it to the platforms table. You can fix the problem like this:

```
UPDATE platforms SET title = "Nintendo Wii" WHERE id = 3
```

In this particular case you could have also used the following query:

```
UPDATE platforms SET title = "Nintendo Wii" WHERE title = "Nintendo We"
```

Typically though you'll want to use the primary key when identifying a row for modification or deletion purposes. This ensures you'll modify the intended row, rather than multiple rows should they meet the specification of the `WHERE` clause. An even more unfortunate outcome can be expected if you neglect to include the `WHERE` clause altogether. If you executed the following query you'll change the title of *every* row in the platforms table:

```
UPDATE platforms SET title = "Nintendo Wii"
```

Deleting a row works in the same manner as updating it:

```
DELETE FROM platforms WHERE id = 3
```

Like `UPDATE`, make sure you use a unique identifier, otherwise multiple rows could be deleted. Neglecting to include the `WHERE` clause altogether will result in all rows being deleted!

Prepared Statements and PHP

You've by now learned how to connect to MySQL from within a PHP script, and are familiar with SQL syntax, so how are SQL queries sent from PHP to MySQL? Once connected to MySQL, sending a query to the database is as simple as passing it to the `query()` method, like so:

```
<?php

// Connect to MySQL
$db = new mysqli("localhost", "domainname_prod", "secret", "domainname_prod");

// Select some data
$result = $db->query("SELECT * FROM games ORDER BY title");

... Do something with the data

?>
```

Many of your queries will however be dynamic, based on some sort of user interaction or input. As an example, a user might wish to retrieve information about a specific game, in which case the above query would depend upon a `WHERE` clause:

```
$id = $_GET['id'];
$query = "SELECT * FROM games WHERE id = $id";

$result = $db->query($query);

echo "COUNT: {$result->num_rows}";
```

Logically, because the `id` is a primary key, only one row should be returned. And in a perfect world devoid of malcontents and troublemakers, this might be a perfectly suitable approach. However, in our particular world, this approach is a recipe for disaster, because it gives users the ability to modify the query in a variety of ways. For instance, in SQL there's a strange syntactical construct known as

`1=1` which will return all rows in the table. Here's an example:

```
SELECT * FROM games WHERE 1=1
```

With this in mind, what's to prevent somebody from modifying the incoming `id` parameter to read '`4`' OR `1=1` so the resulting query looks like this:

```
SELECT * FROM games WHERE id = '4' OR 1=1
```

This will result in all rows being retrieved from the database! In our particular above example, all that will be retrieved is a count of all rows in the database. However picture a scenario where you provided users with a search interface for finding friends. A malicious user might be able to use this approach (incidentally called an *SQL injection*) to obtain a list of all users in your database! SQL injections can do much more damage than allow for unwarranted access to data. They can be used to modify and even delete data, so you'd be well served to eliminate any such possibility of this sort of attack. Fortunately, there's an easy way to do so using a feature known as the *prepared statement*. Prepared statements greatly enhance the security of an SQL statement requiring user-provided data by properly accounting for any characters which could potentially modify the intended purpose of the query. Let's reconsider the above example:

```
// Prepare query, identifying one dynamic parameter (identified by the ?)
$query = $db->prepare("SELECT * FROM games WHERE id = ?");

// Tell MySQL the provided variable will be an integer
$query->bind_param('i', $_GET['id']);

// Execute the query
$query->execute();
```

In this case, the parameter we're passing to the query has been declared (or bound) to be an integer. Anything else found in that parameter will be properly accounted for to ensure it doesn't interfere with the query! Other commonly used type declarations include `s` for strings, and `d` for doubles (floats).

Don't worry if the concept of prepared statements hasn't completely resonated, as we'll be using them repeatedly in the remaining examples. By the conclusion of this chapter, you'll be entirely familiar with how to properly use them.

NOTE. Later in this chapter I'll introduce you to another type of prepared statement, used to retrieve data from the database. Using prepared statements in this manner results in a significant performance increase.

You now possess enough SQL knowledge to begin performing useful tasks. SQL statements can become decidedly more complex than what is presented here, however you'll have plenty of time to ease into the more complicated statements as the gaming site continues to expand. Next, let's move on to the code used to add new gaming platforms to the database.

Adding a New Platform

The script (`platform-add.php`) handles both the form display and the insertion of submitted form

contents. The code is presented next, followed by a breakdown of the relevant lines.

```
01 <?php
02
03     require_once('config.inc.php');
04     require_once 'gamenomad_validator.php';
05
06     // If the form has been submitted, process it
07     if (isset($_POST['submit'])) {
08
09         // Create array for storing error messages
10         $messages = array();
11
12         // Make sure a platform title was provided
13         if (isBlank($_POST['title'])) {
14             $messages[] = 'Please provide the platform title.';
15         }
16
17         // If no error messages, add platform to the database
18         if (count($messages) == 0) {
19
20             try {
21
22                 $stmt = $db->prepare("INSERT INTO platforms VALUES(NULL, ?)");
23
24                 // Bind the $_POST['title'] parameter
25                 $stmt->bind_param('s', $_POST['title']);
26
27                 // Execute the query
28                 $stmt->execute();
29
30             } catch (Exception $e) {
31                 $messages[] = 'Could not insert the platform.';
32             }
33
34         }
35
36         // Report results to user
37         if (count($messages) == 0) {
38
39             echo "<p>The platform {$_POST['title']} has been inserted.</p>";
40
41         } else {
42             echo "<ul>";
43             foreach ($messages AS $message) {
44                 echo '<li>{$message}</li>';
45             }
46             echo '</ul>';
47         }
48     }
```

```

49      }
50
51 ?>
52
53 <h1>Add a Gaming Platform</h1>
54 <form id="form" method="post" action="platform-add.php">
55   <p>
56     Platform name: <br />
57     <input name="title" type="text" size="35" value="" />
58   </p>
59   <p>
60     <input type="submit" name="submit" value="Add platform!" />
61   </p>
62 </form>

```

Although a longer script than what you've seen so far, there's actually little to it you haven't already encountered:

- Lines 03 and 04 pull the configuration file and validation script (introduced in the previous step) into the script.
- Line 07 determines whether the form (lines 52-60) has been submitted. If so, the submitted contents will be validated and filtered. We first verify (lines 13-15) the submitted `$_POST['title']` variable isn't blank.
- Presuming the validator doesn't determine the `$_POST['title']` value to be blank, lines 18-34 are responsible for inserting the platform into the platforms table.
- Line 22 creates what the prepared statement, specifying the query, and a placeholder (identified by the question mark) for the user-provided parameter. As we discussed earlier, creating the query in this way provides the most secure way possible to ensure malicious input isn't passed into the database. Although you'll presumably be the only person using this particular script, it's nonetheless sound programming practice to secure input at every opportunity.
- Line 25 binds the `$_POST['title']` input parameter to the placeholder, identifying the parameter as a string (defined by the "s").
- Finally, line 28 executes the query, inserting the platform into the database.

Go ahead and execute this script a few times, and check the `platforms` table contents from within phpMyAdmin. For purposes of this exercise I added the following platforms: Xbox 360, Xbox, Playstation 2, Playstation 3, Nintendo Wii, and Nintendo DS.

Listing the Platforms

To retrieve a list of all platforms found in the `platforms` table, you'll use a `SELECT` query coupled with a looping statement to retrieve each row in the result. The following example will retrieve all of the platforms in alphabetical order, outputting each to the browser:

```
01 <?php
02
03     require_once('config.inc.php');
04
05     $sql = 'SELECT id, title FROM platforms ORDER BY title';
06
07     $query = $db->prepare($sql);
08
09     $query->execute();
10
11     $query->store_result();
12
13     $query->bind_result($id, $title);
14
15     while($query->fetch()) {
16         echo "{$id}: {$title}<br />";
17     }
18
19     $query->free_result();
20     $query->close();
21
22 ?>
```

A breakdown follows:

- Line 05 specifies the query, which retrieves all platforms in alphabetical order.
- Line 07 and 09 prepares and executes the query, respectively. You'll notice this process is identical to that used when preparing queries dependent upon user input (as discussed earlier in this chapter).
- Line 11 stores the query results within server memory. This is known as *buffering the result set*, which is in contrast to working with an *unbuffered result set*. Throughout this chapter we'll use the former approach, which is fine when working with small and medium-sized query results, and comes with the bonus of a few additional features which you'll learn about later. See the below sidebar, "Buffered vs. Unbuffered Results" for more information about this topic.
- Line 13 assigns (or binds) two variables to the corresponding columns retrieved from the query (Line 05).
- Line 15 retrieves each row from the returned result set, assigning the column values in that row to the \$id and \$title variables, respectively. Line 16 in turn outputs the \$id and \$title variables to the browser.
- Lines 19 and 20 returns the memory used to store the result, and closes the statement. These calls are not strictly required, since PHP will automatically perform these tasks once the script completes execution. However if your script includes several prepared statements, you should explicitly complete these tasks at the appropriate locations.

Executing this script produces output similar to the following:

```
6: Nintendo DS
3: Nintendo Wii
5: Playstation 2
2: Playstation 3
4: Xbox
1: Xbox 360
```

Buffered vs. Unbuffered Queries

You may occasionally send a query to MySQL which returns a particularly large result set, insomuch that storing the results within server memory (buffering the results) could conceivably affect performance. Alternatively you may opt to instead work with an unbuffered result set, meaning the results will be maintained within MySQL and returned to the PHP script one row at a time. While the unbuffered approach will indeed improve performance when working with large result sets, it comes at the cost of being unable to determine matters such as how many results were returned (see the later section "Determining the Number of Rows Returned and Affected").

Updating a Platform Name

Chances are you won't need to change the names of any platforms once they're inserted into the database, however for purposes of illustrating how to modify existing information let's create a mechanism for doing so. For starters, modify Line 14 of the previous example to read:

```
14 echo "<a href='platform-edit.php?id={$id}'>{$title}<br />";
```

Clicking on one of the links will pass each platform's primary key to a script named `platform-edit.php`. Like the platform addition script, a form will be rendered but this time prepopulated with the contents of the row associated with the primary key. You'll then be able to change the contents and send them back to the database for modification purposes. The script (`platform-edit.php`) follows:

```
01 <?php
02
03     require_once('config.inc.php');
04     require_once('gamenomad_validator.php');
05
06     // If the form has been submitted, process it
07     if (isset($_POST['submit'])) {
08
09         // Retrieve the POSTed id and title
10         $id = $_POST['id'];
11         $title = $_POST['title'];
12
13         // Create array for storing error messages
14         $messages = array();
15
16         // Make sure a platform title was provided
17         if (isBlank($_POST['title'])) {
```

```
18         $messages[] = 'Please provide the platform title.';  
19     }  
20  
21     // If no error messages, add platform to the database  
22     if (count($messages) == 0) {  
23  
24         try {  
25  
26             // Create the SQL statement  
27             $sql = "UPDATE platforms SET title = ? WHERE id = ?";  
28  
29             // Prepare the statement  
30             $query = $db->prepare($sql);  
31  
32             // Bind the parameters  
33             $query->bind_param('si', $title, $id);  
34  
35             // Execute the query  
36             $query->execute();  
37  
38         } catch (Exception $e) {  
39             $messages[] = 'Could not update the platform.';  
40         }  
41     }  
42  
43     // Report results to user  
44     if (count($messages) == 0) {  
45         echo "<p>The platform {$title} has been updated.</p>";  
46     } else {  
47         echo '<ul>';  
48         foreach ($messages AS $message) {  
49             echo "<li>{$message}</li>";  
50         }  
51         echo '</ul>';  
52     }  
53 }  
54  
55 } else {  
56     $id = $_GET['id'];  
57 }  
58  
59 // Whether we've just modified a row or are accessing this  
60 // page for the first time, we want to populate the form  
61 $sql = "SELECT id, title FROM platforms WHERE id = ?";  
62  
63 $query = $db->prepare($sql);  
64  
65 $query->bind_param('i', $id);  
66
```

```
67     $query->execute();
68
69     $query->bind_result($id, $title);
70
71     $query->fetch();
72
73 ?>
74
75 <h1>Modify a Gaming Platform</h1>
76 <form id="form" method="post" action="platform-edit.php">
77 <input type="hidden" name="id" value="<?php echo $id; ?>" />
78 <p>
79     Platform title: <br />
80     <input name="title" type="text" size="35" value="<?= $title; ?>" />
81 </p>
82 <p>
83     <input type="submit" name="submit" value="Modify platform!" />
84 </p>
85 </form>
```

The breakdown follows:

- Like the platform insertion script, lines 07-54 will deal with posted form data, in this case retrieving the ID and title, ensuring the title isn't blank, and carrying out the update.
- If the platform modification script is being accessed for the first time, Line 56 will retrieve the platform ID from the URL. Lines 61-71 retrieve the relevant row from the platform table.
- Lines 75-85 display the form, including the platform ID in a hidden field, and the title in a text field.

You might consider creating a hybrid version of the platform list and update scripts which makes it easy for you to scan the list and update them with a few simple clicks and keystrokes. Figure 4-7 presents a screenshot of what this might look like.

Modify a Gaming Platform

Platform title:
Nintendo DS

Modify platform!

Click on a platform to modify its title

Nintendo DS
Nintendo Wii
Playstation 2
Playstation 3
Xbox
Xbox 360

Figure 4-7. Mockup of a streamlined platform update interface

Deleting a Platform

Because the net result of deleting a row is the destruction of potentially valuable data, typically users will be greeted with a confirmation dialog before the process is finalized. There seem to be countless variations of how the confirmation dialog is presented, however I've found a simple JavaScript confirmation window to be about as practical as any. Figure 4-8 displays a typical such window.



Figure 4-8. A JavaScript confirmation dialog window

Because the deletion process is nearly identical to the processes explained in the previously discussed insertion and modification scripts, I'll forego a complete code breakdown and instead concentrate solely upon the confirmation dialog. JavaScript, being an event-based language, is capable of responding to user actions such as the clicking of a link or submission of a form. Regarding the latter, to initiate some sort of JavaScript action when a link is clicked, all you need to do is embed a call to JavaScript's `onClick()` event handler:

```
while ($query->fetch()) {
    echo "<a href='platform-delete.php?id={$id}'"
        onclick=\"return confirm('Really delete the platform {$title}?');\">
        {$title}<br />";
}
```

When the user selects a platform for deletion, he'll be presented with the confirmation dialog. Clicking OK will cause the `platform-delete.php` script to be requested anew, but this time with the `id` parameter and appropriate value attached, thereby carrying out the deletion process. Clicking Cancel will merely close the confirmation dialog and return the user back to the page.

Dealing with Orphaned Records

There's a secondary matter you need to keep in mind when deleting records from certain tables: *orphaned records*. That is, records from other tables which relate to the row you're trying to delete. Suppose you removed the Nintendo Wii platform from the table, which happens to be the platform you've assigned to several games in your collection. With the platform deleted, the corresponding `id` found in the `games`' table's `platform_id` column is no longer tied to a row in the `platforms` table. What to do? There are actually several solutions to choose from, depending upon the particular situation:

- **Prevent platform deletion from proceeding:** If it is absolutely essential that the `games` table be tied to a particular row in the `platforms` table, you could always first check to determine whether any rows in the `games` table have been assigned the platform ID in question, and if so, prevent the user from deleting the platform.
- **Remove all games tied to that platform:** If the only conceivable reason for removing a

platform is due to the removal of it and its games (because you sold the console and games to a friend), you could perform an additional step which deletes all games tied to that platform.

- **Platform reassignment:** Although an unlikely choice in this particular situation, you might also reassign the platform ID for any game currently assigned the soon-to-be deleted platform. One approach would be to assign the value 0 (which will never be used in an automatically-incrementing primary key sequence) and within your programming logic display "No platform assigned" in any case where a game's platform is set to 0. Again, this approach doesn't make sense in the case of the game collection application, but in the case of a human resources application, it might make sense to set an employee's department to "Currently being reassigned" in the case of reorganization.

Creating the Game Manager

Now that you know how to create the various interfaces for managing the gaming platforms, chances are you already have a fairly clear idea of how the game interfaces should be constructed. In fact, given the similarities I'm going to focus solely on the game addition interface, as it requires you to enhance a simple web form by giving the user the ability to also assign a platform to the game being inserted. Of course, code for all of the required management actions is found in the code download.

Adding a New Game

As mentioned, the game addition interface is really no different from that used to add a platform, with one key difference: we need to offer a means for allowing the user to assign a platform to the game being inserted. This is generally done using a drop-down select box such as that shown in Figure 4-9.

Add a Game to Your Collection

Game title:

Platform:

Release date (YYYY-MM-DD):

Purchase Price:

Figure 4-9. Adding a new game to your collection

As you might imagine, retrieving the platform information is no different from what you've seen so far. Just create a prepared statement, and bind the results:

```
$sql = 'SELECT id, title FROM platforms ORDER BY title';

$platform = $db->prepare($sql);

$platform->execute();

$platform->bind_result($platformID, $platformTitle);
```

Next, within the form, add this code to create the select box:

```
<p>
    Platform:<br />
    <select name="platform_id">
        <option value="">Choose a platform:</option>
        <?php
            while ($platform->fetch()) {
                echo "<option value='{$platformID}'>{$platformTitle}</option>";
            }
        ?>
    </select>
</p>
```

Be sure to check out the code bundle to review the complete code for the game addition script (`game-add.php`), in addition to the modification (`game-edit.php`) and deletion (`game-delete.php`) scripts.

Determining the Number of Rows Returned and Affected

The bulk of our attention has been devoted to creating interfaces for adding, editing, deleting, and listing games and platforms. But the MySQLi extension offers quite a few other features which can enhance your application's usability, such as determining the number of rows returned from a SELECT query, and the number of rows affected from an INSERT, UPDATE, or DELETE operation. In this section I'll show you how to do both.

Counting the Number of Returned Rows

It's often useful to give users an overview of the number of records in a table, such as the number of games in your collection. To do so using the MySQLi extension, call the `num_rows` property:

```
$sql = 'SELECT id FROM platforms ORDER BY title';
$platform = $db->prepare($sql);
$platform->execute();
$platform->store_result();

// Retrieve the number of returned rows
$rows = $platform->num_rows;

echo "There are {$rows} platforms in the database.;"
```

While this code is self-explanatory, there are two matters worth noting. First, you must call the `store_result()` method prior to retrieving the number of returned rows, because otherwise MySQL will not have returned all of the rows back to the server (remember the buffered vs. unbuffered queries discussion?), meaning PHP will be unable to discern how many rows are available. Second, notice how `num_rows` is an object property rather than a method. Attempting to call this as a method (concluding with parentheses) will result in an error.

Counting the Number of Affected Rows

Particularly when creating interfaces capable of inserting, modifying, or deleting numerous rows in a table, it's useful to determine the number of rows which have been affected as a result of the operation. To make this determination, refer to the `affected_rows` property:

```
$sql = 'DELETE FROM platforms WHERE 1=1';

$platform = $db->prepare($sql);

$platform->execute();

$platform->store_result();

// Retrieve the number of affected rows
$rows = $platform->affected_rows;

echo "{$rows} platforms have been deleted.";
```

Step #6. Restricting Access

You'll want to restrict access to the game collection management scripts, otherwise anybody could potentially access and modify or even destroy your data. Numerous strategies exist for limiting access, although the easiest involves relying on the authentication mechanism built directly into the same communication protocol (known as HTTP) used to power the Web. This mechanism requires users to provide a username and password via a pop-up authentication window (Figure 4-10). If the username and password matches the predefined values, the user is granted access. Otherwise, access is denied.

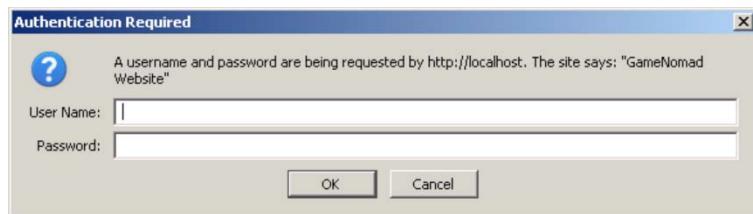


Figure 4-10. The default authentication window

Adding this sort of authentication feature to your site can be done in just a few lines of code, presented below:

```
<?php
    if (($_SERVER['PHP_AUTH_USER'] != 'gamenomad') ||
        ($_SERVER['PHP_AUTH_PW'] != 'secret')) {
        header('WWW-Authenticate: Basic Realm="GameNomad Website"');
        header('HTTP/1.0 401 Unauthorized');
        echo 'You must provide the proper credentials!';
        exit;
}
?>
```

As you can see, the username and password (`gamenomad` and `secret`, respectively) are embedded within the code, and are compared to the `$_SERVER['PHP_AUTH_USER']` and `$_SERVER['PHP_AUTH_PW']` variables, respectively. If the username and password matches, the if-conditional body is bypassed altogether. Otherwise, the authentication window is displayed. If at any point the user presses the Cancel button, the error message found in the if-conditional body is displayed.

Keep in mind code involving calls to the `header()` function must execute before anything is output to the browser. Neglecting to follow this important rule will result in an error.

Of course, there's nothing to prevent you from revising this code to determine whether the provided username and password exist in a database table, opening up the possibility for a multi-user authentication solution.

If you're looking for a more robust authentication solution which can plug into several preexisting authentication implementations such as LDAP, Samba, and Kerberos, check out the PEAR Auth package.

Conclusion

In just three chapters, you've gone from a Web development neophyte to one capable of building a fairly robust, secure Web application capable of managing your gaming collection. I think congratulations are in order!

But in light of this sudden success, your aspirations are growing. Suddenly, you've taken a liking to suspenders and greasing your hair back in the spirit of Gordon Gecko. You stare in the bathroom mirror each morning screaming, "The world is mine!". You initially plotted to take over the universe but decided against it because frankly, it just isn't big enough. So instead, you're going to create the best darned gaming website ever. Read on to learn how.

CHAPTER 5

Introducing the Zend Framework

The steps you've taken in the first four chapters have no doubt been tremendous. In a short time you've gone from the hassle of updating and e-mailing a spreadsheet around to your friends to publishing a database-driven website which encourages users to not only peruse your game collection, but also participate by offering you feedback and pointers to other games through a simple web form.

But in making this transition you've been exposed to a number of challenges faced by all Web developers, including:

- Writing custom validation code for filtering user-provided data such as names, phone numbers, and e-mail addresses
- Managing page layouts within multiple files, namely `header.php` and `footer.php`.
- Figuring out how to manage configuration information such as database connection parameters and mail server addresses.
- Mixing substantial amounts of PHP and HTML together which tends to make it difficult for you to easily update the layout, and almost impossible for you to hire a Web designer to improve the website's look-and-feel for fear of mistakenly broken code.

And this is to say nothing of the issues you'll undoubtedly encounter as the website grows, such as:

- Implementing a multi-user authentication system which plays a key role in granting friends the ability to manage their own game collections
- Expanding the Web site's capabilities by integrating third-party Web services such as YouTube, Flickr, and Amazon
- Going overseas with localized versions of the website tailored to the user's native language, currency, and other customs.

What if I were to tell you there's a solution which will greatly reduce, if not altogether eliminate these issues, along the way making you a faster, more efficient, less error-prone Web developer? This solution is called a *Web framework*, and much of the rest of this book is devoted to showing you how to use a best-of-class PHP framework known as the Zend Framework.

Chapter Steps

The goals of this chapter are accomplished in nine steps:

- **Step #1. What is a Web Framework?**: The first four chapters have gone so well that you might wonder why we're switching strategies at this point in the book. This step explains how a framework can greatly reduce the time and effort you'd otherwise spend dealing with these issues when building a website from scratch.

- **Step #2. Introducing the Zend Framework:** With so many Web frameworks available, why should you choose the Zend Framework? In this step I'll highlight some of the features that make the Zend Framework such an attractive solution.
- **Step #3. Installing the Zend Framework:** Installing the Zend Framework is actually quite simple, however configuration can be a bit of a chore if you're not familiar with the process. In this step I'll show you how to create the required structure and create the files Zend requires to properly function.
- **Step #4. Testing Your Installation:** The easiest way to test a new Zend Framework installation is to start building your application! In this step we'll create a few starter scripts to ensure everything is functioning as expected, in the process laying the groundwork for the GameNomad application.
- **Step #5. Creating the Error Controller:** Being able to properly address errors such as non-existent URLs is a crucial part of sound website design. In this step you'll learn how to create an error controller, which can not only provide users with a friendly error message, but also e-mail you or perform other diagnostics should an error occur.
- **Step #6. Creating the Website Layout:** Web Frameworks like Zend's greatly reduce and even eliminate any design redundancy by breaking the website into reusable parts. Of these parts, the largest is the *layout*, and in this step I'll show you how to create it. You'll also learn how to take advantage of other Zend Framework features such as the *view helper* and *partial* to more effectively manage your site.
- **Step #7. Creating a Configuration File:** Maintaining a site's configuration data in a single location makes it trivial for you to update a MySQL password or mail server address, not to mention gives you peace of mind knowing that the entire site's behavior will reflect a single configuration change. Furthermore, a centralized solution can greatly reduce the time involved to update configuration settings when migrating your site from the development to live stage. The Zend Framework was built with a feature which fulfills all of these conveniences, and by the end of this step you'll wonder how you ever lived without it.
- **Step #8. The init() Method:** You'll often want to carry out a few tasks before any method in a particular controller is executed. Rather than repetitively include this code at the top of each method, you can include it within the controller's *init()* method, which will execute prior to any method found in the controller. In this step I'll show you how this is done.
- **Step #9. Creating Action Helpers:** In the final step of this action-packed chapter, it seems fitting to conclude with the creation of an action helper, which can be used to share initialization variables and other tasks across controllers.

Step #1. What Is a Web Framework?

While I could come up with my own definition of a Web framework, it would likely not improve upon the already fantastic definition published on the Wikipedia website (http://en.wikipedia.org/wiki/Web_application_framework):

A web application framework is a software framework that is designed to support the development of dynamic websites, Web applications and Web services. The framework aims to alleviate the overhead associated with common activities used in Web development. For example, many frameworks provide libraries for database access, templating frameworks and session management, and often promote code reuse.

That's quite a mouthful. Let's break this definition down into several segments and dissect their meaning.

...designed to support the development of dynamic Websites

Dynamic websites, like any software application, are ultimately composed of three components: the data, the presentation, and the logic. In the world of computer science, these components are referred to as the *model*, *view*, and *controller*, respectively. Together, these three components work in unison to power the website. What you've seen so far in the book is a mish-mash of the three, which is acceptable for small projects but becomes increasingly difficult to manage as the project grows in size and complexity. The potential for problems due to unchecked intermingling of these components becomes obvious after some consideration:

- **Technology Shifts:** MySQL has long been my preferred database solution, and I don't expect that sentiment to change anytime soon. However, if another more attractive database comes along one day, it would be foolhardy to not eventually make the switch. But having implemented the first version of GameNomad with little regard to tier separation, we'd be forced to rewrite every MySQL call and possibly much of the SQL to conform to the syntax supported by the new database, in the process potentially introducing coding errors and breaking HTML output due to the need to touch nearly every script comprising the application.
- **Presentation Maintainability and Flexibility:** Suppose you decide your graphical design skills have reached their natural limit, and accordingly you want to hire a graphic designer to redesign the site. Unfortunately, this graphic designer knows little PHP, and proceeds to remove all of those weird lines of text before uploading the redesigned website, causing you several hours of downtime while you recover the site from a backup. Furthering your problems, suppose your website eventually becomes so popular that you decide to launch a version optimized for handheld devices. This is certainly a feature which would excite users and potentially attract new ones, however because the logic and presentation are so intertwined it's impossible to simply create a set of handheld device-specific interfaces and plug them into the existing code. Instead, you're forced to create and subsequently maintain an entirely new application!
- **Code Evolution:** Over time it's only natural your perspective on certain approaches to building websites will evolve. For instance, suppose you may initially choose to implement the OpenID solution in order to authenticate users, but later decide to host the authentication mechanism and data autonomously. Yet because the authentication-specific code is sprinkled throughout the entire website, you're again forced to spend a considerable amount of time updating this code to reflect the new authentication approach.

So what's the solution to avoiding these very real problems? The solution is to separate these com-

ponents into distinct parts (also known as *tiers*), and write code which *loosely couples* these tiers together. By removing the interdependencies otherwise incurred when the three tiers are intertwined, you create a considerably more manageable and scalable site. A framework (or more specifically, an *MVC framework*, which is the type we'll be focusing upon for much of the remainder of this book), provides you with the foundation for separating these tiers from the very outset of development!

Let's take a moment to understand precisely what role each tier serves within the MVC architecture.

The Model

Simply put, you can snap up the coolest domain name, and hire the world's most talented graphic designer, but without content, your project is essentially dead-in-the-water. In the case of GameNomad that data is largely user- and game-related. To manage this data, you'll logically need to spend some time thinking about and designing the database structure. But there's much more to effectively managing an application's data than simply creating a well-defined schema. You'll need to take things such as session state, data validation, and other data-related constraints into account. Further, as your schema evolves over time, it would be ideal to minimize the number of code modifications you'll need to make in order to update the application to reflect these changes. *The model* tier takes these sorts of challenges into account, acting as the conduit for all data-related tasks, and along the way greatly reducing the complexity of your code.

The View

The second tier comprising the MVC architecture is *the view*. The view is responsible for managing the application's interface; that is all of the forms, buttons, logos, graphics, and other HTML elements found throughout the website. Of course, a view isn't restricted to HTML. Views might also be used to generate RSS for a user's aggregator, or a Flash-based presentation. By separating the interface from the application's logic, we can greatly reduce the opportunity for mishaps occurring when the graphic designer decides to tweak the site logo or a table layout, while simultaneously making it much easier for the developer to maintain the code's logical underpinnings without getting lost in a mess of HTML and other graphical assets.

Try as one may, a typical view will almost certainly not be devoid of PHP code. You'll still use simple logic such as looping mechanisms and if statements to carry out various tasks, however the bulk of the complex logic will be hosted within the third and final tier: the controller.

The Controller

The third part of this triumvirate is the *controller*. The controller is responsible for processing events, whether they are initiated by the user or some other actor, such as a system process. You can think of the controller like a librarian, doling out information based on a patron's request, be it the date of Napoleon's birth, the location of the library's collection of books on postmodern art, or directions to the library. To do this, the librarian reacts to the patron's input (a question), and forms a response by looking to the model (in this case, either her brain, the card catalog, or consultation of her colleague). In answering these questions, the librarian may dole out answers in a variety of formats; talking to the patron in person, responding to an e-mail, or posting to a community forum.

A framework controller operates in the same manner as a librarian, accepting incoming requests, acquiring the necessary resources to respond to that request, and returning the response in an appropriate format back to the requesting party. As you've probably already deduced, the controller typically responds to these requests by invoking some level of logic and interacting with the model to produce a response (the view) which is formatted and returned to the requesting party. This process is commonly referred to as an *action*, and they're generally referred to as verbs, for example *add game*, *find friend*, or *contact administrator*.

MVC in Action

So how do these three components work in unison to power a website? Picture a police officer standing in the middle of an intersection directing traffic. This officer (the controller) behaves according to the set of requests being directed towards him. He employs logic based on his understanding and experience with area traffic laws to formulate a response appropriate to the actions of the oncoming drivers, which might include the blow of a whistle, hand signal, or cast a scornful glare at the teenager sending a text message while driving with his elbows. Breaking this down into the three tiers:

- **The Model:** The officer's knowledge of the traffic laws, such as whether one oncoming driver has the right of way over another.
- **The View:** The officer's response to incoming driver requests, for instance a whistle blow or hand signal.
- **The Controller:** The officer's use of eyes and ears to identify and process an incoming process, followed by an internal signal to use an appropriate response after consultation of the model (the information stored in his brain).

While this is perhaps an overly abstract representation of the MVC architecture in action, it should nonetheless help you to visualize the purpose of each tier.

...alleviates overhead associated with common activities used in Web development

Web frameworks were borne from the understanding that all dynamic websites, no matter their purpose, share a common set of characteristics. For instance, chances are all will need to validate user input, communicate with a data source such as a relational database, and rely upon various configuration settings such as mail server addresses and site-specific values such as developer keys for talking to Amazon Web Services. Yet although these characteristics are as common to a website as the typical HTML form, over the years they've been implemented in countless ways. A framework removes many of these design decisions by preaching an approach known as *convention over configuration*.

Frameworks Eliminate the Hassle of Data Validation Strategies

This notion of convention over configuration eliminates variations in implementation from one project to the next by defining and implementing the approach for you. One shining example of the advantages of such an approach is data validation. In past chapters you learned how to validate user input by creating a validation library. While beneficial, it's likely you found this task rather tedious

and error-prone, with no end in sight as the complexity of data increases alongside the website.

Furthermore, while generally you might continue to use and rely upon your custom validation library, in the midst of occasional fits of laziness you might quickly hardcode a validation mechanism into a new website, only to later forget you did so, and be forced to create it again at some later time. Recognizing these difficulties, most web frameworks come packaged with not only a well-defined approach for validating user input, but in some cases will actually automatically perform the validation for you! Further, many come with a set of standard validators for examining e-mail addresses, phone numbers, and credit cards, altogether eliminating the need for you to write these complex bits of logic.

Frameworks Remove the Uncertainty of Website Configuration Management

Another obvious advantage brought about by convention over configuration is the website's configuration data itself. In previous chapters we used a file named `config.inc.php` to store database connection settings. While this is a common approach and naming convention, it's hardly the only one I've seen (or used!) over the years, resulting in not only the internal confusion wrought by having to maintain multiple configuration solutions as your collection of websites and/or client projects grow, but also in confusion brought about should another developer join the team or take over a project. Ask yourself, is it really worth your time to devise yet another configuration solution every time a new project arises? The answer is no, a sentiment shared by Web framework users all over the world, who appreciate the standardized configuration solution offered by their chosen framework (you'll learn about one such solution later in this chapter).

In summary, I believe this alleviation of uncertainty is one of the strongest points to consider when weighing the advantages of a framework against creating a website from scratch. Ask yourself, should you be spending valuable time doing the middling tasks which will invariably come up every time you set out to create a new website, or should you simply let a framework do the thinking for you in those regards while you concentrate on building the most compelling website possible? I think you know the answer.

...provides libraries for database access, templating frameworks, and session management

In many ways this feature is representative of a concrete conclusion to the previously discussed convention over configuration. What I mean by this is that not only does a Web framework provide you with a prescribed approach to solving a problem, but it will also provide you with the tools (libraries) for solving it. By taking advantage of these robust, tested code libraries, you can focus upon using them to build a compelling website rather than have to worry about implementing a solution according to a framework's recommended approach. Let's look at just two examples of how a solution such as the Zend Framework reduces the amount of code you have to write and maintain.

Database Interaction

In previous chapters you learned how to integrate a MySQL database into your website. At the time this was clearly a huge step away from the clunky spreadsheet and towards an effective data management solution. But if you're like most developers (this one included), the process of jumping from one

language, in our case PHP, to another, namely SQL, is a rather inefficient affair. What if you could write everything in PHP? For instance, in the previous chapter an example included this sequence of statements:

```
$sql = "SELECT id, platform_id, title, price FROM games ORDER BY title";
$query = $db->prepare($sql);
$query->execute();
$query->store_result();
$query->bind_result($id, $platform_id, $title, $price);
```

Using the Zend Framework's Zend_Db component, you can achieve an identical result while foregoing altogether the need to write SQL statements:

```
$game = new Game();
$query = $game->select();
$query->from(array('id', 'platform_id', 'title', 'price'));
$query->order('title');
$result = $game->fetchAll($query);
```

This programmatic approach to interacting with the database has an additional convenience of giving you the ability to move your website from one database to another with minimum need to rewrite your code. Because most frameworks abstract the database interaction process, you're free to switch your website from one supported database to another almost at a whim. Try doing this with the code we've written so far!

User Authentication

Another more impressive example of what a framework has to offer in terms of readily available code is the Zend Framework's Zend_Auth component (discussed in Chapter 8). Whether your website consists of just a small community of friends or is an enormous project with international reach, chances are you'll require a means for uniquely identify each user who interacts with your site at some level (typically done with user accounts). Zend_Auth not only provides you with a standardized solution for authenticating users, but also provides you with interfaces to multiple authentication storage backends, such as a relational database, LDAP, and OpenID. With all of these solutions at your disposal, all you have to do is integrate it into your project and move to the next task!

Further, while each backend depends upon custom options for configuration, the authentication process is identical for all solutions, meaning that even when switching authentication solutions you'll only have to deal with configuration-related matters.

Web Services

Websites these days are rarely self-contained, meaning their features are often hybridized constructs built from the infrastructures and data of several sites and services. GameNomad is a perfect example of this, relying upon the Amazon Associates Web Service for gaming data, and the Facebook Platform for social networking, among others. Without this ability to integrate with other online services such as these, GameNomad would be a far less compelling project.

While many of these services are built using standardized protocols and data formats, there's no doubt that writing the code capable of talking to them is a time-consuming and difficult process. Recognizing this, many frameworks provide libraries which do the heavy lifting for you, giving you the tools capable of connecting to and communicating with these third-party services. For its part, the Zend Framework offers Zend_Gdata, for interacting with Google services such as Book Search, Google Calendar, Google Spreadsheets, and YouTube. You'll also find Zend_Service_Twitter, for talking to the Twitter service (<http://www.twitter.com/>), Zend_Service_Amazon, for retrieving data from Amazon's product database through its Web Services API (<http://aws.amazon.com/>), and Zend_Service_Flickr, for creating interesting photo-based websites using Flickr (<http://www.flickr.com/>), one of the world's largest photo sharing services.

Step #2. Introducing the Zend Framework

Cofounded in 1999 by two of the PHP project's most prominent developers, Zeev Suraski and Andi Gutmans, Zend Technologies (<http://www zend com>) counts some of the largest organizations in the world among its client base. Accordingly, as the world's leading PHP products and services company, when in 2005 they decided to launch the Zend Framework project, the community took notice. Almost two years later, version 1.0 was released, already boasting millions of downloads and hundreds of contributors. Since then, the project has continued to gather steam, with the latest numbers indicating over 10 million downloads and more than 500 contributors since the project's announcement (Source: Zend Monthly Newsletter - December, 2008).

Given the company's lofty stature, it might not come as a surprise that the framework has been the focus of so much attention. Yet attaining such a level of success is more difficult than one might think, given the considerable competition from not only competing frameworks hailing from other languages, but also frameworks developed by fellow members of the PHP community. So what characteristics separate the Zend Framework from competing solutions? While the answer to this question is perhaps a subjective one, it's fair to say we could at least speculate as to what are the most attractive features:

- **Accountability:** While companies around the globe are rushing to embrace the opportunities made available by open source software, many prefer the "one neck to wring" when it comes to software support. The availability of support packages, training programs, and a certification program gives many companies peace of mind they'll at least have a support option if necessary.
- **Community:** While the namesake Zend Framework project is fostered by Zend Technologies, the company has wisely applied an open source license (BSD) to the project, and opened up the development process to the PHP community. This strong community involvement gives the Zend Framework project the best of both worlds, namely a strong corporate backing coupled with from hundreds of volunteers participating in everything from code contributions to forum support.
- **Flexibility:** Like PHP, although the Zend Framework's capabilities are vast, you can start building powerful websites even by understanding just a fraction of what's available to you. As your knowledge and needs grow, you can begin looking into the almost 70 components made available through the framework.

- **Capability:** While the Zend Framework is a perfectly useable MVC framework, the developers realize that the typical user's needs will quickly expand far beyond fundamental features. To help with these complex tasks, the Zend Framework includes features such as database support, RSS feed creation, Ajax integration, and communication with disparate services such as those offered by Amazon.com, Yahoo, Google, Flickr, and SlideShare.

Convinced the Zend Framework is for you? I thought so! Let's get our hands dirty by installing and configuring the Zend Framework, in the process building the skeleton for our revamped game collection site. Before doing so however, it's worth taking some time to understand more about how the Zend Framework operates from a conceptual level, along the way defining some of the key concepts and files that you'll repeatedly encounter when working with the framework.

NOTE. Admittedly the PHP community was rather late to the party when it comes to Web frameworks. Perhaps spurred on by the unexpected success of the Rails framework (<http://www.rubyonrails.org/>), several like-minded PHP developers came together to produce a number of framework solutions which have subsequently grown in stature and popularity. In addition to learning more about the Zend Framework in this chapter, I suggest spending some time checking out other popular PHP-oriented solutions such as CakePHP (<http://www.cakephp.org/>), symfony (<http://www.symfony-project.org/>), and CodeIgniter (<http://www.codeigniter.com/>).

How the Zend Framework Works

The Zend Framework operates in a manner which isn't entirely intuitive to newcomers. As mentioned it's based around the MVC development paradigm, which is an attempt to neatly separate the website's model, logic, and presentation into three distinct parts. Further though, the framework also comes packaged with some default conveniences which make the development process significantly easier, and dare I say fun. For instance, by default it offers what are commonly referred to as "pretty URLs", which turn URLs from unintelligible strings such as <http://www.gamenomad.com/games.php?game=12> into <http://www.gamenomad.com/games/show/mariokart>. By design you can also manage your code within well-organized classes which are representative of the models and controllers you were introduced to earlier in this chapter. The Zend Framework will respond in accordance with the incoming request, invoking the appropriate controller (and accordingly, the appropriate action), and working in conjunction with the models, will assemble the response into the corresponding view to produce the desired response.

But how does the framework know which controller action to invoke? It knows because unlike the approach we've used so far in which the user specifically requests (by name) the script he'd like to execute, the Zend Framework funnels *all* requests through what's called the *front controller*. This front controller examines the URL layout, and calls upon certain resources as defined by that URL. Consider for instance the following URL:

<http://www.gamenomad.com/games/show/mariokart>

Given this URL request, the Zend Framework can be configured to retrieve the *games controller*, and execute the *show action* within that controller. The show action will be passed the *mariokart* parameter, which the show action can use to consult the database (by way of some corresponding model), and retrieve further details about that particular game.

Rewriting Requests

But if the Web server's default behavior is to map a request directly to a resource, how does it know to instead pass all requests to the Zend Framework's front controller? It does this using what's known as *rewrite rules*. The Apache web server has long offered a powerful feature by way of a URL rewriting module known as mod_rewrite, which is enabled by default. We can take advantage of mod_rewrite's powerful syntax within an `.htaccess` file. Because a site's `.htaccess` file will be examined before any other action occurs, we can specify that any incoming URL request be routed to a specific script (the by now infamous front controller). That URL will then be passed to the front controller as a parameter, parsed, giving the front controller the means for initiating a response. Think of the described process as a workflow which looks that shown in Figure 5-1.

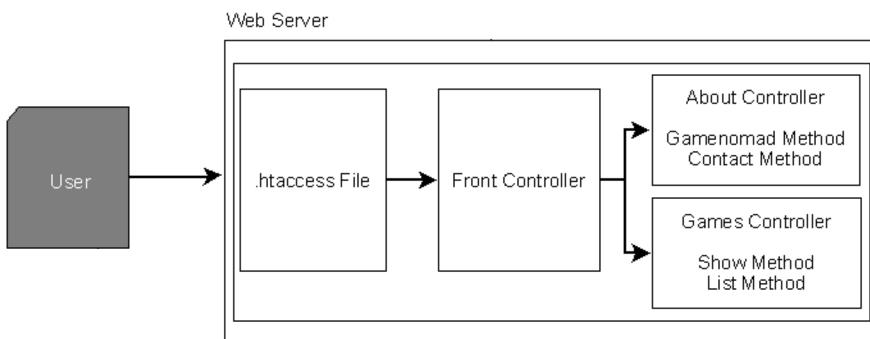


Figure 5-1. Routing requests to the appropriate Zend Framework controller and action

Forming the Response

You understand conceptually how the Zend Framework accepts and routes a request. But how is the request carried out, and the response assembled and returned to the requesting user? Each controller action is accompanied by a corresponding *view template*, which is returned to the user when that action is requested, along with any dynamic data which might be inserted into the template. For instance, a request to provide a list of all games in your collection (`http://www.gamenomad.com/games/list`) might result in the Games controller's List action being executed. This action would retrieve an alphabetical list of games found in your collection from the MySQL database, and pass that list as an array to the List action's view. A PHP looping construct in the view would iterate through the array, thereby embedding the list alongside any surrounding text (which could be statically embedded into the view, such as the heading "My Video Game Collection").

Keep in mind the content found in that action's view should be specific to the user's request, and not contain for instance the website's template. This is because the Zend Framework offers a special convenience known as the *layout*, which contains the website header and footer! A special call within this layout tells the framework where to insert the action's view data, thereby producing a page complete with the page header, footer, and relevant requested data. Figure 5-2 depicts this process.

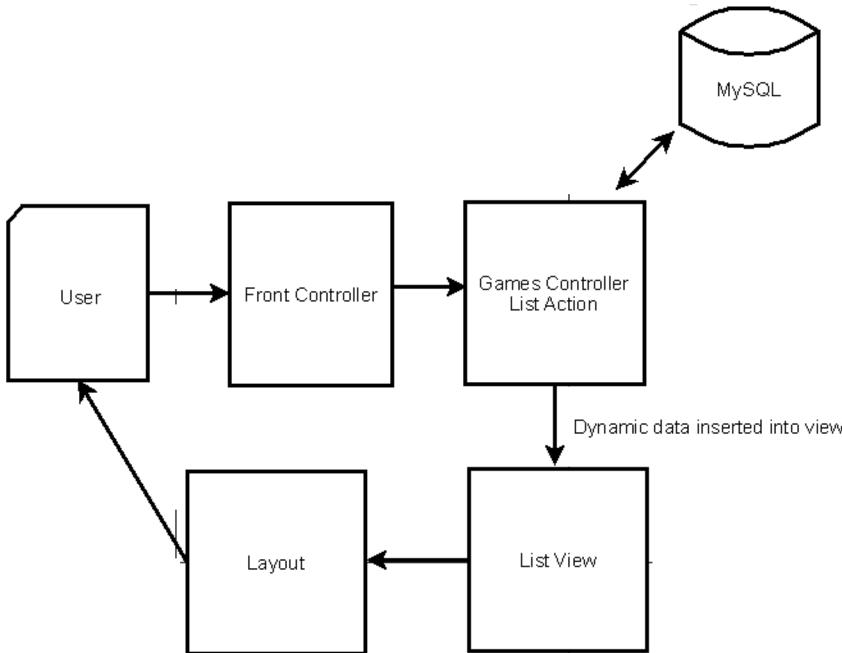


Figure 5-2. Processing a user request

Of course, what I've shown you so far is accurate, if an oversimplification of how the Zend Framework accepts requests and processes responses. It's far more powerful than what's been illustrated here, for instance it also allows you to do things such as process a specified bit of code prior to executing any action falling under a certain controller. You can also create templates (known as *partials*) which can subsequently be embedded into other templates should you find these partials need to be used in multiple locations, thereby reducing the total amount of code you'll need to maintain. Furthermore, if you would rather a request to `http://www.gamenomad.com/games/` was routed to the `Collection` controller's `List` action (rather than presumably the `Games` controller's default action (which is incidentally the action named `Index`), you can override this behavior at will.

If you find any of this confusing, don't spend too much time dwelling on it as all will become much more obvious as we progress through the chapter! Speaking of which, let's move onto the crux of this material by installing and configuring the framework.

Step #3. Installing the Zend Framework

The Zend Framework requires very few tasks to install, primarily because almost all of the files required to use it are located within a self-contained directory which you'll simply place within an appropriate location on your web server. All you really need to do is make sure you're running PHP 5.1.4 or newer, which is a certainty if you installed PHP based on the instructions found in this book. The Zend Framework is supported by all of the standard operating systems, so you'll be able to develop on Windows and move the application to Linux with minimum overhead.

VIDEO. Building Your First Zend Framework Website

There's no question even beginning web developers should look immediately to a web framework such as the Zend Framework for building even the simplest of websites. Many developers however quickly grow frustrated by the initial learning curve, not only struggling with applying the MVC concept, but also with configuration issues. This video seeks to dispel all such vexations, helping you to master these issues in just minutes. Watch it at <http://www.easyphpwebsites.com/zfw/videos>.

Although several installation options exist, the easiest is to simply download the latest release from <http://framework.zend.com/download/latest>. I recommend downloading the full version as it contains everything you could possibly need to follow along with the rest of this book, and a whole lot more. To install the framework, follow these steps:

1. Download the latest version to the computer you're using for development purposes, and unzip it.
2. Create the directory you intend on using as your website's home directory. In previous chapters we used `C:\apache\htdocs\gamenomad` (Windows) or `/usr/local/apache/htdocs/gamenomad` (Linux). Feel free to continue using this directory for the remainder of the book, although you should consider backing up any files you've created so far in case you want to refer to them at some future time.
3. Inside this home directory, create another directory named `application`. This directory will house all of the custom scripts we create when building the website. Within the `application` directory, create a directory named `controllers`, another named `views`, and within the `views` directory, create a directory named `scripts`.
4. Move the library directory found in the Zend Framework download into the home directory. For instance, on Windows the library directory should be placed along this path: `C:\apache\htdocs\gamenomad\library`.
5. Create a directory named `public` which resides within the `application` directory. This directory is used to store images, CSS files, JavaScript files, and anything else you'd like to make directly available through the browser. Within this `public` directory, create three directories named `css`, `images` and `javascript`.

TIP. If you're developing just a single website with the Zend Framework, then the above instructions are going to work just fine. However if you plan on developing multiple sites, then it doesn't make sense to have multiple copies of the library directory lying around. Instead, place a single copy of the directory within a central location (for instance `C:\php\includes\zend\library`) and then add that directory to PHP's `include_path` configuration directive. If you don't have access to your `php.ini` file, don't worry as later in this section I'll show you how to add that directory to your `include_path` from within Zend's `bootstrap.php` file.

At this point, your website directory structure should look like this:

```
/htdocs
  /gamenomad
    /application
      /controllers
      /views/
        /scripts
    /library
      [contents omitted for space reasons]
  /public
    /css
    /images
    /javascript
```

Believe it or not, the Zend Framework is installed and ready to use. But before you can begin using it to build a website, several configuration-related steps remain.

Configuring the Zend Framework

The Zend Framework is now installed, however a few remaining steps remain before you can begin taking advantage of it. Namely, we need to create three files: `.htaccess`, `bootstrap.php`, and `index.php`. Each plays a crucial role in the Zend Framework's operation, as will be explained next.

Creating the `.htaccess` File

The `.htaccess` file is Apache's directory-specific configuration file, used for tasks such as password-protection or redirection. It is this file which is responsible primarily for redirecting all user requests to the Front Controller. This is accomplished using an Apache module known as `mod_rewrite`, which is capable of reviewing incoming URL requests and rewriting or redirecting those requests in practically any conceivable manner. It is precisely this file which allows you to create well-organized URL structures (for instance `http://www.gamenomad.com/games/show/mariokart`) for your website without having to actually create and maintain a corresponding directory structure.

Create a file named `.htaccess` (the preceding period is important!), and place the contents found in Listing 5-1 into it. Save this file within the `public` directory. Alternatively, you can copy the `.htaccess` file found in the Chapter 5 code download directory to this location.

Listing 5-1. The Zend Framework's `.htaccess` File

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
RewriteRule ^.*$ /index.php [NC,L]
```

As you can see, `mod_rewrite`'s considerable power comes at the cost of some pretty odd-looking syntax, a sentiment shared by even experienced server administrators. In fact, the first page of the `mod_`

rewrite documentation comically includes the following quote: *"Despite tons of examples and docs, mod_rewrite is voodoo. Damned cool voodoo, but still voodoo. - Brian Moore"* That said, because in all likelihood you'll never touch this file again, I'm not even going to bother explaining it as doing so could come at the cost of several pages, and instead suggest you just set it and forget it.

Create the index.php File

As was noted in the .htaccess file, it's also the entry point for all incoming requests. In all, this file is responsible for completing four tasks: setting PHP's `include_path` directive to include the Zend Framework's `library` directory, making it possible for Zend Framework's classes to be called within scripts without having to first include them using the `require()` statement, referencing the `bootstrap.php` file, and finally jumpstarting Zend's front controller, which handles the request routing. Copy the `index.php` file shown in Listing 5-2 into your public directory. Alternatively, you can copy the `index.php` file found in the Chapter 5 code download directory to this location.

Listing 5-2. The index.php File

```
<?php

// Identify the location of the application directory in respect to
// the bootstrap file's location, and configure PHP's include_path to
// include the library directory's location

define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/../application'));
set_include_path(
    APPLICATION_PATH . '/../library'. PATH_SEPARATOR . get_include_path()
);

// Give the Zend Framework the ability to load classes on demand,
// as you request them, rather than having to deal with require() statements.

require_once "Zend/Loader.php";
Zend_Loader::registerAutoload();

// Retrieve the bootstrap.php file
try {
    require '../application/bootstrap.php';
} catch (Exception $exception) {
    printf("Could not locate bootstrap.php");
    exit(1);
}

// Start using the front controller in order to route URL requests

Zend_Controller_Front::getInstance()->dispatch();
```

Creating the Bootstrap File

The front controller is invoked from the `bootstrap.php` file, which is in addition responsible for set-

ting up other key parts of the application environment. Place the code found in Listing 5-3 into a file named `bootstrap.php` and save it within your application directory. Alternatively, you can copy the `bootstrap.php` file found in the Chapter 5 code download directory to this location.

Listing 5-3. The `bootstrap.php` File

```
<?php

// Configure the site environment status.

defined('APPLICATION_ENVIRONMENT')
    or define('APPLICATION_ENVIRONMENT', 'development');

// Invoke the front controller

$frontController = Zend_Controller_Front::getInstance();

// Identify the location of the controller directory

$frontController->setControllerDirectory(APPLICATION_PATH . '/controllers');

// Create the env parameter so you can later access the environment
// status within the application.

$frontController->setParam('env', APPLICATION_ENVIRONMENT);

// Clean up allocated script resources

unset($frontController);
```

At this point, your website directory should look like this:

```
/application
  /controllers
  /views
    /scripts
  bootstrap.php
/library
  [contents omitted for space reasons]
/public
  /css
  /images
  /javascript
  .htaccess
  index.php
```

Adjust Your Document Root

Ok, there's just one more step before we can actually begin using our Zend Framework-powered website. Because the `.htaccess` and `index.php` files reside within the `public` directory, the `public`

directory must be identifiable by Apache as the website's document root and root directory. To change these settings, open your `httpd.conf` file and search for `DocumentRoot`. Until now the `DocumentRoot` directive has been set to `C:\apache\htdocs\gamenomad`, however now the `public` directory must serve as that root, so change this directive to read:

```
DocumentRoot "C:/apache/htdocs/gamenomad/public"
```

From there, scroll down about one-half page and you'll find another directive named `Directory`, with a preceding comment stating "This should be changed to whatever you set `DocumentRoot` to". Change this directory setting to read:

```
<Directory "C:/apache/htdocs/gamenomad/public">
```

Take care to change this specific `Directory` directive (the one with this particular preceding comment), as several other `Directory` directives exist throughout the `httpd.conf` file.

Save the `httpd.conf` file and restart Apache in order for these changes to take effect. Congratulations! The Zend Framework has been installed. To test the installation, proceed to the next step.

Adjusting Your Document Root on a Shared Host

Of course, if you're using the Zend Framework on a shared hosting provider, you almost certainly won't have access to the `httpd.conf` file. However you'll typically be able to change the `DocumentRoot` setting using some alternative administrative interface. For instance, my eApps (`http://www.eapps.com/`) control panel offers a "Custom Settings" interface which gives users the ability to override default web server settings, shown in Figure 5-3.



Figure 5-3. Changing the DocumentRoot on my shared host

Upgrading the Zend Framework

Because any custom code you create is found solely in the application directory, upgrading the Zend Framework is as simple as swapping out the existing library directory with the directory of the same name offered in a newer framework release. Before performing the upgrade, always be sure to consult the `README` file accompanying each version download to verify there are no known compatibility

issue with earlier versions.

Step #4. Testing Your Installation

The easiest way to test your installation is to create your first controller and corresponding view, and then call that controller from within the browser! All controllers inherit from the framework's `Zend_Controller_Action` class, which embodies fundamental controller-related behavior that must be available to your controllers in order to function properly. For purposes of demonstration, let's create a simple controller named `Index` which contains a single action also named `index`. This file should be named identically to the class (`IndexController.php`) and saved to the `controllers` directory. The controller is shown in Listing 5-4.

Listing 5-4. The IndexController.php script

```

01 <?php
02     class IndexController extends Zend_Controller_Action
03     {
04         public function indexAction()
05         {
06             $this->view->gameCount = 457;
07         }
08     }
09 ?>
```

The code breakdown follows:

- Line 02 creates the `Index` controller, inheriting from the class `Zend_Controller_Action`.
- Line 04 creates the `index` action. Note this is a public method, which makes it possible for the action to be invoked when requested.
- By default any variable defined in an action is not accessible by the corresponding view. You can however make a variable available in the view scope when it's declared as is done on Line 06. You can access and manipulate a variable declared in this manner within the class just as you would any other variable; just remember it will ultimately be accessible from the view, as you'll soon see.

Next let's define the corresponding view. Each action is coupled with a view named identically to the action, in this case `index`. Furthermore, all views use the extension `.phtml`, meaning this view should be named `index.phtml`. Create the file *and place it in a directory named index within the views/scripts directory*. This view is shown in Listing 5-5.

Listing 5-5. The index.phtml File

```

01 <h1>Welcome to GameNomad!</h1>
02 <p>Total games in database: <?php echo $this->gameCount; ?></p>
```

The code breakdown follows:

- Line 01 shows you can insert HTML into a view just as you would any typical PHP script. Likewise, as is shown in Line 02 the PHP code can be intermingled with the HTML simply by using the PHP escape sequences (<?php and ?>).
- Line 02 outputs the \$gameCount variable set inside the action. Pay particular attention to how it's invoked, because referring to \$gameCount or \$this->view->gameCount won't work!

At this point, your website directory structure should look like this:

```
/application
    /controllers
        IndexController.php
    /views
        /scripts
            /index
                index.phtml
        bootstrap.php
/library
/public
    /css
    /images
    /javascript
    .htaccess
    index.php
```

Next, call the action through the browser. You can call it in several ways:

- `http://localhost`: This is because the framework considers the `Index` controller to be by default the "home page" controller, and within the controller, the `index` action to be the default action.
- `http://localhost/index`: A more explicit call will specifically request the `Index` controller, leaving it to the framework to retrieve the `index` action by default should no other action be requested.
- `http://localhost/index/index`: The most explicit of the three calls, this asks the framework to specifically invoke the `Index` controller, and within it, the `index` action.

Calling the action will unsurprisingly produce the output shown in Figure 5-4.



Figure 5-4. Output from the Index controller's index action

Congratulations, you've just created your first working Zend Framework-powered website! Before moving on to the next section, try creating a few other controllers and actions, to really get a feel for the process. In this next step, we'll cover several other fundamental tasks you'll likely undertake when

setting up a Zend Framework-driven site.

Step #5. Creating the Error Controller

While you might be tempted to barrel ahead with the creation of other controllers such as `AboutController` and `ContactController`, I suggest starting with the `ErrorController`. This controller is special to the Zend Framework, as it will automatically be executed should errors occur due to missing controllers or action methods, for instance.

This `ErrorController` includes a single action titled `errorAction()` which will execute when a website error has occurred. You can customize this action to respond to errors with an impressive degree of accuracy, executing custom responses based on the HTTP response code (404 for file not found or 500 for internal server error). Using other readily available environment variables you can also customize the action to respond differently based on the application environment status (more about this Zend Framework feature later in the chapter), offering more information if you're still developing the website, and less if the website is live. Of course, the `errorAction()`'s purpose isn't limited to simply outputting an error message; you can also perform other tasks such as emailing diagnostic information to an administrator's address.

The `ErrorController` shown in Listing 5-6 demonstrates some of the features at your disposal for managing errors.

Listing 5-6. A typical ErrorController

```
01 <?php
02
03 /**
04  * ErrorController
05 */
06 class ErrorController extends Zend_Controller_Action
07 {
08
09     public function errorAction()
10     {
11
12         // Retrieve the type of error
13         $errors = $this->_getParam('error_handler');
14
15         // Produce an appropriate message based on the error type
16         switch ($errors->type) {
17             case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION:
18                 // 404 error
19                 $this->getResponse()->setHttpStatusCode(404);
20                 $this->view->message = 'The page could not be found';
21                 break;
22             default:
23                 // Application error
24                 $this->getResponse()->setHttpStatusCode(500);
25                 $this->view->message = 'An application error has occurred';
```

```

26         break;
27     }
28
29     // Pass the environment to the view script
30     $this->view->env = $this->getInvokeArg('env');
31
32     // Pass the exception object to the view
33     $this->view->exception = $errors->exception;
34
35     // Pass the request to the view
36     $this->view->request = $errors->request;
37 }
38 }
39
40 ?>
```

Let's examine some of Listing 5-6's more interesting features:

- Line 13 retrieves the `error_handler` parameter, which the Zend Framework will automatically set in the case of an error. The `$errors` variable can subsequently be compared against a series of predefined values. In this case, it's compared against the `ZEND_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION` value; if it matches, the 404 HTTP response code will be set, and an appropriate message prepared for display within the view. If no match is found, a default HTTP response code is assigned and a general message prepared.
- Lines 30, 33, and 36 aggregate diagnostic information which we'll subsequently use in an attempt to determine the source of the error during the development process.

As is typical, the error action is accompanied by a corresponding view. Listing 5-7 presents this view.

Listing 5-7. The GameNomad error action view

```

01 <h1>An Error Has Occurred</h1>
02 <h2><?= $this->message ?></h2>
03
04 <? if ($this->env == 'development') { ?>
05   <h3>Exception information:</h3>
06   <p>
07     <b>Message:</b> <?= $this->exception->getMessage(); ?>
08   </p>
09
10   <h3>Stack Trace:</h3>
11   <pre><?= $this->exception->getTraceAsString(); ?></pre>
12
13   <h3>Request Parameters:</h3>
14   <pre><? var_dump($this->request->getParams()); ?></pre>
15 <? } else { ?>
16
17   <p>
```

```

18     An error has occurred. If this error persists, please contact
19     the site administrator.
20     </p>
21
22 <? } ?>
```

This should be relatively straightforward to understand, however let's touch upon some highlights:

- Line 04 examines the `$this->env` variable. If set to development, a detailed set of error diagnostics will be output to the browser screen (lines 05-14). Otherwise, a general error message will be output.
- The stack trace and request parameters output on lines 11 and 14 can be very helpful in terms of diagnosing the cause of an error. While the amount of data may seem overwhelming at first, over time you'll consider this output a valuable part of your development practice.

Step #6. Creating the Website Layout

As mentioned earlier in this chapter, one of the primary reasons you'll want to use a framework is to eliminate code redundancy. Of course, the term code isn't solely limited to PHP code; it also includes HTML and other design assets such as graphics. In the past chapters we successfully eliminated redundancy by wrapping scripts in headers and footers, using the `require_once()` statement to do so. While this works fine, the Zend Framework offers an even more convenient solution known as the layout. The layout is a single file containing the site header, footer, and other global elements. Named `layout.phtml`, you'll place this file within the `views/scripts` directory, and it will subsequently be retrieved with each action call. Within the layout a call to `$this->layout()->content` tells the framework where exactly to insert action's corresponding view:

```
<?php echo $this->layout()->content; ?>
```

Listing 5-8 offers an example layout file, followed by a few comments regarding its contents.

Listing 5-8. The layout.phtml file

```

01 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
02     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml">
04     <head>
05         <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
06         <title>My Game Collection</title>
07         <link rel="stylesheet" type="text/css" href="/css/styles.css" />
08     </head>
09     <body>
10
11         <div id="logo">
12             
13         </div>
14
```

```

15  <div id="container">
16      <?php echo $this->layout()->content; ?>
17  </div>
18
19  </body>
20 </html>
```

Some code commentary follows:

- Line 07 references a style sheet. Notice how it points to a directory named `css` residing in the website's root directory (identified by the preceding forward slash). Because all requests are routed through the `index.php` file, which resides in the `public` directory, the `public` directory is actually considered to be the website's root directory despite an examination of the website's hierarchical structure seeming to indicate otherwise! This notion applies to all CSS, JavaScript, and images found in the site, as is also demonstrated on Line 12.
- Line 16 is responsible for displaying the requested action's view contents.

Add the `layout.phtml` file to the `views/scripts` directory (along with placing the CSS and image files in their appropriate locations). Next you'll need to make a minor modification to the `bootstrap.php` file, adding the following lines:

```
// Layout Configuration
Zend_Layout::startMvc(APPLICATION_PATH . '/layouts/scripts');
$view = Zend_Layout::getMvcInstance()->getView();
unset($view);
```

Once the `bootstrap.php` file is saved, call the `Index` controller's `index` action anew. You'll be greeted with the output shown in Figure 5-5.



Figure 5-5. Using a layout template

Customizing the Page Title

Based on Listing 5-8, the site's title will always be set to "My Game Collection". You can change this easily enough to reflect the page-specific contents. Adjust the layout's `<title>` tag so it looks like this:

```
<title>$this->pageTitle</title>
```

Within each controller action, you can set the title to correspond with the action's purpose:

```
$this->view->pageTitle = "Welcome to GameNomad";
```

Creating a View Helper

You'll often want to repeatedly perform complex logic within your code, such as formatting a user's birthday a certain way, or rendering a certain icon based on a preset value. To eliminate the redundancy within your code, you can package this code within classes known as *view helpers*, and simply call the view helper as necessary. The Zend framework comes with quite a few such helpers, the majority used to facilitate in the creation of forms within views (consult the manual for more information about these helpers). However you can create your own view helpers, a task which will prove invaluable as your site grows in size and complexity.

While there are several supported approaches for creating a view helper, the easiest is to extend the framework's `Zend_View_Helper_Abstract` class, as is shown in Listing 5-9. I use this helper on the GameNomad website so as to refer to the proper possessive gender based on the gender designation ("m" or "f") within a user's profile. Once created, you can execute the helper from within your views like so:

```
<?php echo $this->Gender("m"); ?>
```

As you can see from the listing, there's remarkably little overhead involved, although there are a few rules you need to follow when creating and configuring view helpers, which I'll enumerate following the listing.

Listing 5-9. A view helper for determining gender (Gender.php)

```
01 <?php
02
03     class My_View_Helper_Gender extends Zend_View_Helper_Abstract
04     {
05
06         /**
07          * Produces string based on whether value is
08          * masculine or feminine
09          *
10         * @param string $gender
11         * @return string
12         */
13         public function Gender($gender)
14         {
15
16             if ($gender == "m") {
17                 return "his";
18             } else {
19                 return "her";
20             }
21         }
22     }
```

```

22      }
23
24  }
25
26 ?>

```

The code breakdown follows:

- Line 03 defines the helper class. Notice the naming convention and format used in the class name. While not a requirement, it is recommended you prefix the helper name with a string such as "My_View_Helper" or perhaps your initials, such as "WJG_View_Helper", which makes the purpose and owner of the file perfectly clear. Also, while not a requirement, for reasons of convenience you should extend this class from `Zend_View_Helper_Abstract` as doing so will eliminate the need to include additional configuration-related code which I'll not discuss here.
- Line 13 defines the class method, `Gender()`. This method must be named identically to the concluding part of your class name (`Gender`, in this case). Likewise, the helper's file name must be named identically to the method, and include the `.php` extension (`Gender.php`).

Organizing Your Helpers

I recommend organizing your helpers in a single location for practical purposes. For instance, I store mine in a directory named `My/Helper` which resides within the `application` directory. You'll need to update the `bootstrap.php` file so this directory is accessible to the framework, done using the following command:

```
$view->setHelperPath('C:\apache\htdocs\www.gamenomad.com\application\My\Helper',
                      'My_View_Helper');
```

The second parameter must match the prefix used to name your helper class, in this case `My_View_Helper`. If you omit this prefix, the framework will look for a class prefixed `Zend_View_Helper`.

If you're following along with the website file structure's evolution, it now looks like this:

```

/application
    /controllers
        IndexController.php
    /My
        /Helper
            Gender.php
    /views
        /scripts
            /index
                index.phtml
        bootstrap.php
/library
/public

```

```
/css
    styles.css
/images
    logo.png
/javascript
.htaccess
index.php
```

Creating a View Partial

Many web pages are built from components which are found time and again throughout the web site. For instance, you might insert information about the currently hottest selling video game title within select pages. The HTML might look like this:

```
<p>
    Hottest game this hour:<br />
    <a href="/games/show/<?= $this->asin; ?>"><?= $this->title; ?></a>
</p>
```

So how can we organize these templates for easy reuse? The Zend Framework makes it easy to do so, calling them *partials*. A partial is simply a template which can be retrieved and rendered within a page, meaning you can use it repeatedly throughout the site. If you later decide to modify the partial to include for instance the current Amazon sales rank, the change will immediately occur within each location the partial is referenced.

However partials have an additional useful feature in that they can contain their own variables and logic without having to worry about potential clashing of variable names. This is useful because the variables `$this->asin` and `$this->title` may already exist in the page calling the partial, but because of this behavior, we won't have to worry about odd side effects.

For organizational purposes, I prefix all partial filenames with an underscore, and store them within the `views/scripts` directory. For instance, the above partial might be named `_hottestgame.phtml`. To insert a partial into a view, use the following call:

```
<?= $this->partial('_hottestgame.phtml',
                    array('asin' => $hottestASIN, 'title' => $hottestTitle)); ?>
```

Notice how each key in the array corresponds to a variable found in the referenced partial. You'll also need to add the `views/scripts` directory to PHP's include path, which you can do via the `php.ini` file or by adding this line to the `index.php` file:

```
set_include_path('..../application/views/scripts' . PATH_SEPARATOR .
    get_include_path());
```

The Partial Loop

The Zend Framework offers a partial variation useful for looping purposes. Revising the hottest game partial, suppose you instead wanted to occasionally offer a list of the five hottest selling games. You can create a partial which represents just one entry in the list, and use the `PartialLoop` to iterate

through the games and format them accordingly. The revised partial might look like this:

```
<li><a href="/games/show/<?= $this->asin; ?>"><?= $this->title; ?></li>
```

Using the PartialLoop construct, you can pass along a a partial and array of values, prompting PartialLoop to iterate until the array values have been exhausted:

```
<ul>
<?= $this->partialLoop('_hottestgames.phtml', array('asin' => '', 'title' => '',
    'asin' => '', 'title' => '', 'asin' => '', 'title' => ''))>
</ul>
```

Step #7. Creating a Configuration File

As has been said time and again throughout this book, one of a programmer's most important duties is to eliminate redundancy at every opportunity. This opportunity for redundancy is perhaps no more apparent than when the need arises to embed data classified as configuration-related into a website. E-mail addresses, URLs, database login parameters, and other information required to properly run the website, such as Amazon and Facebook API keys. It would be easy to simply embed this information into your scripts as needed, however what if you later decided to change the site's main contact e-mail address, which has since been embedded into dozens of different pages as the site grew in size? For that matter, what if you wanted the e-mail address to change depending upon whether the site was still under development or whether it had been deployed for production (live) use? Using the Zend Framework's Zend_Config component, we can manage all of this configuration data in a single location (namely a file named `config.ini`), and easily switch from one configuration set to the next depending on the environment (`development` or `production`).

Listings 5-10 and 5-11 demonstrate the utility of such a feature. Listing 5-10 shows an e-mail address as it would be created when the website was operating in development mode. Notice the URL (`http://beta.gamenomad.com/`) and the reference to the GameNomad bug database (`http://bugs.gamenomad.com/`). When operating in production mode (Listing 5-11), the URL is `http://www.gamenomad.com`, and the contact point is `support@gamenomad.com`.

Listing 5-10. Registration E-mail in Development Mode

Dear Jason,

Welcome to the GameNomad community! To confirm your e-mail address, click on the following URL:

<http://beta.gamenomad.com/gamers/verify/key/zs7lquz958qknj1hqq70gcf89egqsdf1>

Questions? Comments? Contact us at <http://bugs.gamenomad.com/>.

Thank you,
The GameNomad Team

Listing 5-11. Registration E-mail in Production Mode

Dear Jason,

Welcome to the GameNomad community! To confirm your e-mail address, click on the following URL:

<http://www.gamenomad.com/gamers/verify/key/zs7lquz958qknj1hqq70gcg89egqsdf1>

Questions? Comments? Contact us at support@gamenomad.com!

Thank you,

The GameNomad Team

We can create these e-mail variations by passing configuration variables into a template (Listing 5-12).

Listing 5-12. The Registration E-mail Template

Dear {\$firstName},

Welcome to the GameNomad community! To confirm your e-mail address, click on the following URL:

{\$this->config->website->url}/gamers/verify/key/{\$registrationKey}

Questions? Comments? Contact us at \$this->config->email->support

Thank you,

The GameNomad Team

The Zend Framework will be able to identify which set of configuration variables should be passed based on the value of APPLICATION_ENVIRONMENT, defined in the bootstrap.php file:

```
defined('APPLICATION_ENVIRONMENT')
    or define('APPLICATION_ENVIRONMENT', 'development');
```

Also in the bootstrap.php file you'll need to execute two other commands:

```
$config = new Zend_Config_Ini('/home/webadmin/html/application/config.ini',
    APPLICATION_ENVIRONMENT);
Zend_Registry::set('config', $config);
```

The first line identifies the location of the config.ini file, and identifies the defined application environment. The second line uses the Zend_Registry component to register the \$config variable within the website's registry, meaning it can later be retrieved using the following command:

```
$this->config = Zend_Registry::get('config');
```

You'll need to ensure this command executes prior to retrieving the configuration variables from the

`config.ini` file. Of course, the easiest way is to simply call the command directly prior to referencing one or more configuration variables, however as you'll likely need to do so for multiple actions within a controller, later in this chapter I'll show you an easy way to consolidate the number of times you need to make this call.

The config.ini File

The `config.ini` file (located in the `application` directory) is the central repository for your Zend Framework-powered website's configuration data. You're free to name the configuration parameters stored within anything you please, however they typically are identified using a dotted notation like so:

```
database.params.username = gamenomad_prod
```

When referring to the parameter within a script, you'll prefix it with `$this->config` and converting the periods to right-facing arrows (`->`), like so:

```
$this->config->database->params->username
```

Notice how in the same `config.ini` file (Listing 5-13) these parameters are listed twice. Once under the section titled `[production]` and a second time under the section titled `[development]`. The `Zend_Config` component will choose the appropriate parameter according to the aforementioned `APPLICATION_ENVIRONMENT` setting.

Listing 5-13. Sample config.ini file

```
; Production site configuration data
[production]

website.params.url = http://www.gamenomad.com

database.params.host = mysql.gamenomad.com
database.params.username = gamenomad_prod
database.params.password = supersecret
database.params dbname = gamenomad_prod

email.params.support = support@gamenomad.com

; Development site configuration data
[development]

website.params.url = http://beta.gamenomad.com

database.params.host = mysql.gamenomad.com
database.params.username = gamenomad_dev
database.params.password = supersecret
database.params dbname = gamenomad_dev

email.params.support = http://bugs.gamenomad.com
```

Be sure to integrate this powerful component into your website at the earliest opportunity, as it will no doubt save both time and inconvenience as your site grows in size.

Step #8. The init() Method

To close out this chapter introducing the Zend Framework, I'd like to discuss a feature you'll return to time and again, namely the `init()` method. Placed within a given controller, the `init()` method will execute prior to any other method found in the controller, giving you a convenient means for initializing variables and executing tasks which might need to occur prior to any other event related to that controller's operation. For instance, if you know you'll need access to the contents of the `config.ini` file within multiple actions you can add the following command to the `init()` method:

```
$this->config = Zend_Registry::get('config');
```

Listing 5-14 demonstrates this concept in action, retrieving the `$config` variable from the registry, and then using it within multiple actions.

Listing 5-14. The init() method in action

```
class MailController extends Zend_Controller_Action {

    function init() {
        $this->config = Zend_Registry::get('config');
    }

    public function registrationAction()
    {
        echo $this->config->mail->admin;
    }

    public function confirmationAction()
    {
        echo $this->config->db->username;
    }
}
```

Step #9. Creating Action Helpers

Continuing the theme of figuring out ways to eliminate redundancy at every opportunity, the `init()` method is very useful if you want to share access to for instance the configuration file handle across multiple actions, but in all likelihood you'll want access to the configuration file and certain other variables throughout your application. The result is a duplicated `init()` method within each controller, resulting in the need to touch every controller each time you want to make another variable globally available, a clear violation of framework strategy.

Or suppose you require a solution for generating random strings, which might be used when resetting

passwords or within CAPTCHAs (see Chapter 7). It's easy to insert such a method into a controller, however over time it's likely the method will begin to multiply like rabbits as it's needed within other parts of the application.

One of the easiest solutions for remedying both of these problems involves creating an *action helper*. Much like view helpers facilitate the rendering of data within the view, action helpers facilitate the execution of certain tasks within controllers. Once in place, you'll be able to refer to certain variables defined within the helper from any controller, not to mention call methods such as the aforementioned random string generator.

Let's conclude this chapter by creating an action helper which will make the Zend_Registry's \$config variable automatically available to all controllers. I'll also show you how to create a method which can easily be executed within any action.

Creating the Initializer Action Helper

All action helpers inherit from the Zend_Controller_Action_Helper_Abstract class, so let's begin by creating the action helper container:

```
class My_Action_Helper_Initializer extends Zend_Controller_Action_Helper_Abstract
{
}
```

Save this as `Initializer.php` and place it alongside your view helpers in the `application/My/helper` directory. Because action helpers share many of the same characteristics of controllers, we're going to create an `init()` method which will automatically execute when the action helper is invoked. As you'll soon see, by registering the action helper within the `bootstrap.php` file, we will ensure this `init()` method is automatically invoked. Because the action helper is registered in the `bootstrap.php` file, it will execute prior to any controller action, and given the proper commands, will make certain variables available to that action. Here's the `init()` method:

```
public function init()
{
    $controller = $this->getActionController();
    $controller->config = Zend_Registry::get('config');
}
```

Thanks to the way the Zend Framework operates, the action helper knows which controller action has been invoked, meaning we can retrieve the executing controller using line 03. Line 04 will subsequently create a variable named `config` which will be attached to the Zend_Registry's configuration handle. Believe it or not, that's all that's required to assign variables which will then be made available to the controller! But before you can put this action helper into service, you need to identify the location and prefix for your action helpers, just as you did previously during the introduction to the view helper.

Loading the Action Helper

To identify the location and prefix of your action helper, add the following line to your `bootstrap.php` file. Of course, you'll need to change the values of each self-explanatory `addPath()` parameter as necessary:

```
Zend_Controller_Action_HelperBroker::addPath(
    'C:\apache\htdocs\gamenomad.com\application\My\Helper', 'My_Action_Helper');
```

You can then invoke an action helper as needed using the following command

```
$initStuff = Zend_Controller_Action_HelperBroker::getStaticHelper('Initializer');
```

Note I'm just identifying the action helper by the name following the prefix. If you want the `config` variable to be available to all controllers, invoke the helper in your `bootstrap.php` file. Alternatively, if you just wanted these variables or other features to be available within select actions, you can call the helper as needed within the actions. The latter approach is particularly useful when you want to call a method, as is explained next.

Calling an Action Helper Method

I regularly use a method called `generate_id()` for generating random strings. Rather than define this method within every controller, it makes much more sense to define it within an action helper and call it as needed. This approach is not only useful in terms of removing code redundancy within a website, but also in terms of enhancing portability when you require similar features within other websites. You'll create such a method within the action helper as you would any other class, just be sure to declare it as public so the method can be accessed outside of the class:

```
public function generate_id()
{
    return "superrandomstring";
}
```

Once defined, you'll be able to access the method from within any controller (after ensuring it's location is registered within the `bootstrap.php` file), using the following bit of code:

```
$initializer = Zend_Controller_Action_HelperBroker::
    getStaticHelper('Initializer');
$registrationKey = $initializer->generate_id();
```

Conclusion

This chapter covered a significant bit of ground, introducing you to the Zend Framework's fundamental features. In the next chapter we'll build upon what you've learned, introducing the crucial `Zend_Db` component, which makes database access using the Zend Framework a breeze.

CHAPTER 6

Talking to the Database with Zend_Db

Even the simplest of web sites will almost invariably rely upon a database for data management, meaning you're likely to spend almost as much (if not more) time thinking about accessing and managing data as you will working on any other part of your site. While necessary, the end result is you have to essentially simultaneously think in two languages, in our case PHP and SQL, which is surely a drag on efficiency. Further, mixing SQL with the rest of your website's logic is counter to the goal of separating the data, logic, and presentation tiers. So what's a developer to do? After all, it's clearly not possible to do away with the database, but using one at the cost of sacrificing efficiency and sound development practices seems an impractical tradeoff.

Enter *object-relational mapping* (ORM), a programming strategy which can go a long way towards eliminating all of these obstacles while not detracting from your ability to harness the full power of the database. As a Zend Framework adoptee, you have access to a powerful, intuitive ORM made available via the Zend_Db component. In this chapter you'll learn all about this component, along the way gaining valuable experience building key features which will provide you with a sound working understanding of this important aspect of the Zend Framework.

Chapter Steps

The goals of this chapter are accomplished in ten steps:

- **Step #1. Introducing Object-relational Mapping:** In this opening step I'll introduce you to a concept known as object-relational mapping, which is the fundamental premise behind the Zend_Db component which makes database access so easy using the Zend Framework.
- **Step #2. Introducing Zend_Db:** The Zend_Db component is the conduit for talking to a database using the Zend Framework. In this step I'll introduce you to this component, which is so powerful that it almost manages to make database access fun.
- **Step #3. Creating Your First Model:** When using Zend_Db, you'll rely upon a series of classes (known as models) as the conduits for talking to your database, meaning you'll be able to query and manage data without having to write SQL queries! In this step I'll show you how to create a model for managing video game data which we'll subsequently use throughout the remainder of this chapter.
- **Step #4. Querying Your Models:** With the Game model created, we can set about pulling data from the games table using the query syntax exposed through the Zend_Db component. In this step I'll introduce you to this syntax, showing you how to retrieve data from the games table in a variety of useful ways.
- **Step #5. Creating a Row Model:** Row models give you the ability to query and manipulate tables at the row-level. In this step I'll show you how to create and use this powerful feature.
- **Step #6. Inserting, Updating, and Deleting Data:** Just as you can retrieve data through the

Zend_Db component, so can you use the component to insert, update, and delete data. In this step I'll show you how.

- **Step #7. Creating Model Relationships:** Zend_Db can account for table relationships, allowing you to deftly interact with the database in amazingly convenient ways. In my experience this is one of the component's most compelling features, and in this step I'll show you how to take advantage of it by defining a second model named Platform (for managing gaming platforms, such as Xbox 360 and Nintendo Wii), and tying it to the Game model so we can associate each game with its platform.
- **Step #8. JOINing Your Data:** Most of your time will be spent dealing with simple queries, however you'll occasionally be wanting for a more powerful way to assemble your data. In this step I'll introduce you to the powerful SQL statement known as the join, which will open up a myriad of new possibilities to consider when querying the database.
- **Step #9. Paginating Results with Zend_Paginator:** When dealing with large amounts of data, for usability reasons you'll probably want to spread the data across several pages, or paginate it, so the user can easily peruse it without having to endure long page loading times. But manually splitting retrieved data into multiple pages is a more difficult task than you might think; thankfully the Zend_Paginator component can do the dirty work for you, and in this step I'll show you how to use it.
- **Step #10. Creating and Managing Views:** As the complexity of your data grows, so will the SQL queries used to mine it. Rather than repeatedly refer to these complex queries within your code, you can bundle them into what's known as a *view*, which stores the query within the database. Using an alias assigned to that view, you can now query the data using a far simpler syntax.

Step #1. Introducing Object-relational Mapping

Object-relational mapping (ORM) works by providing the developer with an object-oriented solution for interacting with the database, written in the same language used to power the website. Each database table is mapped to a corresponding class. This class is not only able to communicate with the table, performing tasks such as selecting, inserting, updating, and deleting data, but can also be extended by the developer to include other behavior, such as data validation and custom queries. Best of all, this approach not only makes it possible for the developer to focus on the primary programming language when building the application, but also isolates the database-related actions and therefore be able to more effectively maintain the code throughout the application's lifetime.

Over the years PHP developers have devised many solutions which allow users to take advantage of this powerful concept. Accordingly, it might come as no surprise that the Zend Framework developers made an ORM solution one of the early project priorities. The fruits of their labor are apparent within the Zend_Db component, introduced next.

Step #2. Introducing Zend_Db

The Zend_Db component provides Zend Framework users with a flexible, powerful, and above all, easy, solution for integrating a database into a website. It's easy because Zend_Db almost completely eliminates the need to write SQL statements (although you're free to do so if you'd like), instead providing you with an object-oriented interface for retrieving, inserting, updating, and deleting data from the database.

TIP. In addition to MySQL, Zend_Db supports a number of other databases, including DB2, Microsoft SQL Server, Oracle, PostgreSQL, SQLite, and others.

In this Step I'm going to take the liberty of foregoing much of the introductory material found in the Zend Framework manual, having the belief the Zend_Db component is so intuitive that you'll be able to grasp the basic syntax almost immediately. Either way, if you're looking for a complete treatment on the component, I recommend taking some time to peruse the excellent documentation at <http://framework.zend.com/manual/en/>.

Connecting to the Database

Before doing anything with the database, you'll need to connect to it. As discussed in Chapter 5, the most effective way to manage configuration data is via the `config.ini` file, so let's begin by defining the database connection parameters there:

```
database.params.host      = localhost
database.params.username = gamenomad_user
database.params.password = supersecret
database.params dbname   = gamenomad_prod
```

Of course, you'll likely want to define these parameters twice, once in the `[production]` section of the `config.ini` file, and a second time within the `[development]` section. If you can't recall the reason for doing so, consult Step 7 of Chapter 5.

Within the `bootstrap.php` file you can create the database connection using these parameters:

```
01 $config = new Zend_Config_Ini('/home/webadmin/html/application/config.ini',
02     APPLICATION_ENVIRONMENT);
03 ...
04
05 // Instantiate the database
06 $db = Zend_Db::factory('PDO_MySQL', array(
07     'host'      => $config->database->params->host,
08     'username'  => $config->database->params->username,
09     'password'  => $config->database->params->password,
10     'dbname'    => $config->database->params->dbname
11 ));
12
13 Zend_Db_Table_Abstract::setDefaultAdapter($db);
```

Let's review this listing:

- Line 01 identifies the location of the `config.ini` file, and defines the current environment setting. This was discussed in Step 7 of Chapter 5.
- Lines 06-11 perform the database connection. The `PDO_MySQL` parameter tells the Zend Framework we'll be connecting to a MySQL database. Several other databases are supported, among them SQLite, PostgreSQL, and Microsoft SQL Server. Also provided are the host, username, password, and database name parameters, which were previously set within the `config.ini` file.
- Line 13 is particularly important, because it tells the Zend Framework to automatically use this connection for all subsequent interaction with the database. While not required, this saves us from the hassle of having to explicitly identify the connection when interacting with the database.

Once you've successfully made the connection, it's time to start creating the classes we'll use to interact with the database. This is done by creating a model. I'll show you how to do this next.

Step #3. Creating Your First Model

`Zend_Db` is a particularly compelling database access solution for developers because it was built with the assumption the developer would be most comfortable interacting with the database by way of the very same language used to build the application, in our case, PHP. The developer employs an object-oriented approach, building classes which represent the various tables and even rows within the database. The `Zend_Db` component automatically enhances these special classes, giving you the ability to interact with the database using a variety of well-defined methods. These well-defined methods are immediately available because when creating the class you extend the `Zend_Db_Table_Abstract` class (or the `Zend_Db_Table_Row_Abstract` class when modeling rows).

As usual, the best way to learn how all of this works is by using it. So let's begin by creating a class which we'll use to query the games table. Save this class as `Game.php`, and store it in a directory named `models` which resides within your `application` directory. We'll start with a very simple class (Listing 6-1), and expand the class as your understanding grows.

Listing 6-1. The Game model

```
01 class Game extends Zend_Db_Table_Abstract
02 {
03     protected $_name = 'games';
04     protected $_primary = 'id';
05 }
```

Although just five lines of code, there are some pretty important things going on in this listing:

- Line 01 defines the name of the model (`Game`), and specifies that the model should extend the `Zend_Db_Table_Abstract` class. The latter step is important because in doing so, the `Game` model will inherit all of the traits the `Zend_Db` grants to models. As for naming your model,

I prefer to use the singular form of the word used for the table name (in this case, the model name is `Game`, and the table name is `games`).

- Because of my personal preference for using singular form when naming models, in line 03 I need to override the Zend Framework's default behavior of presuming the model name exactly matches the name of the database table. Neglecting to do this will cause an error, because the framework will presume your table name is `game`, rather than `games`.
- Line 04 identifies the table's primary key. By default the framework will presume the primary key is an automatically incrementing integer named `id`, so this line is actually not necessary; I prefer to include the line as a cue for fellow developers. Of course, if you were using some other value as a primary key, for instance a person's social security number, you would need to identify that column name as I've done here.

Once this model has been created and saved to the appropriate location (`application/models`), modify your `index.php` file to add the models location to PHP's include path:

```
set_include_path('..../application/models' . PATH_SEPARATOR . get_include_path());
```

Congratulations, you've just created an interface for talking to the database's `games` table. What next? Let's start with some queries.

VIDEO. Creating Your First Model

In this video you'll learn how to create a table model, tie it to an existing database table, and query that table via the model. Watch the video at <http://www.easyphpwebsites.com/zfw/videos/>.

Step #4. Querying Your Models

It's likely the vast majority of your time spent with the database will involve retrieving data. Using the Zend_Db component selecting data can be done in a variety of ways. In this section I'll demonstrate several options at your disposal.

Querying by Primary Key

The most commonplace method for retrieving a table row is to query by the row's primary key. The following example queries the database for the row associated with the primary key 1:

```
$game = new Game();
$cod = $game->find(1);
echo "{$cod[0]->title} (ASIN: {$cod[0]->asin})";
```

Returning:

```
Call of Duty 4: Modern Warfare (ASIN: B0016B28Y8)
```

Alternatively, you can forego the need to refer to an index offset by using the `current()` method:

```
$game = new Game();
$cod = $game->find(1)->current();
echo "{$cod->title} (ASIN: {$cod->asin})";
```

But why do we even have to deal with index offsets in the first place? After all, using the primary key implies there should only be one result anyway, right? This is because the `find()` method also supports the ability to simultaneously query for multiple primary keys, like so:

```
$cod = $game->find(array(1,4));
```

Presuming both of the primary keys exist in the database, the row associated with the primary key 1 will be found in offset [0], and the row associated with the primary key 4 will be found in offset [1].

Querying by a Non-key Column

You'll inevitably want to query for rows using criteria other than the primary key. For instance, various features of the GameNomad site search by ASIN. If you only need to search by ASIN at a single location within your website, you can hardcode the query, like so:

```
$game = new Game();
$query = $game->select();
$query->where("asin = ?", "B0016B28Y8");
$cod = $game->fetchRow($query);
echo "{$cod->title} (ASIN: {$cod->asin})";
```

Note that unlike when searching by primary key, there's no need to specify an index offset when referencing the result. This is because the `fetchRow()` method will always return only one row.

Because it's likely you'll want to search by ASIN at several locations within the website, the more efficient approach is to define a Game class method for doing so:

```
function findByAsin($asin) {
    $query = $this->select();
    $query->where('asin = ?', $asin);
    $result = $this->fetchRow($query);
    return $result;
}
```

Now searching by ASIN couldn't be easier:

```
$game = new Game();
$cod = $game->findByAsin("B0016B28Y8");
```

Notice how the method refers to `$this` rather than `$game`. This is because we're inside the `Game` class, so `$this` can be used to refer to the calling object, saving you a bit of additional coding.

Retrieving Multiple Rows

To retrieve multiple rows based on some criteria, you can use the `fetchAll()` method. For instance, suppose you wanted to retrieve all games with a price higher than \$44.99:

```
$game = new Game();
$query = $game->select();
$query->where('price > ?', '44.99');
$results = $this->fetchAll($query);
```

To loop through these results, you can just use PHP's built-in `foreach` construct:

```
foreach($results AS $result) {
    echo "{$result->title} ({$result->asin})<br />";
}
```

Custom Search Methods in Action

GameNomad.com uses the ASIN each game's ASIN to determine what game the user would like to view. For instance, to retrieve the Gears of War 2 (Xbox 360) page, you would navigate to the following page:

```
http://www.gamenomad.com/games/show/B000ZK9QD2
```

As you've just learned, in order to retrieve the row associated with data which isn't the primary key yet is nonetheless unique to the desired row, you'll need to create a custom method. To retrieve games according to ASIN, add the aforementioned `findByAsin()` method to the `Game` class. Once added, you can then pass an ASIN to the method, thereby retrieving the desired game:

```
$asin = $this->request->getParam('asin');
$game = new Game();
$this->view->game = $game->findByAsin($asin);
```

Your searches don't have to be restricted to retrieving a single row. For instance, the following `Game` model method retrieves all games in which the title includes a particular keyword:

```
function getGamesMatching($keywords)
{
    $query = $this->select();
    $query->where('title LIKE ?', "%$keywords%");
    $query->order('title');
    $results = $this->fetchAll($query);
    return $results;
}
```

You can then use this method within a controller action like so:

```
// Retrieve the keywords
$this->view->keywords = $this->request->getParam('keywords');
```

```
$game = new Game();
$this->view->games = $game->getGamesMatching($this->view->keywords);
```

Counting Rows

All of the examples demonstrated so far have presumed one or more rows will actually be returned. But what if the primary key or other criteria aren't found in the database? The beauty of Zend_Db is that it allows you to use standard PHP constructs to not only loop through results, but count them. Therefore, the easiest way to count your results is using PHP's `count()` function. I typically use `count()` within the view to determine whether I need to tell the user no entries are available, or whether I need to loop through the results:

```
<?php if (count($this->games) > 0) { ?>

    <?php foreach($this->games AS $game) { ?>
        <p><?= $game->title; ?>(<?= $game->title; ?>)</p>
    <?php } ?>

<?php } else { ?>

    <p>
        No new games have been added over the past 24 hours.
    </p>

<?php } ?>
```

Selecting Specific Columns

So far we've been retrieving all of the columns in a given row, but what if you only wanted to retrieve each game's title and ASIN? Using the `from()` method, you can identify specific columns for selection:

```
$game = new Game();
$query = $game->select();
$query->from('games', array('asin', 'title'));
$query->where('asin = ?', 'B0016B28Y8');
$cod = $game->fetchRow($query);
echo "{$cod->title} ({$cod->price})";
```

Ordering the Results by a Specific Column

To order the results according to a specific column, use the `ORDER` clause:

```
$game = new Game();
$query = $game->select();
$query->order('title ASC');
$rows = $game->fetchAll($query);
```

To order by multiple columns, pass them to the `ORDER` clause in the order you'd like them to take

precedence:

```
$query->order('release_date, price ASC');
```

This would have the effect of ordering the games starting with the earliest release dates. Should two games share the same release date, their precedence will be determined by the price.

Limits the Results

To limit the number of returned results, you can use the `LIMIT` clause:

```
$game = new Game();
$query = $game->select();
$query->where('title LIKE ?', $keyword);
$query->limit(15);
$rows = $game->fetchAll($query);
```

You can also specify an offset by passing a second parameter to the clause:

```
$query->limit(15, 5);
```

Executing Custom Queries

Although Zend_Db's built-in query construction capabilities should suffice for most situations, you might occasionally want to manually write and execute a query. To do so, you can just create the query and pass it to the `fetchAll()` method, however before doing so you'll want to filter it through the `quoteInto()` method, which will filter the data by delimiting the string with quotes and escaping special characters.

```
$db = Zend_Registry::get('db');
$title = "Cabela's Dangerous Hunts '09";
$sql = $db->quoteInto("SELECT asin, title FROM games where title = ?", $title);
$results = $db->fetchAll($sql);
echo count($results);
```

The `quoteInto()` method is kind of a fix-all for query parameters, both escaping special characters and delimiting it with the necessary quotes.

Step #5. Creating a Row Model

It's important you understand that the `Game` model represents the `games` table, and not each row found in that table. For example, you might use this model to retrieve a particular row, determine how many rows are found in the table, or figure out what row contains the highest priced game. As you've seen, you can also output one or several retrieved rows. However, when performing operations *specific* to a certain row, such as finding the latest Amazon.com sales rank of a row you've retrieved using the `Game` model, you'd preferably do so using another model. To do this, you'll want to associate the row-specific model with the corresponding table-specific model. To do so, add this line to the `Game` model:

```
protected $_rowClass = 'GameRow';
```

Next, create the GameRow model, saving it as `GameRow.php` and storing it within the `application/models` directory:

```
class GameRow extends Zend_Db_Table_Row_Abstract
{
    function latestSalesRank()
    {
        $rank = new Rank();
        $query = $rank->select('rank');
        $query->where('game_id = ?', $this->id);
        $query->order('created_at DESC');
        $query->limit(1);
        $row = $rank->fetchRow($query);
        return $row->rank;
    }
}
```

To demonstrate this feature, suppose you wanted to output the sales ranks of all video games released to the market before January 1, 2009. First we'll use the `Game` model to retrieve the games. Second we'll iterate through the array of games (which are objects of type `GameRow`), calling the `latestSalesRank()` method to output the latest sales rank:

```
$game = new Game();
$query = $game->select()->where("release_date < ?", "2009-01-01");
$results = $game->fetchAll($query);
foreach($results AS $result)
{
    echo "{$result->title} (Latest Sales Rank: {$result->latestSalesRank()}<br />";
}
```

Executing this snippet produces output similar to the following:

```
Call of Duty 4: Modern Warfare (Latest Sales Rank: 14)
Call of Duty 2 (Latest Sales Rank: 2,208)
NBA 2K8 (Latest Sales Rank: 475)
NHL 08 (Latest Sales Rank: 790)
Tiger Woods PGA Tour 08 (Latest Sales Rank: 51)
```

Step #6. Inserting, Updating, and Deleting Data

You're not limited to using `Zend_Db` to simply retrieve data from the database; you can also insert new rows, update existing rows, and delete them.

Inserting a New Row

To insert a new row, you can use the `insert()` method, passing an array of values you'd like to insert:

```
$game = new Game();

$data = array(
    'asin' => 'B000TG530M',
    'title' => 'Call of Duty 4: Modern Warfare',
    'platform_id' => 1,
    'release_date' => '2007-11-05',
    'created_at' => date('Y-m-d H:i:s'),
    'price' => '59.99'
);

$game->insert($data);
```

Updating a Row

To update a row, you can use the `update()` method, passing along an array of values you'd like to change, and identifying the row using the row's primary key or another unique identifier:

```
$game = new Game();

$data = array(
    'price' => 49.99
);

$where = $game->getAdapter()->quoteInto('id = ?', '42');

$game->update($data, $where);
```

Alternatively, you can simply change the attribute of a row loaded into an object of `Zend_Db_Table_Abstract`, and subsequently use the `save()` method to save the change back to the database:

```
$game = new Game();

// Find Tiger Woods PGA Tour 09
$golf = $game->findByAsin('B00164TDUC');

// Change the price to $39.99
$golf->price = 39.99;

// Save the change back to the database
$golf->save();
```

Deleting a Row

To delete a row, you can use the `delete()` method:

```
$game = new Game();

$where = $game->getAdapter()->quoteInto('asin = ?', 'B001B0BB3S');
```

```
$game->delete($where);
```

Step #7. Modeling Table Relationships

Because even most rudimentary database-driven websites rely upon multiple related tables for data management, it's fair to say you'll spend a good deal of time as a developer writing code for effectively managing these relations. Recognizing this, the Zend developers integrated several powerful features capable of dealing with related data. Most notably, these features allow you to transparently treat a related row as another object attribute. For instance, you'll recall from Chapter 4 the `games` table referred to the `platform_id` foreign key, which associates with a primary key found in the `platforms` table. To refresh your memory, the `platforms` table looks like this:

```
CREATE TABLE platforms (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL
);
```

Using Zend_Db's facility for linking related data, you can retrieve a game's platform name using this simple call:

```
$game->findParentRow('Platform')->name
```

Likewise, you can retrieve dependent rows using the `findDependentRowset()` method. For instance, the following snippet will retrieve the count of games associated with the Xbox 360 platform (identified by a primary key of 1):

```
$platform = new Platform();

// Retrieve the platform row associated with the Xbox 360
$xbox360 = $platform->findById(1);

// Retrieve all games associated with platform ID 1
$games = $xbox360->findDependentRowset('Game');

// Display the number of games associated with the Xbox 360 platform
echo count($games);
```

Alternatively, you can use a "magic method", made available automatically to related models. For instance, dependent games can also be retrieved using the `findGame()` method:

```
$platform = new Platform();

// Retrieve the platform row associated with the Xbox 360
$xbox360 = $platform->findById(1);

// Retrieve all games associated with platform ID 1
$games = $xbox360->findGame();
```

```
// Display the count
echo count($games);
```

The method is named `findGame()` because we're finding the platform's associated rows in the `Game` model. If the model happened to be named `Games`, we would use the method `findGames()`.

Finally, there's still another magic method at your disposal, in this case, `findGameByPlatform()`:

```
$platform = new Platform();

// Retrieve the platform row associated with the Xbox 360
$xbox360 = $platform->findById(1);

// Retrieve all games associated with platform ID 1
$games = $xbox360->findGameByPlatform();

// Display the count
echo count($games);
```

To use these features, you need to configure your models so they are able to recognize the relationships. I'll show you how to do this next.

VIDEO. Working with Table Relations

Arguably one of the most confusing features of Zend_Db, understanding how to create table relations is nonetheless crucial to mastering this powerful component. This video dispels that confusion, showing you how to configure table relations and use them in a variety of ways. Watch the video at <http://www.easyphpwebsites.com/zfw/videos/>.

Configuring Your Models to Support Relationships

You'll configure the relationship models by formally defining the relationships within the model. For instance, the `games` table is dependent upon the `platforms` table, so let's start by defining the `games` table as such within the `Platform` model:

```
01 class Platform extends Zend_Db_Table_Abstract
02 {
03
04     protected $_name = 'platforms';
05     protected $_primary = 'id';
06
07     protected $_dependentTables = array('Game');
08 }
```

Line 07 defines the relationship, informing Zend_Db of the existence of a column within the `Game` model which stores a foreign key pointing to a row managed by the `Platform` model. If a model happens to be a parent for more than one other model, for instance the `Game` model is a parent to both the `Rank` and the `GameUser` models, you would just revise that statement to look like this:

```
07 protected $_dependentTables = array('Rank', 'GameUser');
```

Next, you need to reciprocate the relationship within the `Game` model, albeit with somewhat different syntax because this time we're referring to the parent `Platform` model:

```
01 protected $_referenceMap = array (
02     'Platform' => array (
03         'columns'      => array('platform_id'),
04         'refTableClass' => 'Platform'
05     )
06 );
```

In this snippet we're identifying the foreign keys stored in the `Game` model, identifying both the column storing the foreign key (`platform_id`), and the model that foreign key represents (`Platform`). Of course, it's entirely likely for a model to store multiple foreign keys. For instance, GameNomad's `User` model refers to three other models (`State`, `Country`, and `Platform`):

```
protected $_referenceMap = array (
    'States' => array (
        'columns'      => array('state_id'),
        'refTableClass' => 'State'
    ),
    'Countries' => array (
        'columns'      => array('country_id'),
        'refTableClass' => 'Country'
    ),
    'Platforms' => array (
        'columns'      => array('favorite_platform'),
        'refTableClass' => 'Platform'
    )
);
```

With this snippet in mind, returning to the magic methods, what methods would be at your disposal when attempting to retrieve a list of all users belonging to the state of Ohio? Ohio belongs to the `User` model's parent `State` model, so we're looking for dependent rows, meaning we have these methods available:

```
$ohio->findDependentRowset('User');
$ohio->findUser();
$ohio->findUserByState();
```

Likewise, the `User` model can refer to the parent `State` model like so:

```
$user->findParentRow('State')->name;
```

For example, to retrieve a count of users associated with the state identified by the primary ID 35, use this snippet:

```
$state = new State();
```

```
$sv = $state->find(35)->current();
$users = $sv->findDependentRowset('User');
echo count($users);
```

NOTE. The Zend_Db component can also automatically perform cascading operations if your database does not support referential integrity (for instance, MySQL's MyISAM storage engine does not). This means you can configure your website model to automatically remove all games associated with the Playstation 2 platform should you decide to quit supporting this platform and delete it from the platforms table. See the Zend Framework documentation for more information.

Step #8. JOINing Your Data

I'm a big fan of ORM solutions because they effectively abstract the gory SQL syntax that I've grown to despise over the years. But being able to avoid the syntax doesn't mean you should be altogether ignorant of it. In fact, ultimately you're going to need to understand some of SQL's finer points in order to maximize its capabilities. This is no more evident than when you need to retrieve data residing within multiple tables, a technique known as *joining* tables together.

There are numerous types of joins, and in fact entire chapters have been devoted to the topic. Rather than exhaustively (not to mention monotonously) introduce each, I'd like to instead guide you through various scenarios, introducing join queries alongside their practical application.

Join Scenarios

In this section you'll learn how to construct SQL joins by examining their application within various parts of the GameNomad website. If you'd like to experiment with each join (recommended), you can do so using phpMyAdmin's SQL interface, or using MySQL's command-line client.

Finding a User's Friends

The typical social networking website offers a means for examining a user's list of friends. GameNomad is no different, using a table called friends to manage these relationships:

```
CREATE TABLE friends (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    user_id INTEGER UNSIGNED NOT NULL,
    friend_id INTEGER UNSIGNED NOT NULL,
    friends_since TIMESTAMP NOT NULL
);
```

Let's begin by examining the most basic type of join, known as the inner join. An inner join will return the desired rows whenever there is at least one match in both tables, the match being determined by a shared value such as a user's primary key. So for example, you might use a join to retrieve a list of a particular user's friends:

```
mysql>SELECT u.handle FROM users u
->INNER JOIN friends f ON f.friend_id = u.id WHERE f.user_id = 44;
```

This join requests the handle of each user found in the `friends` table who is mapped to a friend of the user identified by 44.

Determine the Number of Copies of a Game Found in Your Network

Suppose you would like to borrow a particular game, but knew your best friend John had already loaned his copy to Carli. Chances are however somebody else in your network owns the game, but how can you know? Using a simple join, it's possible to determine the number of copies owned by friends, a feature integrated into GameNomad and shown in Figure 6-1.



Figure 6-1. Determining the number of copies of a game within a user's network

You might notice in Figure 6-1 this feature is actually used twice; once to determine the number of copies found in your network, and a second time to determine the number of copies found in your network which are identified as being available to borrow. To perform the former task, use this SQL join:

```
mysql>SELECT COUNT(gu.id) FROM games_users gu
->INNER JOIN friends f ON f.friend_id = gu.user_id
->WHERE f.user_id = 1 AND gu.game_id = 3;
```

As an exercise, try modifying this query to determine how many copies are available to borrow.

Determining Which Games Have Not Been Categorized by Platform

In an effort to increase the size of your site's gaming catalog, you've acquired another website which was dedicated to video game reviews. While the integration of this catalog has significantly bolstered the size of your database, the previous owner's lackadaisical data management practices left much to be desired, resulting in both incorrect and even missing platform assignments. To review a list of all video games and their corresponding platform (even if the platform is NULL), you can use a join variant known as a *left join*.

While the inner join will only return rows from both tables when a match is found within each, a left join will return all rows in the leftmost table found in the query even if no matching record is found in

the "right" table. Because we want to review a list of all video games and their corresponding platforms, even in cases where a platform hasn't been assigned, the left join serves as an ideal vehicle:

```
mysql>SELECT games.title, platforms.name FROM games
->LEFT JOIN platforms ON games.platform_id = platforms.id
->ORDER BY games.title LIMIT 10;
```

Executing this query produces results similar to the following:

| title | name |
|-------------------------------------|---------------|
| Ace Combat 4: Shattered Skies | Playstation 2 |
| Ace Combat 5 | Playstation 2 |
| Active Life Outdoor Challenge | Nintendo Wii |
| Advance Wars: Days of Ruin | Nintendo DS |
| American Girl Kit Mystery Challenge | Nintendo DS |
| Amplitude | Playstation 2 |
| Animal Crossing: Wild World | Nintendo DS |
| Animal Genius | Nintendo DS |
| Ant Bully | NULL |
| Atelier Iris Eternal Mana | Playstation 2 |

Note how the game "Ant Bully" has not been assigned a platform. Using an inner join, this row would not have appeared in the listing.

Counting Users by State

As your site grows in terms of registered users, chances are you'll want to create a few tools for analyzing statistical matters such as the geographical distribution of users according to state. To create a list tallying the number of registered users according to state, you can use a *right join*, which will list every record found in the rightside table, even if no users are found in that state. The following example demonstrates the join syntax used to perform this calculation:

```
mysql>SELECT COUNT(users.id), states.name
->FROM users RIGHT JOIN states ON users.state_id = states.id
->GROUP BY states.name;
```

Executing this query produces output similar to the following:

| | | | |
|-----|--|----------------|--|
| 145 | | New York | |
| 18 | | North Carolina | |
| 0 | | North Dakota | |
| 43 | | Ohio | |
| 22 | | Oklahoma | |
| 15 | | Oregon | |
| 77 | | Pennsylvania | |

There's no doubt SQL joins take some getting used to, as even the simplest of examples tend to require some rather focused thinking. The best advice I can give you is to spend an afternoon experimenting with the tables and data, creating your own joins and reviewing the results.

Joins and Zend_Db

Now that you have some understanding of how joins work, let's move on to how the Zend_Db makes it possible to integrate joins into your website. To demonstrate this feature, consider the following join query, which retrieves a list of a particular user's (identified by the primary key 3) friends:

```
mysql>SELECT u.id, u.handle FROM users u
    ->JOIN friends ON friends.friend_id = u.id
    ->WHERE friends.user_id = 3;
```

I implemented this join within a method named `getFriends()` found in the `UserRow` model:

```
01 function getFriends()
02 {
03     $user = new User();
04     $query = $user->select();
05     $query->from(array('u' => 'users'), array('u.id', 'u.handle'));
06     $query->join(array('f' => 'friends'), 'f.friend_id = u.id', array());
07     $query->where('f.user_id = ?', $this->id);
08
09     $results = $user->fetchAll($query);
10
11 }
```

Let's break this down:

Line 05 identifies the left side of the join, in this case the `users` table. You'll also want to pass along an array containing the columns which are to be selected, otherwise all column will by default be selected.

- Line 06 identifies the joined table, and join condition. If you'd like to select specific columns from the joined table, pass those columns along in an array as was done in line 05; otherwise pass in an empty array to select no columns.
- Line 07 defines a `WHERE` clause, which will restrict the result set to a specific set of rows. In this case, we only want rows in which the `friends` table's `user_id` column is set to the value identified by `$this->id`.

You'll come to find the Zend_Db's join capabilities are particularly useful as your website grows in complexity. When coupled with Zend_Db's relationship features (see Step #7), it's possible to create impressively powerful data mining features with very little code.

Step #9. Paginating Results with Zend_Paginator

For reasons of performance and organization, chances are you're going to want to spread returned database results across several pages if a lengthy number are returned. However, doing so manually can be a tedious chore, requiring you to track things such as the number of results per page, the page number, and the current offset of the overarching query result. Recognizing this importance of such a feature, the Zend Framework developers created the Zend_Paginator component, giving developers an easy way to paginate result sets without having to deal with all of the gory details otherwise involved in a custom implementation.

The Zend_Paginator component is quite adept, capable of paginating not only arrays, but also database results. It will also autonomously manage the number of results returned per page and the number of pages comprising the result set. In fact, Zend_Paginator will even create a formatted page navigator which you can insert at an appropriate location within the results page (see Figure 6-2).



Figure 6-2. Zend_Paginator will automatically segment results across multiple pages

In this section I'll show you how to paginate a large set of video games across multiple pages.

Update Your config.ini File

Begin by updating your `config.ini` file to specify the number of results you'd like returned per page. Of course, if you're planning on working with multiple sets of result types and want to adjust the number of results output per page, feel free to create additional configuration parameters.

```
; Pagination
pagination.item.count.per.page = 10
```

Create the Pagination Query

Next you'll want to add the pagination feature to your website. What's particularly nice about the Zend_Paginator component is that it can be easily integrated into an existing query (which was presumably previously returning all results). All you need to do is instantiate a new instance of the Zend_Paginator class, passing the query to the object, and Zend_Paginator will do the rest. The following script demonstrates this feature:

```
01 function getGamesByPlatform($id, $page=1, $order="title")
02 {
03     $query = $this->select();
04     $query->where('platform_id = ?', $id);
05     $query->order($order);
06
07     $paginator = new Zend_Paginator(new Zend_Paginator_Adapter_
08                                     DbTableSelect($query));
09     $paginator->setItemCountPerPage($config->pagination->item->count->per->page);
```

```

09 $paginator->setCurrentPageNumber($page);
10 return $paginator;
11 }

```

Let's break down this method:

- Lines 03-05 create the query whose results will be paginated. Because the method's purpose is to retrieve a set of video games identified according to a specific platform (Xbox 360 or Playstation 3 for instance), the query accepts a platform ID (`$id`) as a parameter. Further, should the developer wish to order the results according to a specific column, he can pass the column name along using the `$order` parameter.
- Line 07 creates the paginator object. When creating this object, you're going to pass along one of several available adapters. For instance, the `Zend_Paginator_Adapter_Array()` tells the Paginator we'll be paginating an array. In this example, we use `Zend_Paginator_Adapter_DbTableSelect()`, because we're paginating results which have been returned as instances of `Zend_Db_Table_Rowset_Abstract`. When using `Zend_Paginator_Adapter_DbTableSelect()`, you'll pass in the query.
- Line 08 determines the number of results which should be returned per page.
- Line 09 sets the current page number. This will of course adjust according to the page currently being viewed by the user. In a moment I'll show you how to detect the current page number.
- Line 10 returns the paginated result set, adjusted according to the number of results per page, and the offset according to our current page.

Using the Pagination Query

When using Zend_Paginator, each page of returned results will be displayed using the same controller and view. Zend_Paginator knows which page to return thanks to a `page` parameter which is passed along via the URL. For instance, the URL representing the first page of results would look like this:

```
http://gamenomad.com/games/platform/id/xbox360
```

The URL representing the fourth page of results would typically look like this:

```
http://gamenomad.com/games/platform/id/xbox360/page/4
```

Although I'll formally introduce this matter of retrieving URL parameters in the next chapter, there's nothing wrong with letting the cat out of the bag now, so to speak. The Zend Framework looks at URLs using the following pattern:

```
http://www.example.com/:controller/:action/:key/:value/:key/:value/.../:key/value
```

This means following the controller and action you can attach parameter keys and corresponding values, subsequently retrieving these values according to their key names within the action. So for

instance, in the previous URL the keys are `id` and `page`, and their corresponding values are `xbox360` and `4`, respectively. You can retrieve these values within your controller action using the following commands:

```
$platform = $this->_request->getParam('id');
$page = $this->_request->getParam('page');
```

What's more, using a feature known as *custom routes*, you can tell the framework to recognize URL parameters merely according to their location, thereby negating the need to even preface each value with a key name. For instance, if you head over to GameNomad you'll see the platform-specific game listings actually use URLs like this:

```
http://gamenomad.com/games/platform/xbox360/4
```

Although not required knowledge to make the most of the Zend Framework, creating custom routes is extremely easy to do, and once you figure them out you'll wonder how you ever got along without them. Head over to <http://framework.zend.com/manual/en/zend.controller.router.html> to learn more about them.

Step #10. Creating and Managing Views

You've seen how separating the three tiers (Model, View, and Controller) can make your life significantly easier in terms of website creation and manageability. This particular chapter has so far focused on the Model as it relates to Zend_Db, along the way creating some fairly sophisticated SQL queries. However there's still further you can go in terms of separating the database from the application code.

Most relational databases offer a feature known as a *named view*, which you can think of as a simple way to refer to a typically complex query. This query might involve retrieving data from numerous tables, and may evolve over time, sometimes by the hand of an experienced database administrator. By moving the query into the database and providing the developer with a simple alias for referring to the query, the administrator can separately manage and evolve that query without having to change any code lying within the application. Even if you're a solo developer charged with both managing the code and the database, views are nonetheless a great way to separate these sorts of concerns.

Creating a View

One of the first views I created used to gather the data used to create the list of the most popular games found in GameNomad according to their current Amazon.com sales ranks (<http://gamenomad.com/games/ranks>). Believe it or not, the query used to retrieve this data is fairly involved:

```
mysql>SELECT MAX(ranks.id) AS id, games.title AS title, games.asin AS asin,
->games.platform_id
->AS platform_id, ranks.rank AS rank FROM (games JOIN ranks
->ON((games.id = ranks.game_id)))
->GROUP BY ranks.game_id
->ORDER BY ranks.rank LIMIT 100;
```

Although by no means the most complex of queries, it's nonetheless a mouthful. Wouldn't it be much more straightforward if we can simply call this query using the following alias:

```
mysql>SELECT view_latest_sales_ranks;
```

Using MySQL's view feature, you can do exactly this! To create the view, login to MySQL using the MySQL client and execute the following command:

```
mysql>CREATE VIEW view_latest_sales_ranks AS
->SELECT MAX(ranks.id) AS id, games.title AS title, games.asin AS asin,
->games.platform_id
->AS platform_id, ranks.rank AS rank FROM (games JOIN ranks
->ON((games.id = ranks.game_id)))
->GROUP BY ranks.game_id
->ORDER BY ranks.rank LIMIT 100;
```

Alternatively, you can log into phpMyAdmin, navigate to the SQL tab, and paste this code into the text box (removing the mysql> and -> prefixes).

TIP. View creation statements are not automatically updated to reflect any structural or naming changes you make to the view's underlying tables and columns. Therefore if you make any changes to the tables or columns used by the view which reflect the view's SQL syntax, you'll need to modify the view accordingly. Modifying a view is demonstrated in the section "Reviewing View Creation Syntax".

Adding the View to the Zend Framework

The Zend Framework recognizes views as it would any other database table, meaning you can build a model around it!

```
<?php

class ViewLatestSalesRanks extends Zend_Db_Table_Abstract
{
    protected $_name = 'latest_sales_ranks';
    protected $_primary = 'id';

    protected $_referenceMap = array (
        'Platforms' => array (
            'columns'      => array('platform_id'),
            'refTableClass' => 'Platform'
        )
    );
}

?>
```

Deleting a View

Should you no longer require a view, consider removing it from the database for organizational reasons. To do so, use the `DROP VIEW` statement:

```
mysql>DROP VIEW latest_sales_ranks;
```

Reviewing View Creation Syntax

You'll often want to make modifications to a view over its lifetime. For instance, when I first created the `view_latest_sales_ranks` view, I neglected to limit the results to the top 100 games, resulting in a list of the top 369 games being generated. But recalling the view's lengthy SQL statement isn't easy, so how can you easily retrieve the current syntax for modification? The `SHOW CREATE VIEW` statement solves this dilemma nicely:

```
mysql>SHOW CREATE VIEW latest_sales_ranks\G
View: latest_sales_ranks
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `latest_sales_ranks` AS select max(`ranks`.`id`) AS `id`, `games`.`title` AS `title`, `games`.`asin` AS `asin`, `games`.`platform_id` AS `platform_id`, `ranks`.`rank` AS `rank` from (`games` join `ranks` on(`games`.`id` = `ranks`.`game_id`)) group by `ranks`.`game_id` order by `ranks`.`rank`
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

We're particularly interested in the three lines beginning with ``latest_sales_ranks``, as this signifies the start of the query. It looks different from the original SQL statement because MySQL takes care to delimit all table and column names using backticks to account for special characters or reserved words. You can however reuse this syntax when modifying the query so copy those lines to your clipboard. Next, remove the query using `DROP VIEW`:

```
mysql>DROP VIEW latest_sales_ranks;
```

Now recreate the view, modifying the syntax by adding `LIMIT 100` to the end of the query:

```
mysql>CREATE VIEW latest_sales_ranks `latest_sales_ranks` AS select max(`ranks`.`id`) AS `id`, `games`.`title` AS `title`, `games`.`asin` AS `asin`, `games`.`platform_id` AS `platform_id`, `ranks`.`rank` AS `rank` from (`games` join `ranks` on(`games`.`id` = `ranks`.`game_id`)) group by `ranks`.`game_id` order by `ranks`.`rank` ASC LIMIT 100;
```

Conclusion

Writing this chapter was particularly exciting, because it demonstrates the Zend Framework's ability to remove many data-related obstacles which have long fallen into the paths of many a developer.

In the next chapter, I'll introduce you to another equally important topic; the framework's form-related and e-mail features.

CHAPTER 7

Processing Forms and Sending Email

In Chapter 4 you learned how to process user-submitted forms data, thereby fulfilling a crucial requirement towards becoming a capable Web developer. Given the importance of being able to capture user input whether it's simply to facilitate his ability to contact you, or to manage far more complex processes such as managing the status of a user's video game collection, it isn't surprising that the Zend Framework comes packaged with fairly sophisticated forms processing features. In this chapter you'll learn all about these features, recreating GameNomad's contact and registration forms in this process.

Chapter Steps

The goals of this chapter are accomplished in six steps:

- **Step #1. Zend Framework Forms Fundamentals:** In this opening step you'll learn how the Zend Framework recognizes forms variables as they're passed from the form back to the server.
- **Step #2. Creating the GameNomad Contact Form:** In this step we'll put what you learned in Step #1 into action by recreating GameNomad's contact form.
- **Step #3. Validating Forms Data:** In Chapter 4 we built a validation library for ensuring user input is submitted in an appropriate format. The `Zend_Validate` component removes the need for you to maintain a custom library by providing a wide array of validation features. In this step you'll learn how to use `Zend_Validate` to shore up the contact form, ensuring the user provides complete and syntactically correct information.
- **Step #4. E-mailing User Input Using Zend_Mail:** We need a means for transmitting the user input to your e-mail inbox, and in this step you'll learn how to use the Zend Framework's `Zend_Mail` component to fulfill that role.
- **Step #5. Filtering Forms Data:** While validation ensures input is submitted in the desired format, sometimes you may want to massage the data without necessarily declaring it invalid. For instance, if a user provides a valid e-mail address inadvertently followed by a few blank spaces, you'll want to remove those blank spaces before storing the address in the database. In this step you'll learn how to use `Zend_Filter` to perform these sorts of normalization actions.
- **Step #6: Confounding Spammers with CAPTCHAs:** In their quest to spread the word about lottery winnings and weight-loss supplements, spammers have created programs for automating the completion and transmission of Web forms. Thankfully you can use a tool known as a *CAPTCHA* to thwart these programs from exploiting your forms to flood your inbox.

NOTE. Presuming you've been nosing around the Zend Framework documentation, you probably came across the `Zend_Form` component, which makes it possible to programmatically create and validate forms using PHP syntax. While I suggest taking some time to familiarize yourself with this component, I do not discuss it in this book for reason that I feel it violates one of the central tenets of an MVC framework: namely the unnecessary intermingling of logic and presentation. Ultimately though, one of the most appealing features of PHP is there's usually more than one way to do things, therefore if you're more comfortable creating forms programmatically, consider investigating this powerful component.

Step #1. Zend Framework Forms Fundamentals

The Zend Framework's approach to processing forms data really doesn't vary in terms of its conceptual approach from the principles you learned in Chapter 4. The main difference is it simply employs its own syntax for retrieving the POST values. For instance, to retrieve a POSTed form value named `last_name`, you would use the `getPost()` method like so:

```
$lastName = $this->_request->getPost('last_name');
```

Of course, in some cases the form value might be optional and therefore no corresponding POST value would be passed if the user left it blank, so you might first verify the value's existence using the `isSet()` method:

```
if (isSet($this->_request->getPost('last_name'))) {
    // do something
}
```

If some of the related form data is appended to the URL, you can use the `getQuery()` method to retrieve it. For instance, if a game's Amazon ASIN (Amazon Standard Identification Number) is appended to the form's action address (example: <http://www.gamenomad.com/games/show/asin/B0016B28Y8>), you could retrieve it like so:

```
$asin = $this->_request->getQuery('asin');
```

Determining if a Form Has Been Submitted

Because you'll typically use a single controller method for both displaying the form information and processing any changes made through it, you'll want to determine whether the form has been submitted before attempting to process its contents. This is done using the `isPost()` method:

```
// Has the form been submitted?
if ($this->_request->isPost()) {
    // Process form contents
}
```

Step #2. Creating the GameNomad Contact Form

The contact form provides your user with a simple way to contact you, while ensuring they provide all of the information you require to ensure a timely and accurate response. For instance, if you were building a website which managed magazine subscriptions, you might want users to include the subscription number found on the magazine label when contacting you for support. A contact form can force the user to include this, thereby saving the hassle of your support team to otherwise ask for it if the user contacted you directly via e-mail.

The GameNomad contact form (<http://www.gamenomad.com/about/contact>) is less rigid in its requirements, asking the user to include only his name and e-mail address along with the message. An additional feature is included allowing the user to CC his e-mail address on the message should he desire a record of the communication. Figure 7-1 presents the contact form.

Contact GameNomad

Have a question about GameNomad? Registration problems? Want to contact us regarding advertising opportunities? This is the place to do it! Just fill out the below form, and we'll get back to you as soon as possible.

Your Name:

Your E-mail Address:

Your Message:

Send me a copy of the message

Send your message!

Figure 7-1. The GameNomad contact form

In this initial step we'll just post back the form contents in order to get familiar with the Zend Framework's form parameter retrieval syntax. The view used to display the user's input and accompanying form is presented next:

```

01 <?php if (isset($this->posted)) { ?>
02
03     <p>
04         Your name is: <?php echo $this->escape($this->name); ?>
05     </p>
06
07     <p>
```

```

08     Your e-mail address is: <?php echo $this->escape($this->e-mail); ?>
09     </p>
10
11     <p>
12         Your message:<br />
13         <?php echo $this->escape($this->message); ?>
14     </p>
15
16 <?php } ?>
17
18 <form action="/about/contact" method="post">
19     <p>
20         <label for="name">Your Name</label>:<br />
21         <input id="name" name="name" size="50" type="text" value="" />
22     </p>
23
24     <p>
25         <label for="email">Your E-mail Address</label>:<br />
26         <input id="email" name="email" size="50" type="text" value="" />
27     </p>
28
29     <p>
30         <label for="message">Your Message</label>:<br />
31         <textarea id="message" name="message" cols="65" rows="15"></textarea>
32     </p>
33
34     <p>
35         <input type="checkbox" id="cc" name="cc" value="1" /> Send me a copy
            of the message
36     </p>
37
38     <p>
39         <input name="commit" id="submit" type="submit"
            value="Send your message!" />
40     </p>
41 </form>
```

Some commentary follows:

- Lines 18-41 offers nothing out of the norm, presenting the contact form. However, take a look at lines 01-16, as these are responsible for echoing the user's input once the form has been submitted. We'll set the four variables found in these lines within the controller, doing so only if the previously introduced `isPost()` method returns TRUE.

Next up is a basic version of the contact action (found in the About controller), followed by the usual breakdown of significant lines:

```
01 public function contactAction()
02 {
03
04     // If the form has been submitted, process it
05     if ($this->getRequest()->isPost()) {
06
07         $this->view->posted = TRUE;
08         $this->view->name = $this->_request->getPost('name');
09         $this->view->email = $this->_request->getPost('email');
10         $cc = $this->_request->getPost('cc');
11         $this->view->cc = isset($cc) ? "YES" : "NO";
12         $this->view->message = $this->_request->getPost('message');
13
14     }
15
16 }
```

The review follows:

- Line 05 determines whether the form has been submitted. If so, lines 07-10 assign the posted data to several variables which will be made available to the view.
- Because the user can optionally request for the message to be cc'd to his address, line 11 uses the ternary operator to determine whether the POST variable `cc` has been set, and sets the corresponding view variable is assigned accordingly. Line 10 first sets the POST variable to a local variable, because of a constraint with the ternary operator in which the value of a function's return variable cannot be properly determined, requiring this additional step to be made.

Pretty simple, right? Figure 7-2 displays the view after some form data has been submitted.

Contact GameNomad

Your name is: Jason

Your e-mail address is: wj@wjgilmore.com

Should you be cc'd? YES

Your message:

This is a test message.

Have a question about GameNomad? Registration problems? Want to contact us regarding advertising opportunities? This is the place to do it! Just fill out the below form, and we'll get back to you as soon as possible.

Your Name:

Jason

Your E-mail Address:

wj@wjgilmore.com

Your Message:

This is a test message.

Send me a copy of the message

Send your message!

Figure 7-2. Displaying user input within the browser

Of course, this example is simply a proof-of-concept intended to familiarize you with how the Zend Framework accepts forms data. As you'll learn later in this chapter, you should never simply accept user-supplied data at face value. It should always be thoroughly validated before anything else is done with it. In the next step we'll expand the contact method to validate each piece of user-supplied data.

Step #3. Validating Form Data

In Chapter 4 we created a small validation library used to validate user-supplied data such as e-mail addresses, phone numbers, and dates. While useful, writing and maintaining these validators can be tedious, not to mention their portability is lacking when it comes to localization matters. For instance, the `isValidDate()` function found in our custom library works great for United States, but in Europe the day and month values are often positioned using the format DD-MM-YYYY, meaning this function would fail if the unmodified library was used in conjunction with European date format. This is only the tip of the iceberg when it comes to the challenges of maintaining the code responsible for such an important task.

Thankfully, the `Zend_Validate` component removes such an important responsibility from our plate

altogether by providing a unified set of validation classes capable of vetting data using any number of parameters, from whether the data consists solely of numerical characters to whether it's a valid e-mail address, to whether it is a valid barcode as defined by the Universal Product Code or European Article Number specifications. And if Zend_Validate doesn't offer a particular validation rule, you can easily extend the component to include your own custom class.

Let's take a look at an example. One of the most commonplace validation tasks is to make sure the user hasn't inadvertently left a required field blank. This is easily confirmed using Zend_Validate's NotEmpty class.

```
$validator = new Zend_Validate_NotEmpty();
if ($validator->isValid($this->request->getPost('name'))) {
    foreach($validator->getMessages() as $message) {
        echo "{$message}<br />";
    }
}
```

If the user submits the form with a blank name variable, the following message will be displayed:

Value is empty, but a non-empty value is required

Of course, this default error message does little to help the user understand the source of the error, particularly when multiple form fields are involved. Later in this step I'll show you how to modify these messages to offer much more user-friendly alternatives. For now, take some time to review Table 7-1 to get familiar with the other Zend_Validate validation classes at your disposal.

Table 7-1. Zend_Validate's Validation Classes

| Validator | Description |
|-----------|--|
| Alnum | Determines whether the supplied value contains only alphabetic and numeric characters. An optional flag tells Alnum to allow white space characters. |
| Alpha | Determines whether the supplied value contains only alphabetic characters. An optional flag tells Alpha to allow white space characters. |
| Barcode | Determines whether the supplied value is a valid barcode, in accordance with the Universal Product Code (UPC) or European Article Number (EAN) specifications. |
| Between | Determines whether the supplied value falls between designated minimum and maximum values. |
| Ccnum | Determines whether the supplied value is a valid credit card number, in accordance with the formula defined by the Luhn algorithm (http://en.wikipedia.org/wiki/Luhn_algorithm). |
| Date | Determines whether the supplied value is valid as defined by the format YYYY-MM-DD. You can set an optional locale parameter to change the format definition, and the optional format parameter to change the format. |
| Digits | Determines whether the supplied value consists solely of numeric characters. |

| | |
|--------------|--|
| EmailAddress | Determines whether the supplied value is a valid e-mail address. |
| Float | Determines whether the supplied value is a floating-point value (for instance 2.43 or 0.838299). |
| GreaterThan | Determines whether the supplied value is greater than a designated maximum value. |
| Hex | Determines whether the supplied value consists solely of hexadecimal characters (characters falling within the ranges 0-9 and A-F). |
| Hostname | Determines whether the supplied value is a valid hostname. Three types of hostnames are supported, including DNS hostnames (gamenomad.com), IP addresses (192.168.1.1), and local hostnames such as localhost. |
| InArray | Determines whether the supplied value is found within the supplied array. An optional strict parameter will cause InArray to also verify the supplied value's type matches that of any identical value in the array. |
| Int | Determines whether the supplied value is an integer. |
| Ip | Determines whether the supplied value is a valid IP address. |
| LessThan | Determines whether the supplied value is less than a designated minimum value. |
| NotEmpty | Determines whether the supplied value is blank. |
| Regex | Determines whether the supplied value meets a designated regular expression. |
| StringLength | Determines whether the supplied value consists of a number of characters no greater than a designated maximum number, and no less than a designated minimum number. |

Streamlining the Validation Syntax

Instantiating each validation class before using it can make for some lengthy code, particularly when working with complex forms. In the interests of producing the most streamlined code possible I prefer to use an alternative syntax by way of the `Zend_Validate` class' `is()` method. This method allows you to combine the steps of instantiating the class and using it into one single command, like so:

```
if (! Zend_Validate::is($this->_request->getPost('name'), 'NotEmpty')) {
    echo "Please provide your name.";
}
```

Customizing Error Messages

The two validation examples you've seen so far simply output the error message if validation of a particular field fails. This isn't a particularly appealing approach, notably because it defies one of the main tenets of the Zend Framework's MVC strategy: separating logic from content. Further, as you've seen, the default messages aren't exactly user-friendly. Let's address the latter issue first; you can overwrite the default message with one of your own using the `setMessage()` method, like so:

```
$validator = new Zend_Validate_NotEmpty();
$validator->setMessage("Please provide your name.");
```

```

if (! $validator->isValid($this->_request->getPost('name'))) {
    foreach($validator->getMessages() as $message) {
        echo "{$message}<br />";
    }
}

```

However, this approach presents its own problems. While you now have a custom message, there's no simple way to accrue these messages together when multiple fields are validated. Further, the act of having to use a `foreach` statement simply to access these custom messages seems to be a pretty tedious process.

That said, the easiest solution I've found to the problem of both configuring custom messages and aggregating all of the messages together for reason of passing them to the view is to simply ignore `Zend_Validate`'s messaging feature altogether and use a custom solution. This solution involves validating each field, and in the case validation fails, append a message to an array created specifically for the reason of aggregating these errors together for subsequent output in the view. The following code demonstrates this approach:

```

// Initialize the errors array
$this->view->errors = array();

// Valid name?
if (! Zend_Validate::is($this->_request->getPost('name'), 'NotEmpty')) {
    $this->view->errors[] = "Please include your name";
} // end valid name

// Valid email address?
if (! Zend_Validate::is($this->_request->getPost('email'), 'EmailAddress')) {
    $this->view->errors[] = "Please include your e-mail address.";
} // end valid email

// Valid message?
if (! Zend_Validate::is($this->_request->getPost('message'), 'NotEmpty')) {
    $this->view->errors[] = "Please include a message.";
} // end valid message

```

Because the respective error messages are bound into an array, you're now at liberty to format them however you please within the view! For instance, I generally use the following snippet to output errors:

```

<?php if (isset($this->errors)) { ?>
    <div id="errors">
        Errors were encountered while sending your message:<br />
        <ul>
            <?php foreach ($this->errors AS $error) { ?>
                <li><?= $error; ?></li>
            <?php endforeach; ?>
        </ul>
    </div>

```

```
<?php } ?>
</ul>
</div>
<?php } ?>
```

This produces output similar to that shown in Figure 7-3.

The screenshot shows a contact form titled "Contact GameNomad". The form includes fields for "Your Name", "Your E-mail Address", and "Your Message". A message box displays validation errors: "Errors were encountered while sending your message:" followed by three bullet points: "Please include your name", "Please include your e-mail address.", and "Please include a message.". At the bottom, there is a checkbox labeled "Send me a copy of the message" and a button labeled "Send your message!".

Contact GameNomad

Have a question about GameNomad? Registration problems? Want to contact us regarding advertising opportunities? This is the place to do it! Just fill out the below form, and we'll get back to you as soon as possible.

Errors were encountered while sending your message:

- Please include your name
- Please include your e-mail address.
- Please include a message.

Your Name:

Your E-mail Address:

Your Message:

Send me a copy of the message

Send your message!

Figure 7-3. Displaying error messages within the view

Chaining Validators Together

The previous validation examples all involved using a single validation class to validate a particular characteristic of the data. But what if you wanted to validate several characteristics? You could instantiate a separate validator for each characteristic, however Zend_Validate offers a feature known as *validator chaining* which cuts down on the logic otherwise required with the latter approach.

The following example validates a social security number by first verifying that the value consists solely of numbers, and then ensuring it's exactly nine characters in length. If either of these rules is broken, an appropriate error message is added to the \$errors array:

```
$validator = new Zend_Validate();
$validator->addValidator(new Zend_Validate_Digits())
    ->addValidator(new Zend_Validate_StringLength(9,9));

if (! $validator->isValid($this->_request->getPost('name'))) {
    $this->view->errors[] = "Please provide a valid SSN.";
}
```

Creating a Custom Validator

The Zend_Validate component helps you to validate a great deal of data, but it doesn't cover every conceivable situation. For instance, there's nothing available for determining the validity of U.S. phone numbers. Of course, the process is fairly simple; just strip out all non-numeric characters, and then ensure the string consists of exactly ten digits. But because validating phone numbers is such a commonplace task, wouldn't it be nicer if we could just call a class named GameNomad_Validate_Phone?

Creating custom validators is easily done by extending the framework's Zend_Validate_Abstract class. Remember that extending a class gives the new class all of the characteristics of the extended class, leaving you to just worry about the new functionality provided by the extending class. The GameNomad_Validate_Phone class is found next:

```
01 <?php
02
03     class GameNomad_Validate_Phone extends Zend_Validate_Abstract
04     {
05
06         public function isValid($value)
07         {
08
09             // Strip out all characters except the digits
10             $value = preg_replace('/[^0-9]/', '', $value);
11
12             // Determine the number of digits remaining in the string
13             if (strlen($value) == 10) {
14                 return TRUE;
15             } else {
16                 return FALSE;
17             }
18
19         }
20
21     }
22
23 ?>
```

Let's review the code:

- Line 03 extends `Zend_Validate_Abstract`, thereby giving `GameNomad_Validate_Phone` all of the characteristics granted to other classes found within the `Zend_Validate` component. Keep in mind this particular validator is a bare-bones version; you can also create custom messages which operate in the same manner as the messages provided by other validator classes.
- Lines 06-17 defines the `isValid()` method, which we'll subsequently use when calling the `GameNomad_Validate_Phone` class. This method performs two tasks; first line 10 strips all characters other than digits from the provided string. This means users can enter their phone number in a free-form manner, using any formatting variation they please. For instance, (555) 555-1212, (555) 555.1212 and 555-555-1212 would all be acceptable. Next, lines 13-17 determine whether 10 digits remain in the string. If so, the phone number is valid, otherwise it is not.

Place this class within the directory `library/GameNomad/Validate`, and name it `Phone.php`. This gives the Zend Framework the ability to autoload the class like any other. Now you're able to call the `GameNomad_Validate_Phone` class from any controller, like so:

```
$validator = new GameNomad_Validate_Phone();

if (! $validator->isValid('555-555-1212')) {
    $this->view->errors[] = "Invalid phone number!";
}
```

Step #4. E-mailing User Input Using Zend_Mail

You might recall from Chapter 4 that while generally a simple process, sending e-mail via a PHP script can occasionally be a tedious affair due to the platform-specific configuration issues. The `Mail` package introduced in that chapter did away with those problems by providing a unified, cross-platform solution which requires minimum changes when migration your application from one platform to another.

VIDEO. Sending E-mail Using PHP

Whether it's for confirming e-mail addresses, recovering passwords, or notifying users of responses to their forum posts, you'll inevitably need to use PHP to send e-mails from varying locations within your website. This video introduces the `Zend_Mail` component, showing you how to send e-mails for a variety of purposes. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

The `Zend_Mail` component (<http://framework.zend.com/manual/en/zend.mail.html>) offers its own solution to the configuration dilemma, while simultaneously facilitating the transmission of messages by offering a programmatic means for adding CC and BCC addresses, sending HTML-formatted mail, and including attachments.

`Zend_Mail` gives you the option of sending e-mail using either your server's Sendmail service (the default), or via SMTP. If you're relying on the former, then sending an e-mail is as easy as this:

```
$email = new Zend_Mail();
$email->setFrom('admin@example.com', 'Example Administrator');
$email->addTo('client@example.net', 'Joe Q. Client');
$email->setSubject('Your account has been created');
$email->setBodyText('Your account has been created! We hope you enjoy the
                     service.');
$email->send();
```

Other methods are available for CC'ing and BCC'ing users, including `addCc()` and `addBcc()`, respectively.

Changing the Transport

If you plan on using an SMTP address, you'll need to configure `Zend_Mail` to use the address. This is done by way of the `Zend_Mail_Transport_Smtp` class. For instance, the following will configure `Zend_Mail` to send e-mail through my Gmail account:

```
$config = array('auth' => 'login',
                'username' => 'wgilmore@gmail.com',
                'password' => 'secret',
                'ssl' => 'tls');

$transport = new Zend_Mail_Transport_Smtp('smtp.gmail.com', $config);
Zend_Mail::setDefaultTransport($transport);
```

Because you'll typically want to send e-mails from various locations within the application, not to mention don't want to mix configuration information with your logic, it makes sense to place this code in the application's `bootstrap.php` file. See Chapter 5 for more information about this file.

NOTE. If you plan on using Gmail or Google's e-mail hosting service, you'll need to enable PHP's OpenSSL extension.

Creating E-mail Templates

Line 05 of the previous example showed you how to assign body text to an e-mail sent through `Zend_Mail`. However in practicality, your messages will be far larger. So how do you maintain the message text? I prefer to place the message text in a view partial, storing it within the `views/scripts` directory (see Chapter 8 for an introduction to view partials). The following script (`_e-mail-confirm-registration.phtml`) demonstrates this practice:

```
<?php

$email = <<< email
Dear {$firstName},

Welcome to the GameNomad community! To confirm your e-mail address, click on
the following URL:
```

```
{$this->config->website->url}/gamers/verify/key/{$registrationKey}
```

Questions? Comments? Contact us at {\$this->config->email->support}!

Thank you,
The GameNomad Team
email;

?>

Following the insertion of the user information into the database, the e-mail is prepared and sent (located within the `registerAction()` method of the `Gamers` controller):

```
01 try {
02     // Create a new mail object
03     $mail = new Zend_Mail();
04
05     // Set the From, To, and Subject headers
06     $mail->setFrom($this->config->email->from_admin);
07     $mail->addTo($this->_request->getPost('email'),
08                  "{$this->_request->getPost('first_name')}"
09                  "{$this->_request->getPost('first_name')}");
10     $mail->setSubject('Your GameNomad account has been created');
11
12     // Retrieve the e-mail template
13     require '_email-confirm-registration.phtml';
14
15     // Attach the e-mail template to the e-mail and send it
16     $mail->setBodyText($email);
17     $mail->send();
18
19     $this->view->success = 1;
20 } catch (Exception $e) {
21     $this->view->errors[] = "We were unable to send your confirmation e-mail.
22                             Please contact {$this->config->email->support}.";
23 }
```

Breaking down the code:

- Nothing should be new here, other than the inclusion of the e-mail template (line 13). Once included, the `$email` variable and its contents are made available to the scope of this script, meaning any variables found within the template are also parsed. The variable's contents are then passed to the `setBodyText()` method (line 16), and the e-mail is sent (line 17).

Including Attachments

Suppose your website started offering analysis of the gaming industry, and you wanted to provide subscribers with PDF-formatted reports which allow for easy printing. Each time a report is pub-

lished, you could post the report to your website, and send subscribers an e-mail containing the link, or you could save subscribers a step by simply attaching the PDF to each e-mail. Thankfully, `Zend_Mail` makes it easy to add an attachment to the e-mail; just pass the file to `Zend_Mail`'s `createAttachment()` method:

```
$email->createAttachment("/home/gamenomad/reports/report.pdf");
```

`Zend_Mail` will by default assume the attachment is a binary object (`application/octet-stream`), however if you want to override this assumption you can change the MIME type, encoding, and other attributes. See the Zend Framework documentation for more information about these options if you need to tweak these default settings.

Sending HTML-Formatted E-mail

I've long argued that sending HTML-formatted e-mail is like entering a hot dog eating contest. It's possible to eat 50 ballpark franks, but simply not necessary. Nonetheless, many marketing departments seem to be in love with this twisted form of communication, and so it seems a topic worthy of inclusion here.

`Zend_Mail` makes sending HTML-formatted e-mail a breeze; all you need to do is use the `setBodyHtml()` method in place of `setBodyText()`. Doing so will cause `Zend_Mail` to automatically configure the message using the `text/HTML` MIME setting, meaning the recipient's e-mail client will understand the e-mail message is intended to be read in HTML format, causing the client to format it accordingly.

Drawing on the strategy used in the earlier section, "Creating E-mail Templates", the following example presents a sample HTML message which would then be injected into the sample newsletter action. Figure 7-4 presents that message as it's rendered within the Microsoft Outlook e-mail client.

```
01 $email = <<< email
02 <body>
03 <style type="text/css">
04 #logo {
05     width: 100%;
06     background-color: #CC7610;
07     background-image: url("http://www.gamenomad.com/images/grunge-bg.png");
08     background-repeat: repeat-x;
09     padding: 3px; /* pad the content a little */
10    margin: 0px auto; /* this centers the container */
11    height: 95px;
12    border-bottom: 1px solid black;
13 }
14 </style>
15 <div id="logo">
16   <a href="http://www.gamenomad.com/">
17     </a>
18 </div>
```

```

19 <p>
20 Dear Jason,
21 </p>
22 <p>
23 Welcome to the GameNomad community! To confirm your e-mail address, click
on the following URL:
24 </p>
25
26 <p>
27 http://www.gamenomad.com/gamers/verify/key/opgktu6qaip3jjq2e20qzqs2zwzndf5f
28 </p>
29
30 <p>Questions? Comments? Contact us at support@gamenomad.com!</p>
31
32 <p>Thank you,<br />
33 The GameNomad Team
34 </p>
35 </p>
36
37   </body>
38 email;

```

One code-related note follows:

- Most e-mail clients will strip everything prior to and following the `<body>` tag enclosure, meaning if you want to use CSS styling within your e-mail, you'll need to include the styles within the `<style>` tag following the `<body>` tag, as was done in lines 03-14. Keep in mind that some e-mail clients will downright remove anything found within `<style>` tags, altogether removing any possibility of formatting e-mail using CSS.

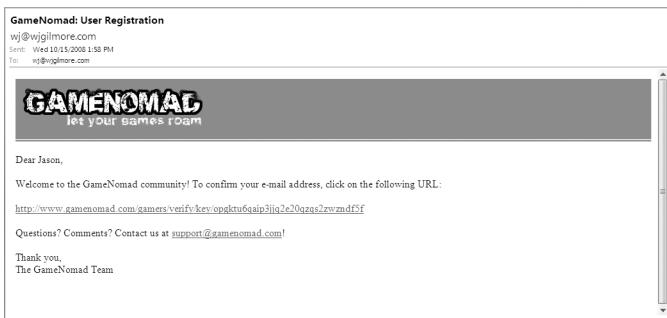


Figure 7-4. HTML-formatted e-mail rendered within Microsoft Outlook

The opening comments of this section probably betray my personal bias against HTML-formatted e-mail, however the decision is ultimately yours. If you do choose to deliver HTML-formatted e-mail to your users, at a minimum I suggest doing ample research to ensure the e-mail is properly formatted within all major e-mail clients. One particularly useful resource is a study published by Campaign Monitor (<http://www.campaignmonitor.com/>) which thoroughly summarizes CSS support variations within popular clients. You can read the study here: <http://www.campaignmonitor.com/css/>.

Alternatively, consider taking advantage of services such as Campaign Monitor to do the dirty work for you, if you plan to depend heavily upon HTML-formatted e-mails as part of your website's service offerings.

Step #5. Filtering Form Data

The Zend_Validate component serves an important role in that it's intended to halt the completion of a task should the supplied data not meet a set of strictly defined constraints. But there's a fine line separating validating user input and being so tedious that the usability of your application is hindered due to overbearing standards. For instance, what if the user mistakenly pressed the tab key after entering an otherwise valid e-mail address into a newsletter subscription form? While you definitely do not want to store the hidden tab sequence in the database, this doesn't seem to be a sufficient reason to halt the newsletter subscription process. Instead, you should *filter* the data, silently removing the tab sequence without informing the user of this change.

To perform this and other related tasks, you can use the Zend_Filter component. This component includes several classes which will prove useful when filtering data of all sorts. Take a moment to peruse these classes, presented in Table 7-2.

Table 7-2. The Zend_Filter Classes

| Class Name | Description |
|---------------|--|
| Alnum | Removes all characters but those qualified as alphanumeric. An optional flag tells Alnum to allow white space characters. For instance, the string <i>*GameNomad* Rules!</i> will become <i>GameNomad Rules</i> . |
| Alpha | Removes all characters but those qualified as alphabetical. An optional flag tells Alpha to allow white space characters. For instance, the string <i>GameNomad 2008</i> will become <i>GameNomad</i> (note the space between <i>GameNomad</i> and <i>2008</i> was also removed on the assumption the white space flag wasn't passed). |
| BaseName | Removes any preceding path information from a string concluding with a filename. For instance, <i>/home/gamenomad.com/logo.png</i> becomes <i>logo.png</i> . |
| Digits | Removes all characters but those qualified as digits. For instance, the string <i>GameNomad 2008</i> will become <i>2008</i> (note the space between <i>GameNomad</i> and <i>2008</i> was also removed on the assumption the white space flag wasn't passed). |
| Dir | Removes any concluding filename from a string consisting of a path and file. For instance, <i>/home/gamenomad.com/logo.png</i> becomes <i>/home/gamenomad.com/</i> . |
| HtmlEntities | Converts any characters to the corresponding HTML entity. For instance, <i>2008 > 2007</i> becomes <i>2008 &gt; 2007</i> . |
| Int | Converts the string to an integer. For instance, <i>10 days</i> becomes <i>10</i> . |
| StripNewLines | Removes any newline sequences from the string. |

| | |
|---------------|--|
| RealPath | Converts symbolic links and relative paths to the absolute equivalent. |
| StringToLower | Converts the string to its lowercase equivalent. For instance, <i>GameNomad</i> becomes <i>gamenomad</i> . |
| StringToUpper | Converts the string to its uppercase equivalent. For instance, <i>GameNomad</i> becomes <i>GAMENOMAD</i> . |
| StringTrim | Removes control sequences and whitespace from the beginning and end of the string. |
| StripTags | Removes HTML and PHP tags from the string. You can pass preapproved tags into the method. |

Using Zend_Filter is straightforward; For instance, to remove whitespace and other control sequences from the beginning and end of an e-mail address:

```
$filter = new Zend_Filter_StringTrim();
$email = $filter->filter($this->_request->getPost('email'));
```

Creating Filter Chains

Just as you can create validation chains, so can you create filter chains. For instance, to trim control sequences and whitespace, and also make sure only alphanumeric characters are found in the string, you would chain together the `StringTrim` and `Alnum` filters like so:

```
$filter = new Zend_Filter();
$filter->addFilter(new Zend_Filter_StringTrim())
    ->addFilter(new Zend_Filter_Alnum());
$name = $filter->filter($this->_request->getPost('name'));
```

TIP. The Zend Framework also offers a hybrid filter/validation mechanism called Zend_Filter_Input. This streamlines the validation and filtering process, saving you from writing some additional code when vetting user input. See the Zend Framework documentation for more information.

Step #6. Preventing Spamming Using CAPTCHAs

I use Google's corporate e-mail service offering for all of my company mail, and thanks to their aggressive spam filtering service, am rarely annoyed by unwanted advertisements. If you're inundated with spam, I highly recommend checking out their service (<http://www.google.com/a/>).

However, if you're already locked into an e-mail hosting solution or otherwise don't want to change, it might be worth seeking some additional spam protection by taking steps to prevent them from "contacting" you via your website's contact form. In recent years, the CAPTCHA (*Completely Automatic Public Turing test to tell Computers and Humans Apart*) has been the safeguard of choice, for one simple reason: despite all the claims of computers' pending takeover of the world, they remain inca-

pable of performing certain tasks which we humans handle with ease. The CAPTCHA exploits one such task, namely the computer's inability to understand pictures. By asking the user completing the form to confirm the alphanumeric contents of the picture, we can confirm with some accuracy the humanity of the user, and therefore declare the form data to be at least supplied by a human rather than an automated program. Figure 7-5 displays a screenshot of Google Groups' CAPTCHA mechanism.



Figure 7-5. The CAPTCHA mechanism used within Google Groups

So how are CAPTCHAs generated, and how do you compare the text found in the graphic with the user's confirmation? The Zend_Captcha component can manage your CAPTCHAs with minimal effort, not only creating the image but also validating the user input to ensure it is correctly echoed.

Zend_Captcha technically supports four mechanisms for producing challenges, however only two are of practical use:

- **Zend_Captcha_Figlet:** This adapter turns strings into text-based images known as FIGlets (<http://www.figlet.org/>). An example FIGlet is presented in Figure 7-6.
- **Zend_Captcha_Image:** This adapter converts text into binary images such as the one shown in Figure 7-5. It requires the GD extension (<http://www.php.net/gd>) be enabled.
- **Zend_Captcha_Recaptcha:** A ReCAPTCHA (<http://recaptcha.net/>) is an ingenious approach to transparently turning the human effort used to solve CAPTCHAs into a means for deciphering digitized text that a computer is otherwise unable to understand. This adapter plugs into the ReCAPTCHA service, retrieving images drawn from the ReCAPTCHA database and submitting the supplied results back to the service.



Figure 7-6. A FIGlet image generated with the Zend_Captcha_Figlet adapter

NOTE. A text-based CAPTCHA generator might seem a bit dated, however the minimalistic approach not only negates the need to install server-side extensions, but also client-side extensions otherwise required should a Flash-based CAPTCHA generator be used. Personally, I find FIGlets to be far easier to read than their image-based counterparts, as the latter can occasionally be formatted in an exceedingly distorted manner.

Creating and Validating CAPTCHAs

As with most of Zend Framework components, once you understand the syntax, using them is exceedingly easy. In fact, create a simple FIGlet can be done in as little as two lines:

```
$captcha = new Zend_Text_Figlet();
$this->view->captcha = $captcha->render('gamer');
```

The captcha view variable is subsequently output in a corresponding view, producing the image shown in Figure 7-6. Remember to output the image between `<pre>` tags, as the component uses a monospace font by default. If you're interested in another font format, consult the list of available alternatives at <http://www.figlet.org/>. Also, Zend_Text_Figlet supports a number of other options for setting the font size, height and width of the generated image, and other useful characteristics. Consult the Zend Framework documentation for more information about these options.

To create an image-based CAPTCHA, you'll use the Zend_Captcha_Image component. The most basic code is a tad more involved than that required to create a FIGlet, because unlike the FIGlet you need to reference a font, in addition to specifically generate the image:

```
$captcha = new Zend_Captcha_Image(array('wordLen' => 5));
$captcha->setFont("/usr/share/fonts/truetype/ttf-bitstream-vera/VeraBd.ttf");
$captcha->generate();
$this->view->captcha = $captcha->render($this->view);
```

Incorporating the `$captcha` view variable into the view, I used this code to create the image shown in Figure 7-7.

In order to verify you're human, please confirm the image's contents in the below form field:

x&maj

Enter contents here:

Submit Form

Figure 7-7. Creating an image-based CAPTCHA

You'll notice I passed the `wordLen` option into the `Zend_Captcha_Image` constructor, which is used to determine the number of characters subsequently shown in the CAPTCHA. This is just one of numerous available options, giving you among others the ability to define the image height and width. Also,

keep in mind you need to create the directory `images/captcha` (the `images` directory residing in your application's `public` directory), or alternatively use `Zend_Captcha_Image`'s `setImgDir()` method to explicitly specify an alternative directory.

Validating the CAPTCHA

Earlier I mentioned `Zend_Captcha` will handle the CAPTCHA validation for you. To validate a captcha, just tell `Zend_Captcha_Image` the name of the form variable which will be passed back to the controller when instantiating the class. Then when the form variable is passed back, use the `isValid()` method like you would any other validation class:

```
01 $captcha = new Zend_Captcha_Image(array('name' => 'gamer', 'wordLen' => 5));
02 $captcha->setFont("/usr/share/fonts/truetype/ttf-bitstream-vera/VeraBd.ttf");
03 $captcha->generate();
04 $this->view->captcha = $captcha->render($this->view);
05
06 if ($captcha->isValid($this->_request->getPost('gamer'))) {
07     echo "VALID!";
08 }
```

Conclusion

The impressive form creation, validation and filtering features offered by the Zend Framework leave you with no excuse for not properly vetting input arriving from external sources. Be sure to take advantage of these time-saving (and perhaps job-saving) features no matter the type of website. Likewise, the `Zend_Mail` component makes sending e-mail from your website a total breeze, replacing the tricky and error-prone strategies of the past.

The next chapter tackles another important topic: user management. In this chapter you'll learn how implement user registration and login mechanisms, along with requisite features such as password recovery.

CHAPTER 8

Managing Your User Community

One of the most compelling aspects of the Web is it's a two-way street. A website can disseminate information almost as easily as it can ask for it, greatly expanding the attractiveness of your website in the process by allowing users to manage profiles, control site preferences such as layout and content availability, and interact with other users. By providing similarly rich levels of interactivity to your users, you'll build brand loyalty, encourage users to tell their friends about the site, and be able to better respond to user needs by monitoring their behavior.

Of course, your website will require a means for tying these sorts of interactions back to a specific user. The standard process for doing so is by prompting a registered user to login to the site using a username and password. Once logged in, any actions the user undertakes are then associated with the account tied to the provided username and password.

As the developer, you'll need to create mechanisms for not only allowing the user to register, but also login, logout, and carry out various account-related tasks such password recovery and perhaps profile management. In this chapter you'll learn all about these tasks, building several of them with the help of the Zend_Auth framework component. You'll also learn how to create a facility for allowing users to build an onsite network by identifying certain other users as friends.

Chapter Steps

The goals of this chapter are accomplished in five steps:

- **Step #1. Creating the Users Table and Model:** Before anything can be done with user management, we'll need to create a database table and model used to manage this data. We'll kick off this chapter by creating this table and model.
- **Step #2. Registering Users:** Once the model has been created we'll want to begin populating it by providing users with an account registration form. This form will prompt users to create an account password while providing any other registration-related information you'd like to collect, such as the user's name and location. In this step we'll build upon what you learned in the last chapter by creating the registration form which collects this information. You'll also learn how to confirm registration by forcing the user to click on a link found in an e-mail sent to his account following submission of the registration form.
- **Step #3. Managing User Logins:** The Zend Framework's Zend_Auth component makes it easy to manage user logins, providing mechanisms for logging the user into the site, maintaining the user's session as he interacts with the site, and logging the user out of the site. In this step you'll learn how these features are implemented. You'll also learn how to create a password recovery feature so the user can autonomously reset his password in the event it is forgotten.
- **Step #4: Displaying User Profiles:** Chances are you'll want to display user profile information on the website, in addition to other site content he's compiled over time. In this section

you'll learn how to do this. For the purposes of demonstration this example will only include the user's name and last login date, but it will nonetheless serve as a basis for adding additional content. In fact, in the final step (discussed next) you'll learn how to integrate the user's friends list into this profile page.

- **Step #5. Making Friends:** In the final step of this chapter we'll create a mechanism for giving users the ability to identify other users as friends, thereby opening up the possibility for you to build features which allow friends to track the interactions of each other on the site, such as new additions to their game collection, or status updates regarding what game they're currently playing.

Step #1. Creating the Users Table and Model

Before creating any of the scripts used to power the aforementioned features, it makes sense to first spend some time designing the table and model used to store and manage the user data. Let's start by creating the `users` table, subsequently building the `Users` model based on the corresponding schema.

The Users Table

The `users` table will store each user's key account information, namely the username and password, profile-related information such as his name and gender, and data which will help us gather simple usage metrics, such as when the account was created, when it was last updated, and when the user last logged into the system. The `users` table follows:

```
CREATE TABLE users (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password CHAR(32) NOT NULL,
    registration_key CHAR(32) NOT NULL,
    confirmed TINYINT UNSIGNED NOT NULL DEFAULT 0,
    handle VARCHAR(32) NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    gender ENUM('m','f') NOT NULL,
    created_at DATETIME NOT NULL,
    updated_at DATETIME NOT NULL,
    last_login DATETIME NOT NULL
);
```

Given each relevant line of this listing can be easily identified by the column name rather than a line number, I'll break from the usual protocol and instead use these names to break down the listing:

- The `id` serves as the table's primary key which you learned all about in Chapter 4.
- The `email` will serve double-duty not only as the vehicle for contacting each user, but also as the account's username which the user will have to supply when logging into the system. Some sites ask the user to contrive a nickname (for instance, I typically use `wjgilmore`), how-

ever doing so requires the user to recall another piece of information when logging in. On the contrary, an e-mail address is almost certainly instantly recallable by the user, and therefore serves as an ideal username.

- The `password` is stored as a 32-character encrypted string, created using a one-way calculation based on the user's provided password. Storing the password in this fashion makes it impossible for an attacker to determine the original password text even if he somehow gets his hands on the table data. Each time the user logs in, we will encrypt the provided password and compare it to the stored encrypted string. You'll learn how this encryption process occurs in the section "Registering a New User Account".
- The `registration_key` column stores a random 32-character string used as an important part of the registration confirmation process. You'll learn more about the role of this column in the section "Confirming Registration".
- The `confirmed` column is set once the user has confirmed his e-mail address. You'll learn more about the role of this column in the section "Confirming Registration".
- The `handle` column is used to refer to the gamer on the site and among friends. Think of the handle as a fun nickname, much like a gamer handle used when playing the Xbox 360.
- The `first_name` and `last_name` columns are used as alternative ways to refer to the user when we need to do for more formal reasons, such as when sending administrative emails.
- The `gender` column identifies the user's gender, and is primarily used for referring to the proper possessive grammar when talking about for instance the user's game collection (e.g. *Jason has 14 games in his collection*), but is also useful for determining website user trends.
- The `created_at`, `updated_at`, and `last_login` columns are used to determine when the user's account was created, last updated, and last logged into, respectively.

With the `users` table created, it's time to create the data model we'll use to interact with it.

The Users Model

As is typical, the model used to represent the `users` table is actually broken into two parts: the `User` model, which is used for performing actions against the `users` table writ large, and the `UserRow` model, which is used for accessing and manipulating the `users` table at the row level. In this section we'll build both models, and subsequently use them in later sections.

Creating the User Model

The `Users` model, displayed in Listing 8-1, includes basic functionality for accessing the `users` table. Obviously as your website grows in size and complexity, so will the `Users` model, however what is presented here serves as a solid starting point. Take some time to review Listing 8-1 and thoroughly review the explanation that follows it.

Listing 8-1. The User Model

```
01 class User extends Zend_Db_Table
02 {
03
04     /**
05      * The actual name of the table
06      *
07      * @var string
08      */
09     protected $_name = 'users';
10
11    /**
12     * The table's primary key
13     *
14     * @var string
15     */
16     protected $_primary = 'id';
17
18    /**
19     * The User class' row model
20     *
21     */
22     protected $_rowClass = 'UserRow';
23
24    /**
25     * Retrieve a user using his email address
26     *
27     * @param string $email
28     * @return UserRow
29     */
30     function getUserByEmail($email)
31     {
32         $query = $this->select();
33         $query->where('email = ?', $email);
34         $result = $this->fetchRow($query);
35         return $result;
36     }
37
38    /**
39     * Retrieve user according to his gamer handle
40     *
41     * @param string $handle
42     * @return UserRow
43     */
44     function getUserByHandle($handle)
45     {
46         $query = $this->select();
47         $query->where('handle = ?', $handle);
48         $result = $this->fetchRow($query);
```

```

49         return $result;
50     }
51
52     /**
53      * Calculate the total number of registered users
54      *
55      * @return integer
56      */
57     function getUserCount()
58     {
59         $query = $this->select('id');
60         $result = $this->fetchAll($query);
61         return count($result);
62     }
63
64 }
```

The code review follows:

- Line 01 defines the model name, and extends the model from the `Zend_Db_Table` class. As mentioned in Chapter 6, I prefer to use single tense for model names (because we're referring to one table), and plural tense for table names (because the table manages multiple entities presumably of the same name, for instance the `users` table manages multiple `users`).
- In order to override the Zend Framework's default convention of presuming the model name matches the table name it represents, line 09 overrides the default and specifies the table name is `users` rather than `user`.
- Line 16 identifies the table's primary key.
- Line 22 identifies the model used to represent the rows of the table represented by the `User` model. We'll talk about this model next.
- Lines 30-36 define the `getUserByEmail()` method, which can be used to retrieve a user when all you have available is the user's e-mail address.
- Lines 44-50 define the `getUserByHandle()` method, which can be used to retrieve a user when all you have available is the user's e-mail address.
- Lines 57-62 define a useful method we'll use to determine the total number of users in the system. For instance, GameNomad uses this method to display the total number of registered users at the top right of each page.

Creating the UserRow Model

Once a user or group of users have been identified using the `User` model, you can begin performing row-specific operations using the `UserRow` model. Listing 8-2 presents a simple example of this model, complete with a simple method. Later in this chapter we'll add other methods to the model as

Listing 8-2. The UserRow model

```
01 class UserRow extends Zend_Db_Table_Row
02 {
03
04     /**
05      * Determines whether the user is male
06      *
07      * @return boolean
08      */
09     function isMale()
10    {
11         if ($this->gender == "m")
12        {
13            return TRUE;
14        } else {
15            return FALSE;
16        }
17    }
18
19 }
```

Let's breakdown some code:

- Line 01 defines the model name (`UserRow`). Note this inherits from `Zend_Db_Table_Row`. The Zend Framework will be able to determine the relation to the `User` model based on the `$_rowClass` assignment made in the `User` model.
- Lines 09-17 define the `isMale()` function. Note how we can refer to the user's attributes using `$this`, although keep in mind only those attributes which were made available by way of the query are available to the `UserRow` model.

Step #2. Registering Users

To create accounts, we need to provide users with an autonomous means for registering. This is typically done in two steps, the first providing the user with a form for creating the account, and the second requiring the user to confirm registration by clicking on an emailed link. In this section I'll show you how to carry out both steps.

Creating a New User Account

The registration process requires the user to complete a short form which will then be validated. If all provided data is proved valid, it will be inserted into the database and the second step (validation) will ensue.

Let's begin with a screenshot (Figure 8-1) of a typical registration form, which should serve to give you a visual idea of the data we're collecting:

Create a GameNomad Account!

To join the GameNomad community, just fill out the below form and click on the confirmation link which will arrive in your inbox within minutes.

Your Email Address:

Create a Gamer Handle: (letters and digits only)

Create a Password: (minimum six characters)

Your First Name:

Your Last Name:

Your Gender:

Figure 8-1. A user registration form

By this point in the book the form syntax should be pretty easy to figure out, so I'll move on to where the action's at (no pun intended), namely the Gamers controller's register action. This action is rather long, so rather than simply pasting in a listing which spans several pages I'll instead focus on two select sections and leave it to you to review the entire action located in the code download. Let's begin the review with Listing 8-2, which contains the code used to validate the form fields.

Listing 8-2. The register action's validation tasks

```

01 // If the form has been submitted, process it
02 if ($this->getRequest()->isPost()) {
03
04     // Valid email address?
05     if (! Zend_Validate::is($this->_request->getPost('email'),
06         'EmailAddress')) {
07         $this->view->errors[] = "Invalid e-mail address.";
08     } // end valid email
09
10    // E-mail cannot already exist in database
11    $user = new User();
12    $foundUser = $user->getUserByEmail($this->_request->getPost('email'));
13    if ($foundUser->id != "") {
14        $this->view->errors[] = "E-mail address already in database.";
15    }
16
17    // Handle must be between 2 and 20 characters
18    $validHandle = new Zend_Validate_StringLength(2,20);

```

```

18     if (! $validHandle->isValid($this->_request->getPost('handle'))) {
19         $this->view->errors[] = "Handle must be between 2 and 14 characters.";
20     } // end valid handle
21
22     // Handle must consist solely of alphanumeric characters
23     $validHandle = new Zend_Validate_Alnum();
24     if (! $validHandle->isValid($this->_request->getPost('handle'))) {
25         $this->view->errors[] = "Handle must consist of letters and numbers.";
26     } // end valid handle
27
28     // Handle cannot already exist in database
29     $foundUser = $user->getUserByHandle($this->_request->getPost('handle'));
30     if ($foundUser->id != "") {
31         $this->view->errors[] = "Handle already exists in database.";
32     }
33
34     // Password must be at least 6 characters
35     $validPswd = new Zend_Validate_StringLength(6,20);
36     if (! $validPswd->isValid($this->_request->getPost('password'))) {
37         $this->view->errors[] = "Password must be at least 6 characters.";
38     } // end valid password
39
40     // First name must not be empty
41     $validFirstName = new Zend_Validate_NotEmpty();
42     if (! $validFirstName->isValid($this->_request->getPost('first_name'))) {
43         $this->view->errors[] = "Please provide your first name.";
44     } // end valid first name
45
46     // Last name must not be empty
47     $validLastName = new Zend_Validate_NotEmpty();
48     if (! $validLastName->isValid($this->_request->getPost('last_name'))) {
49         $this->view->errors[] = "Please provide your last name.";
50     } // end valid last name
51
52     // Valid gender?
53     if (! Zend_Validate::is($this->_request->getPost('gender'), 'NotEmpty')) {
54         $this->view->errors[] = "Please identify your gender.";
55     } // end valid gender
56
57     // If errors exist, prepare the form data for inclusion in the form so
58     // the user doesn't have to repopulate the data
59     if (count($this->view->errors) > 0) {
60
61         $this->view->email          = $this->_request->getPost('email');
62         $this->view->handle         = $this->_request->getPost('handle');
63         $this->view->first_name      = $this->_request->getPost('first_name');
64         $this->view->last_name       = $this->_request->getPost('last_name');
65         $this->view->gender          = $this->_request->getPost('gender');
66

```

```

67     // No errors, add the user to the database and send the confirmation e-mail
68 } else {

```

Let's review the code:

- We'll process the form once we've determined it's been posted by examining the return value of the `$this->getRequest()->isPost()` method, as shown in line 02.
- Lines 04-55 perform the series of validations, checking various facets of each form field value. Should any of the validations fail, an appropriate message will be appended to the `$this->view->errors` array. These aren't all standard validation procedures as defined by the Zend Framework; we also rely upon two methods found in the User model (`getUserByHandle()` and `getUserByEmail()`) to ensure the provided handle and e-mail address don't already exist.
- Once the validations are complete, the size of the `$this->view->errors` array will be determined (line 59). If it's greater than zero, meaning errors have occurred, several view variables will be created and assigned the values of the user's provided form field entries, so we can repopulate the form and save the user some time and frustration. Otherwise, if no errors have occurred, (line 68) we'll begin the process of adding the user's registration data to the `users` table and preparing and sending the confirmation e-mail.

Next, let's take a look at the second step in the register action, in which the user's registration data is inserted into the table (Listing 8-3). In the next section, "Confirming Registration", we'll review the final step of the action, in which the user's confirmation e-mail is prepared and sent.

Listing 8-3. Inserting the registration data and sending the confirmation email

```

01 } else {
02
03     // Generate the random key used for registration confirmation
04     $registrationKey = $this->generate_random_string();
05
06     // Prepare the data array for database insertion
07     $data = array (
08         'email'           => $this->_request->getPost('email'),
09         'password'        => md5($this->_request->getPost('password')),
10         'registration_key' => $registrationKey,
11         'handle'          => $this->_request->getPost('handle'),
12         'first_name'       => $this->_request->getPost('first_name'),
13         'last_name'        => $this->_request->getPost('last_name'),
14         'gender'           => $this->_request->getPost('gender'),
15         'created_at'       => date('Y-m-d H:i:s'),
16         'updated_at'       => date('Y-m-d H:i:s'),
17         'last_login'        => date('Y-m-d H:i:s')
18     );
19
20     // Insert the registration data into the database
21     $user = new User();
22     $user->insert($data);

```

Let's review the code:

- Line 01 continues where we left off in Listing 8-2 (reproducing line 68), inserting the user's registration data into the users table and sending the confirmation e-mail. Line 04 creates the random string (how this is done is covered in the next section) used to identify the user during the confirmation process.
- Lines 07-18 create an array containing the user data which will subsequently be inserted into users table. Note how the password is encrypted using PHP's `md5()` function, which converts the provided password into a 32-character string which cannot be reverted back to the original value. When logging the user into the system we'll also encrypt the provided password and compare it to the value stored in the password column. We'll also auto-assign the current timestamp to the `created_at`, `updated_at`, and `last_login` columns.
- Lines 21-22 complete the insertion process by inserting the array into the `users` table.

Once the user's registration data has been saved to the database, it's time to prepare and send the confirmation e-mail. This final step of the process is covered in the next section.

Confirming Registration

To combat spammers and individuals who attempt to potentially manipulate the website by registering multiple accounts, it's a good idea to require the user to confirm registration by verifying his provided e-mail is valid. You can do this by sending a confirmation e-mail to the user's account following submission of the form. You already learned how accomplish the majority of this task in Chapter 7, however there are a few other steps to the task making coverage of this topic a worthwhile endeavor.

The confirmation process is typically carried out by asking the user to click on a link found in the confirmation e-mail. This link will include a unique key which was generated at the moment of registration and associated with that user by storing it in the `registration_key` column of the `users` table. To generate the random key, add the following private method to your `Gamers` controller:

```

01 /**
02  * Generate a unique random string
03 *
04  * @param int    $length  Length of string to be generated
05  * @return string $str      Random string
06 */
07 private function generate_random_string($length=32)
08 {
09     // Allowable random string characters
10     $seeds = 'abcdefghijklmnopqrstuvwxyz0123456789';
11
12     // Generate the random string
13     $str = '';
14     $count = strlen($seeds);
15     for ($i = 0; $length > $i; $i++)
16     {

```

```

17     $str .= $seeds[mt_rand(0, $count - 1)];
18 }
19
20 return $str;
21 }

```

The code breakdown follows:

- Line 07 defines the private method, setting a default length of the randomly generated string to 32 characters.
- Line 10 defines the set of allowable characters which can appear in the string
- Lines 15-18 build the random string, using the `mt_rand()` function to randomly choose an integer between 0 and 35, which is then used to retrieve a character located in that offset position of the `$seeds` string.

As you saw earlier in this section, we'll call this method when inserting the newly registered user's record into the database, adding the random string to the `registration_key` column. Following that, we'll e-mail the user:

```

01 try {
02     // Create a new mail object
03     $mail = new Zend_Mail();
04
05     $mail->setFrom($this->config->email->from_admin);
06     $mail->addTo($this->_request->getPost('email'),
07                 "{$this->_request->getPost('first_name')}"
08                 "{$this->_request->getPost('last_name')}\"");
09     $mail->setSubject('GameNomad: Please confirm your registration');
10
11     include "_email-confirm-registration.phtml";
12
13     $mail->setBodyText($email);
14     $mail->send();
15
16     $this->view->success = 1;
17 } catch (Exception $e) {
18     $this->view->errors[] = "We were unable to send your confirmation
19                           e-mail. Please contact {$this->config->email->support}.";
```

Per usual, let's review the code:

- Lines 03 through 09 configure the e-mail, setting the sender and recipient addresses along with an e-mail subject.
- Line 11 pulls the registration e-mail text into the script. I'll show you what this file looks like in a moment.

- Line 13 adds the e-mail text to the prepared e-mail. Notice it's using an `$email` variable which isn't otherwise found in the script. This is because that variable is found in the included file, `_email-confirm-registration.php`. As mentioned, in a moment you'll understand exactly what's being done to make this possible.
- Line 14 sends the e-mail and sets the success flag accordingly. If an error occurs, the errors array is set.

I mentioned the e-mail body a few times during the code breakdown.

```

01 <?php
02
03 $email = <<< email
04 Dear {$this->_request->getPost('first_name')},
05
06 Welcome to the GameNomad community! To confirm your e-mail address, click
07 on the following URL:
08
09 {$this->config->website->url}/gamers/verify/key/{$registrationKey}
10
11 Questions? Comments? Contact us at {$this->config->email->support}!
12
13 Thank you,
14 The GameNomad Team
15 email;
16
17 ?>
```

The code summary follows:

- Because this code is included into the register action, any variables found in the string assigned to `$email` will be interpolated in the scope of the action. Therefore the variables found on lines 04, 09, and 11 will all be converted to their appropriate values before being assigned along with the rest of the string to the `$email` variable.

Once the user receives the e-mail, he can click on the confirmation link, which would look very similar to this:

```
http://www.gamenomad.com/gamers/verify/key/vhflniwd6omzw87txqsdboi5uyf3bol4
```

As you can surmise by this URL, we'll need to add another action to the `Gamers` controller named `verify`. This action will accept as input a parameter named `key` which contains the registration key:

```

01 /**
02  * Completes the registration process by validating a user's email address
03  *
04  */
05 public function verifyAction() {
06
07     $this->view->pageTitle = "Complete the Registration Process";
```

```

08 // Retrieve the key from the URL
09 $registrationKey = $this->_request->getParam('key');
10
11 // Identify the user associated with this key
12 $user = new User();
13 $query = $user->select()->where('registration_key = ?', $registrationKey);
14 $result = $user->fetchRow($query);
15
16 // If the user has been located, set the confirmed column.
17 if(count($result) == 1) {
18     $result->confirmed = 1;
19     $result->save();
20     $this->view->success = 1;
21     $this->view->firstName = $result->first_name;
22 } else {
23     $this->view->errors = "We were unable to locate your registration key.";
24 }
25
26
27 }

```

Let's review the code:

- Line 10 retrieves the registration key from the URL. This key is subsequently used on line 14 to retrieve the user record from the `users` table.
- If the record is located (line 18), we'll set the record's confirmed attribute (line 19) and save the record back to the database (line 20).
- We'll subsequently set the usual success flag (line 21), and retrieve the user's first name (line 22) so it can be used within the view message.

The corresponding view (`verify.phtml`) looks like this:

```

<h1>Complete the Registration Process</h1>

<?php if (count($this->errors) > 0) { ?>
    <div id="errors">
        Errors were encountered while creating your account:<br />
        <?= $this->Errors($this->errors); ?>
    </div>
<?php } ?>

<?php if ($this->success == 1) { ?>

    <p>
        Congratulations <?= $this->firstName; ?>, your e-mail address has
        been verified! <a href="/gamers/login">Login to your account</a>
        and begin building your game collection.

```

```
</p>
<?php } ?>
```

Step #3. Managing User Logins

Once the user has confirmed registration, he's free to begin logging into the site and take advantage of any special features afforded to registered users. In this section you'll learn how to create the interface and action used to log the user into and out of your website, as well as create a password recovery tool in the event the user forgets it.

Logging the User into the Website

To login to the website, the user must provide an e-mail address and password, as shown in Figure 8-2. This form points to the Gamers controller's login interface, which will be dissected in this section.

The screenshot shows a login form titled "Login to Your GameNomad Account". The instructions say: "To login to your GameNomad account, enter your e-mail address and password below. If you've forgotten your password, [click here to recover it](#). Not a member? [Click here to register](#)". There are two input fields: "Your E-mail Address:" and "Your Password:", both with placeholder text. Below the password field is a "Login!" button.

Figure 8-2. The login interface

The `login` action is responsible for comparing the provided e-mail address and password with those found in the `users` table. While this is easily done using a simple table query, other more complex issues remain. Notably, if a match is found, we need to establish a `session` which will keep the user logged into the site as he navigates from one page to the next.

Also, while a database is a common solution for managing user accounts, it's just one of many popular authentication backends; you might have heard of, or even relied upon, solutions such as LDAP, Open ID, or even a simple text file for user account management. To accommodate both the gory details surrounding session management and the number of account backend solutions, the Zend developers created the `Zend_Auth` authentication component. We'll use this component to build the features introduced in this section.

Listing 8-4 presents the login action. Take some time to review the code, and carefully read the ensuing breakdown.

Listing 8-4. The login action

```
01 public function loginAction()
02 {
03     if ($this->getRequest()->isPost()) {
04
05         // Retrieve the provided email address and password
06         $email = $this->_request->getPost('email');
07         $password = $this->_request->getPost('password');
08
09         // Make sure the email and password were provided
10        if (empty($email) || empty($password)) {
11            $this->view->errors[] = "Provide e-mail address and password.";
12        } else {
13
14            // Identify the authentication adapter
15            $authAdapter = new Zend_Auth_Adapter_DbTable($this->db);
16
17            // Identify the table where the user data is stored
18            $authAdapter->setTableName('users');
19
20            // Identify the column used to store the "username"
21            $authAdapter->setIdentityColumn('email');
22
23            // Identify the column used to store the password
24            $authAdapter->setCredentialColumn('password');
25
26            // How is the password stored?
27            $authAdapter->setCredentialTreatment('MD5(?)');
28
29            // Pass the provided information to the adapter
30            $authAdapter->setIdentity($email);
31            $authAdapter->setCredential($password);
32
33            $auth = Zend_Auth::getInstance();
34            $result = $auth->authenticate($authAdapter);
35
36            // Did the participant successfully login?
37            if ($result->isValid()) {
38
39                // Retrieve the user so can update the login timestamp
40                $user = new User();
41                $updateLogin = $user->getUserByEmail($email);
42
43                if ($updateLogin->confirmed == 1) {
44
45                    // Update the login timestamp and save the row
46                    $updateLogin->last_login = date('Y-m-d H:i:s');
47                    $updateLogin->save();
48
```

```

49                     // Redirect the user to the index page
50                     $this->redirect('/');
51
52             } else {
53
54                 $this->view->errors[] = "The e-mail address associated
55                     with this account has not been confirmed.";
56
57             }
58
59         } else {
60             $this->view->valid = 0;
61             $this->view->errors[] = "Login failed.";
62         }
63
64     }
65
66 }
67 }
```

Let's review the code:

- Lines 06-07 store the provided e-mail address and password in more accessible variables.
- Line 10 performs a quick check to determine whether the user mistakenly omitted either the e-mail address or password. If not the action moves on to the authentication verification step.
- Line 15 identifies the type of authentication adapter we'll be using, in this case a database. Notice the database handle is being passed to the adapter (`$this->db`). This handle is located in the controller's `init()` method, as you learned is a commonplace strategy in Chapter 5.
- Line 18 identifies the name of the table used to store the user account information.
- Lines 21 and 24 identify the column names used to store the identity, or username (in our case the e-mail address), and the password.
- Line 27 defines the encryption scheme used to encode the password. This works like a prepared statement; you could substitute `MD5()` for any supported PHP function, although `MD5()` is recommended.
- Line 33 invokes the `Zend_Auth` class in a special way. Instead of using the `new` operator, we use `getInstance()` to make sure there's only ever one instance of the class available to the script; if there isn't, `getInstance()` will create one, otherwise it will use the one already instantiated. This is known as the Singleton strategy (or pattern).
- Line 34 uses the newly created object to authenticate the user, passing in the adapter information. If the e-mail address and password match a pair found in the `users` table (line 37), lines 38 through 58 carry out various custom tasks such as updating the user's `last_login` column and redirecting the user to the index page.

In this last bullet point I mentioned we're redirecting the user to the index page. While I prefer to do this for reasons of expediency, chances are you'll want to offer some indication to the user that he has indeed successfully logged in. For instance, Figure 8-3 presents two screenshots indicating how the GameNomad website depicts a user's login status.



Figure 8-3. Determining the user's login status

You can easily determine the user's login status using the Zend_Auth class' `hasIdentity()` and `getIdentity()` methods, as is demonstrated below. I place this snippet in the `init()` method of controllers where I'd like to ascertain this status, setting a `$user` variable intended for access within the view scope:

```
if (Zend_Auth::getInstance()->hasIdentity()) {
    $user = new User();
    $this->view->user = $user->getUserByEmail(
        Zend_Auth::getInstance()->getIdentity());
} else {
    $this->view->user = "";
}
```

Within the view scope, you can then simply refer to the `$user` variable, determining whether it's been set:

```
<?php if (! $this->user) { ?>
    <a href="/gamers/register">Register</a> | <a href="/gamers/login">Login</a>
<?php } else { ?>
    Hello, <?php echo $this->user->handle; ?> | <a href="/gamers/logout">Logout</a>
<?php } ?>
```

This example's reference to a logout feature brings us to the next section which shows you how to do exactly that: log a user out of the website.

Logging the User Out of the Website

Most users will prefer to maintain their session for reasons of convenience, allowing them to automatically login upon each return to the site. However, because many users login from insecure locations such as a library or internet cafe, you'll need to provide an explicit means for logging out of the system. To do so, create a logout action within the `Gamers` controller, shown in Listing 8-5.

Listing 8-5. The logout action

```
01 /**
02 * Logs the user out of the application
03 *
04 */
05 public function logoutAction()
06 {
07     Zend_Auth::getInstance()->clearIdentity();
08     $this->_redirect('/');
09 }
```

This action will log the user out of the application by first deleting the session (line 07) and then redirecting the user to the website's index page. Any subsequent attempts to access pages requiring authentication will be denied until the user logs in anew.

Note you won't need a corresponding view for the `logout` action, because the user is immediately redirected to the home page.

Resetting the User's Password

Over time users have devised innumerable strategies in an attempt to recall the often dozens of user passwords they're required to create and manage in order to perform their daily online activities. Inevitably though, a user will eventually forget a password, barring his irresponsible use of the same password with every account. To minimize the user's frustration when the inevitable does occur, your site should have a password recovery feature which allows the user to easily reset and recover the new password. Neglecting to include this feature will ultimately result in users contacting you with requests to reset the password for them, creating a new and therefore redundant account, or worst of all, quitting coming to your site altogether out of frustration. Fortunately, creating this feature is easy!

VIDEO. Recovering Passwords with Zend_Auth

The `Zend_Auth` component makes managing user registrations and logs very easy, but the process of recovering passwords is often a bit more confusing. This video discusses the factors involved in recovering user passwords, and showing you how to implement this feature using `Zend_Auth`. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Because for security purposes the user's chosen password has been encrypted using a one-way algorithm, there is no way to simply retrieve and send it to the user. Therefore we'll need to create a solution for allowing the user to explicitly reset the password. In order to do so securely, the user will need to verify his identity either by responding to a personal question which presumably only he'd know the answer to, clicking on a password reset link sent to his e-mail address, or perhaps some combination of the two approaches, as is often seen when dealing with particularly sensitive data. Personally I think simply requiring the user to click on e-mailed link which initiates the password recovery process is more than suffice for most situations, so for the purposes of this exercise we'll implement the feature in this fashion.

Initiating the Password Recovery Process

Should the user forget his password, he'll need a means for initiating the recovery process. The easiest way to do so is to simply ask the user to provide his username (in our case, his e-mail address). After verifying this username exists in the user table, you can generate a random key (using the same `generate_random_string()` method used to create the registration confirmation key) and send that key to the user in the same fashion we used to confirm a user's registration. For instance, the initial step in GameNomad's password recovery process is shown in Figure 8-4.

The screenshot shows a web form with a light gray background. At the top left is the title "Forgot Your Password?". Below it is a sub-instruction: "Provide your registered e-mail address below, and we will reset your password and send it to you." A text input field is labeled "Your e-mail address?" and contains a placeholder email address. At the bottom right of the form is a dark gray button labeled "Reset Password".

Figure 8-4. Initiating the password recovery process

Once the user's e-mail address has been located in the `users` table, a random string is generated, attached to the user's record (I typically just reuse the `registration_key` field since it serves no other purpose following registration confirmation), and an e-mail is sent to the user in the very same fashion in which was used for registration confirmation. The password recovery URL looks like this:

```
http://www.gamenomad.com/gamers/reset/key/a97r37vo182sp1tquu9npguj07h5hg4p
```

Once clicked, the user will be taken to the reset action, which will begin by prompting the user to choose and confirm a new password, as shown in Figure 8-5.

The screenshot shows a web form with a light gray background. At the top left is the title "Reset Your Password?". Below it is a sub-instruction: "Your password recovery request has been confirmed. Use the following form to reset your password." Two text input fields are present: one labeled "Your New Password" and another labeled "Confirm Your New Password", both containing placeholder text. At the bottom right of the form is a dark gray button labeled "Reset Password".

Figure 8-5. Prompting the user to reset his password

The `reset` action handles the display of the password reset form as well as carries out the process of updating the user's account to reflect the new password. It's presented next (Listing 8-6), followed by a breakdown of the relevant code.

Listing 8-6. Resetting the user's password

```
01 /**
02 * Completes password recovery process
03 */
04
```

```
05 public function resetAction()
06 {
07
08     $this->view->pageTitle = "Reset your password";
09
10    // If the form has been posted, reset the password
11    if ($this->getRequest()->isPost()) {
12
13        // Password must be at least 6 characters
14        $valid_pswd = new Zend_Validate_StringLength(6,20);
15        if (! $valid_pswd->isValid($this->_request->getPost('password'))) {
16            $this->view->errors[] = "Password must be at least 6 characters.";
17        } // end valid password
18
19        if ($this->_request->getPost('password') !=
20            $this->_request->getPost('password2')) {
21            $this->view->errors[] = "Your passwords do not match.";
22        }
23
24        // No errors, so update the password
25        if (count($this->view->errors) == 0) {
26
27            $user = new User();
28            $query = $user->select()->where('registration_key = ?',
29                                         $this->_request->getPost('key'));
30            $result = $user->fetchRow($query);
31
32            if (count($result) == 1) {
33                $result->password = md5($this->_request->getPost('password'));
34                $result->save();
35                $this->view->updated = 1;
36            }
37
38        } else {
39            $this->view->success = 1;
40            $this->view->key = $this->_request->getPost('key');
41        }
42
43    // User has clicked the emailed password recovery link. Find the user
44    // using the recovery key, and prepare the password reset form
45    } else {
46
47        $recoveryKey = $this->_request->getParam('key');
48
49        $user = new User();
50        $query = $user->select()->where('registration_key = ?', $recoveryKey);
51
52        $result = $user->fetchRow($query);
```

```

54     if(count($result)) {
55         $result->save();
56         $this->view->success = 1;
57         $this->view->key = $recoveryKey;
58     } else {
59         $this->view->errors[] = "Unable to locate password recovery key.";
60     }
61
62 }
63 }
```

Let's review some code:

- If the form has been posted, lines 13-22 will perform two validations, ensuring the provided password is of a length between six and twenty characters, and also making sure the password and confirmation password match.
- If validation is successful, lines 27-30 retrieve the user's row by looking up the registration key. If it's located, lines 32-36 perform the password update, making sure the provided password is first hashed using the `md5()` function before saving the password to the database.
- Lines 45-63 execute if the user is retrieving the form for the first time (presumably by way of clicking on the link found in the password recovery e-mail). Lines 50-52 use the provided recovery key to determine whether the key exists in the database. If so, the form is presented. Otherwise, an error message is displayed.

Step #4. Displaying User Profiles

Most, if not all social network driven sites provide users with the ability to view at least some part of the other registered users' profiles. Of course, you might limit the display of certain parts of the profiles to just the user's friends (discussed in the next step), but attributes such as each user's first and last name, gender, and time of last login seem to be fair game.

You'll also want to provide users with an easy way to point others to their profile. One easy way is by using an easily recallable URL which includes the user's gaming handle, such as:

<http://www.gamenomad.com/gamers/profile/wjgilmore>

By taking advantage of the `Gamers` controller's `index` action you could even cut the URL down to just:

<http://www.gamenomad.com/gamers/wjgilmore>

Using this sort of approach, retrieving a user's profile is trivial; just retrieve the handle from the URL and use the `User` model's `getUserByHandle()` method to retrieve the user. The `Gamers` controller's `profile` action demonstrates how this is done:

```

public function profileAction()
{
    $handle = $this->_request->getParam('handle');

    $this->view->pageTitle = "Gamer Profile: {$handle}";

    $user = new User();
    $this->view->gamer = $user->getUserByHandle($handle);

    if ($this->view->gamer->id == "") {
        $this->view->errors[] = "This user does not exist.";
    }
}

```

The corresponding profile view either outputs an error if the provided user handle does not exist, or outputs some basic profile information, including the user's gaming handle, date of account creation, and the last time the user logged into the site:

```

<?php if (count($this->errors) > 0) { ?>
    <div id="errors">
        Errors were encountered while searching for this user:<br />
        <?php echo $this->Errors($this->errors); ?>
    </div><br />
<?php } else { ?>

    <p>
The user <b><?= $this->escape($this->gamer->handle); ?></b> has been a member
since <b><?= date('F d, Y', strtotime($this->gamer->created_at)); ?></b>,
and last logged in on
<b><?= date('F d, Y @ h:i:s', strtotime($this->gamer->last_login)); ?></b>.
</p>

<?php } ?>

```

Step #5. Making Friends

In this fifth and final step of the chapter, we'll discuss one of the fundamental aspects of any website sporting a social slant: connecting users. By granting users the ability to connect with their friends, we can start offering an abundant array of new and interesting social features, such as knowing when your friends are also online and viewing restricted content only available to a user's network. Integrating the basic framework for implementing these sorts of features is easier than you might think!

To begin, we need a way to map these connections. But for starters, how are the connections initiated in the first place? It wouldn't be acceptable to allow users to automatically add friends to their list; instead, one user would have to invite another to join his network. This is done in a manner very similar to that already used in this chapter to confirm user registration and change a password: by generating

a unique key which is attached to an invitation. The user invites a user by initiating a sequence of events which creates an invitation, generates a unique ID, and mails the invitation to the prospective friend-in-question. While you're by now familiar enough with this process that I won't go into it anew here, it is worth showing the `invitations` database table (Listing 8-7) and corresponding `Invitation` model (Listing 8-8) here. Furthermore, in the interests of space I'll only include the method bodies found in the `Invitation` class; based on the method names alone you'll be able to easily identify their purpose. Of course, if you'd like to review the actual code you'll find it in its entirety in the code download.

Listing 8-7. The invitations database table

```
CREATE TABLE invitations (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    created_on TIMESTAMP NOT NULL,
    invitation_key CHAR(32) NOT NULL,
    inviter_id INTEGER UNSIGNED NOT NULL,
    invitee_id INTEGER UNSIGNED NOT NULL
);
```

The code review follows:

- The `created_on` column identifies the time in which the invitation was created. You might track this information in order to periodically delete invitations which have been outstanding for a long period of time.
- The `invitation_key` is the 32 character random string generated in the same fashion as the other random keys in this chapter. It's used to uniquely and securely identify the invitation.
- The `inviter_id` is the primary key assigned to the user who is inviting the user to join his network.
- The `invitee_id` is the primary key assigned to the user who is being invited to join the inviter's network.

Next let's take a look at the `Invitation` model.

Listing 8-8. The Invitation model

```
01 class Invitation extends Zend_Db_Table_Abstract
02 {
03
04     protected $_name = 'invitations';
05     protected $_primary = 'id';
06
07     protected $_referenceMap = array (
08         'Inviter' => array (
09             'columns'      => array('inviter_id'),
10             'refTableClass' => 'User'
11         ),
12         'Invitee' => array (
```

```
13     'columns'      => array('invitee_id'),
14     'refTableClass' => 'User'
15   )
16 );
17
18 public function exists($inviter, $invitee)
19 {
20 }
21
22 public function getInvitation($key)
23 {
24 }
25
26 public function createInvitation($inviter, $invitee)
27 {
28 }
29
30 public function completeInvitation($key)
31 {
32 }
33
34 }
```

The code summary follows:

- As you learned in Chapter 6, the Zend Framework offers a powerful solution for relating a website's data models. In the case of the `Invitation` model, we're formally relating the `inviter_id` and the `invitee_id` to the `User` model (lines 07-16). Of course, you'll also need to adjust the `User` model to complete this relation. If you don't know how to do this, consult the code download, or refer back to Chapter 6.
- Defined on line 18, the `exists()` method determines whether an invitation already exists. If so, you might inform the user that an invitation is already outstanding instead of silently continuing to send additional invitations to a user who may not desire to accept it.
- Defined on line 22, the `getInvitation()` method retrieves an invitation based on the provided unique key.
- Defined on line 26, the `createInvitation()` method creates a new invitation.
- Finally, defines on line 30, the `completeInvitation()` method completes the connection process by formally connecting the inviter and invitee together, and removing the invitation from the `invitations` table.

This last bullet point mentions the `completeInvitation()` method's role in completing the connection process. This is done using another table and corresponding model, namely `friends` and `Friend`, respectively. The `friends` table is shown in Listing 8-9, followed by the `Friend` model found in Listing 8-10.

Listing 8-9. The friends database table

```
CREATE TABLE friends (
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    friend_a_id INTEGER UNSIGNED NOT NULL,
    friend_b_id INTEGER UNSIGNED NOT NULL,
    friends_since TIMESTAMP NOT NULL
);
```

Let's breakdown the code:

- The `friend_a_id` column contains the primary key of one of the friends found in the connection. The `friend_b_id` column contains the primary key of the other. While there are many ways to organize these connections, to eliminate the need for additional code I just insert two rows for each connection, which allows me to easily retrieve a user's friends simply by matching the user's primary key to that found in the `friend_a_id` column. Of course, should you need to remove a connection you'll need to be sure to remove both rows.
- The `friends_since` column identifies the time the connection was made. While not necessary it might be at some time useful should you desire to perform some level of network analysis.

Listing 8-10. The Friend model

```
01 class Friend extends Zend_Db_Table
02 {
03
04     protected $_name = 'friends';
05
06     protected $_primary = 'id';
07
08     protected $_referenceMap = array (
09         'FriendA' => array (
10             'columns'      => array('friend_a_id'),
11             'refTableClass' => 'User'
12         ),
13         'FriendB' => array (
14             'columns'      => array('friend_b_id'),
15             'refTableClass' => 'User'
16         )
17     );
18 }
```

The code review follows:

- Like the `Invitation` model, the `Friend` model relates to the `User` model via the `friend_a_id` and `friend_b_id` columns. We connect the `Invitation` model to the `User` model via the `FriendA` and `FriendB` rule keys. You'll also need to adjust the `User` model to complete this relation. If you don't know how, consult the code download, or refer back to Chapter 6.

Adding a Friend

As mentioned, the connection process is two-step in nature: the process starts with a user (inviter) inviting another user to connect, and ends when the invitee accepts the invitation. The first part of this process should be apparent by this point in the chapter, so let's concentrate on the second part, accomplished using the Gamers controller's `connected` action, presented in Listing 8-11.

Listing 8-11. The connected action

```
01 public function connectedAction()
02 {
03
04     // Set the page title
05     $this->view->pageTitle = "Complete a GameNomad Connection";
06
07     // Retrieve the invitation key
08     $key = $this->_request->getParam('key');
09
10    // Retrieve the invitation
11    $invitation = new Invitation();
12    $this->view->outstandingInvitation = $invitation->getInvitation($key);
13
14    // Complete the invitation process
15    $connected = $invitation->completeInvitation($key);
16
17    // Determine the outcome response
18    if ($connected) {
19        $this->view->success = 1;
20    } else {
21        $this->view->errors[] = "We could not complete the connection request.";
22    }
23
24 }
```

Although the logic found in the `connected` action should be obvious, let's nonetheless quickly review the code:

- Line 08 peels the invitation key from the URL. Remember, the invitee arrived here by clicking on an invitation sent to his e-mail address
- Line 12 retrieves the invitation by passing the key into the `Invitation` model's `getInvitation()` method.
- Line 15 builds the connection between the two users by completing the invitation process using the `Invitation` model's `completeInvitation()` method. If the connection process is successful, the `success` flag is set, otherwise, an error message is generated.

Displaying a User's Friends

You'll need to retrieve a list of a user's friends for various reasons, even if it's simply to display the list on the user's profile. One way to do it is to use a join statement (introduced in Chapter 6). Listing 8-12 presents just such a statement, found in the `User` model's `getFriends()` method.

Listing 8-12. The User model's getFriends() method

```
function getFriends()
{
    $user = new User();

    $query = $user->select();
    $query->from(array('u' => 'users'),
        array('u.id', 'u.handle', 'u.first_name', 'u.last_name'));
    $query->join(array('f' => 'friends'), 'f.friend_id = u.id', array());
    $query->where('f.user_id = ?', $this->id);

    $results = $user->fetchAll($query);
    return $results;
}
```

Within the `Gamers` controller's `profile` action you can add the following line:

```
$this->view->friends = $this->view->gamer->getFriends();
```

Within the profile view you can display a list of friends using the following code snippet:

```
<?php if (count($this->friends) > 0) { ?>
<ul>
<?php foreach($this->friends AS $friend) { ?>
    <li>
        <a href="/gamers/profile/<?= $this->escape($friend->handle); ?>">
            <?= $this->escape($friend->handle); ?>
        </a>
    </li>
<?php } ?>
</ul>
<?php } else { ?>
    <p>This user hasn't connected with any other gamers.</p>
<?php } ?>
```

Conclusion

This chapter introduced you to several important aspects of any socially-driven website. We built features for registering new users, managing logins and important account information such as passwords, and displaying user profiles. We also discussed a simple approach for connecting users together to form social networks. In the next chapter, I'll show you how to add a new dimension to your website using the Google Maps' mapping solution.

CHAPTER 9

Integrating Google Maps

Although Web-based mapping services such as MapQuest (<http://www.mapquest.com/>) have been around for years, it wasn't until Google's release of its namesake mapping API (Application Programming Interface) that we began the love affair with location-based web sites. This API provides you with not only the ability to integrate Google Maps into your website, but also to build completely new services built around Google's mapping technology. Google Maps-driven websites such as <http://www.walkjogrun.net/>, <http://www.housingmaps.com/>, and <http://www.everyblock.com/> all offer glimpses into what's possible when armed with knowledge of this API and a healthy dose of imagination.

In this chapter you'll learn how to take advantage of the Google Mapping API to add an exciting new dimension to your website. You'll not only learn about Google Maps fundamentals, but also how to obtain the latitudinal and longitudinal coordinates of each user, and plot their locations on a map. Many of the topics discussed in this chapter are currently being integrated into the GameNomad website!

VIDEO. A Tour of the Google Maps API

Based on the author's popular six-article Developer.com series (<http://www.developer.com/services/article.php/3680076>), this video introduces you to key features of the Google Maps API, and shows you how to integrate mapping features into your PHP-driven website using the GoogleMapAPI library. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Chapter Steps

The goals of this chapter are accomplished in two steps:

- **Step #1. Introducing the Google Maps API:** In this introductory step I'll introduce you to the Google Maps API, a powerful JavaScript library which makes it easy to integrate Google maps and map-related features into your website. Although in practice we'll communicate with this API using a PHP library named GoogleMapAPI (introduced in Step #2), beginning by interacting directly with the native API is a great way to become familiar with its features.
- **Step #2. Adding Mapping Services to Your Website:** Interacting with the Google Maps API directly through the JavaScript interface is useful for understanding the fundamental features, but for practical purposes we'll want to interact with the API using a PHP-based solution. The GoogleMapAPI library provides the bridge we're looking for, making integration of the Google Maps API into your PHP-driven website a breeze. In this step you'll learn all about it. You'll also learn how to convert mailing addresses into their latitudinal and longitudinal coordinates, which Google requires to perform numerous mapping tasks.

TIP. Although the most popular, Google isn't the only freely available mapping solution. Microsoft has long offered its Microsoft Virtual Earth SDK (<http://dev.live.com/virtualearth/>), as does Yahoo! through Yahoo! Maps (<http://developer.yahoo.com/maps/>). These solutions are both well worth a look.

Step #1. Introducing the Google Maps API

In response to growing interest in creating location-based websites, Google opened access to its popular mapping service (<http://maps.google.com/>) in July, 2005. The JavaScript-based API has since become the de facto online mapping solution, making it possible for developers to create custom maps based on any manner of data capable of being pinpointed on a map.

Because the API is bundled within a single JavaScript file, and hosted on Google's massive server farm, you're not required to muddle through a complex installation and configuration process. All you have to do is reference the remote API through your scripts, and begin using its features. Of course, you have to know how to use these features, which is the subject of much of this chapter.

Although advertising is currently not inserted into maps served through the Google Maps API, the terms of service state Google has the right to do so at some point in the future. Given the stiff competition raging between the various mapping solutions, I believe we're still years away from having to adjust to advertising, however bear in mind it could happen at any time.

Request a Google Maps API Key

Because using the Google Maps API is free, Google is logically interested in being able to monitor and potentially limit the resources it devotes to the service. It does this by tying each request to a unique originating domain and user key. Google uses this information to keep tabs on the total number of requests, limiting the number of requests per account to 500,000 per day. To identify your URL and request a key, navigate to the following URL:

<http://code.google.com/apis/maps/signup.html>

Once you've perused and accepted the Google Maps API terms of use, enter your URL and press the "Generate API Key" button. Chances are you're just going to want to enter your domain name (for instance `wjgilmore.com`), because Google will consider all subdomains and subdirectories valid as well. However if you enter `www.wjgilmore.com`, then you will not be able to use the same API key to communicate with Google Maps via `calendar.wjgilmore.com`. If you're developing on a local machine, and your network has a static IP address, you can use the IP in lieu of a domain name. One easy way to determine your network's static address is to navigate to `http://myip.wjgilmore.com`.

Google will respond to your request with your unique key, along with a sample Google Maps-enabled web page, shown in Listing 9-1.

Listing 9-1. Google's sample map-enabled web page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Google Maps JavaScript API Example</title>
<script src="http://maps.google.com/maps?file=api&v=2&key=YOUR_API_KEY"
 type="text/javascript"></script>
<script type="text/javascript">

//<![CDATA[

function load() {
    if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
    }
}

//]]>
</script>
</head>
<body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
</body>
</html>
```

Copy this page to a PHP file named `firstmap.html` and save it to a convenient location within your web server's `htdocs` directory and call it from the browser. You'll see a Google map identical to the one shown in Figure 9-1.



Figure 9-1. Creating Your First Google Map

Take note of the `onload()` event handler within the body tag; this is a JavaScript feature which will

cause the `load()` function to execute once the web page has been loaded into the browser. Neglecting to include this event handler will cause the map not to render, so be sure it's included within your layout file if you're experimenting with Google Maps from within a Zend Framework-driven website. Also be sure to include the `onunload="GUnload()"` event handler, which will help ensure system stability by properly reverting allocated resources required by the API back to the system.

Google Maps API Fundamentals

The Google Maps API is an object-oriented JavaScript library which provides you with the ability to render, manipulate and add content to your own Google maps. If you're not well acquainted with JavaScript, not to worry; the examples found in the following sections will thoroughly introduce you to its syntax.

Determining Map Size

While it's possible to configure the map size when instantiating the `GMap2` object, the map size is typically set using the map's `<div>` container. In Listing 9-1 this size setting is 500 x 300 pixels, as specified by this line:

```
<div id="map" style="width: 500px; height: 300px"></div>
```

Without other instructions, the map will automatically fill to the size of the `div` container, therefore to change the map size all you need to do is change these values accordingly. While there doesn't seem to be any impractical maximum size limit, at certain levels of magnification the ubiquitous copyright statement will unfortunately spill outside of the map graphic. Figure 9-2 demonstrates this bug.



Figure 9-2. Google's copyright statement spills over at certain zoom levels

Because this spillover behavior can become quite exaggerated, be sure to thoroughly test your website to ensure it doesn't negatively affect readability.

Centering the Map atop a Specific Location

So why did the Listing 9-1 map happen to center on Palo Alto, California? The answer lies in the

default latitudinal and longitudinal coordinates used by the script, in this case 37.4419 and -122.1419. Later I'll show you how to convert mailing addresses into the corresponding coordinates (a process known as *geocoding*) but for the moment replace the default coordinates with those representing the address of my favorite local GameStop location: 40.061782, -82.913647. Reload the script and you'll see the map shown in Figure 9-3.



Figure 9-3. Centering the map on my favorite GameStop location

Setting the Map's Zoom Level

While the map shown in Figure 9-3 is indeed centered on GameStop's Easton Town Center location, it's zoomed too far out to give us an indication of its exact whereabouts. You can easily change this by modifying the second parameter of the map object's `setCenter()` method, which in the example script is set to 13. Change this value to 15 and reload the map within your browser, producing the map displayed in Figure 9-4.



Figure 9-4. Zooming in on GameStop's Easton Town Center location

Now we're getting somewhere! As you've probably surmised, the zoom is determined by a scaling range, with 0 representing the most zoomed out viewport, and 17 representing the most zoomed in.

Of course, zooming in to the shop's general location still doesn't entirely enlighten the user as to his exact path to gaming nirvana. What we need is an icon which pinpoints the store's location. Later I'll

show you how this is done, but first let's give the user the ability to easily zoom in and move the map to his liking.

Adding Map Controls

Google Maps are particularly powerful because of the ability to not only zoom the map to varying levels of detail, but also because you can move (also known as panning) the map from one location to another. The API offers controls for facilitating zooming and panning however they're disabled by default. In this section you'll learn how to enable them in various configurations.

Pan and Zoom Controls

The pan and zoom controls give users the ability to move the map about and zoom the viewport to varying degrees of detail, respectively. To add a pan-and-zoom control complete with a zoom slider, add the following line after invoking the GMap2 object (in this and all other examples I'm presuming the GMap2 object is called `map`):

```
map.addControl(new GLargeMapControl());
```

The result of adding this control is shown in Figure 9-5.



Figure 9-5. Adding a Large Pan-and-zoom control

If you're interested in minimizing the presence of this control yet still want to make it available to the user, add the small control with the following line:

```
map.addControl(new GSmallMapControl());
```

The outcome is shown in Figure 9-6.



Figure 9-6. Adding a small pan-and-zoom control

Finally, if you're solely interested in providing the user with the means to zoom in and out of an otherwise stationary map frame (demonstrated in Figure 9-7), reference the `GSmallZoomControl()`:

```
map.addControl(new GSmallZoomControl());
```

The outcome is shown in Figure 9-7.



Figure 9-7. Adding a Small Zoom Control

Map Type Controls

Although Google's mapping service is typically referred to in the context of its now instantly recognizable graphical rendering, this is actually just one of several interfaces offered by Google. Among other interfaces, you can also view satellite imagery, as shown in Figure 9-8.



Figure 9-8. The local GameStop's neighborhood shown in satellite mode

In addition to the satellite view, you probably noticed Figure 9-8 also contains a new control, known as the map type control. Using this control you can alternate between the map and satellite views, and even view a hybrid version which contains elements of both the map and satellite. You can add it to your map by calling the `GMapTypeControl()` method like so:

```
map.addControl(new GMapTypeControl());
```

If you want to prevent users from using one of the map types, you can remove it by subsequently calling the `removeMapType()` method, passing in one of the following options: `G_NORMAL_MAP` (map version), `G_SATELLITE_MAP` (satellite version), `G_HYBRID_MAP` (hybrid version). For instance, the following code will add the map type control but disable the hybrid view:

```
map.addControl(new GMapTypeControl());
map.removeMapType(G_HYBRID_MAP);
```

Managing Map Markers

The maps you've created so far put the user in the general vicinity, but what about pointing him directly to a specific location? Marking a location, or even several, on a Google map is done using a *marker*. Figure 9-9 shows a marker used to pinpoint GameStop's Easton Town Center location.

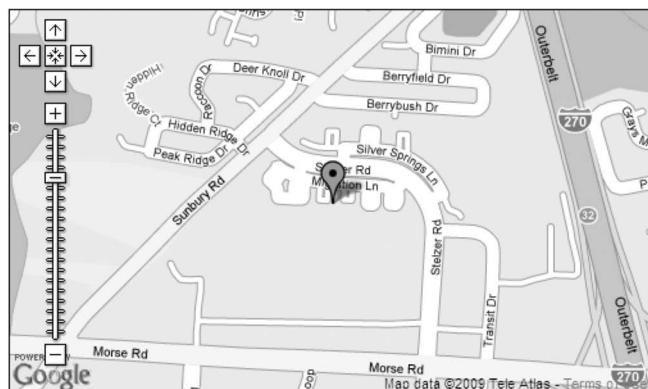


Figure 9-9. Pinpointing GameStop's Easton Town Center location

Creating an icon is done by creating a new point using `GLatLng` and then overlaying it on the map by first instantiating a new `GLatLng` object as was previously done for setting the map's center point, and then passing the `GLatLng` object to a `GMarker` object. The following code was used to create the map shown in Figure 9-9:

```
function load() {  
    if (GBrowserIsCompatible()) {  
        var map = new GMap2(document.getElementById("map"));  
        center = new GLatLng(40.061782, -82.913647);  
        map.setCenter(center, 15);  
  
        gamestop = new GLatLng(40.061782, -82.913647);  
        map.addOverlay(new GMarker(gamestop));  
        map.addControl(new GLargeMapControl());  
    }  
}
```

Adding multiple markers is equally as easy. The following code adds markers representing three of GameStop's Columbus locations:

```
function load() {  
    if (GBrowserIsCompatible()) {  
        var map = new GMap2(document.getElementById("map"));  
  
        center = new GLatLng(39.995649,-83.007116);  
        map.setCenter(center, 10);  
  
        gateway = new GLatLng(39.995649,-83.007116);  
        map.addOverlay(new GMarker(gateway));  
  
        kingsdale = new GLatLng(40.023309,-83.058443);  
        map.addOverlay(new GMarker(kingsdale));  
  
        georgesville = new GLatLng(39.919828,-83.12046);  
        map.addOverlay(new GMarker/georgesville));  
    }  
}
```

Executing this code produces the map shown in Figure 9-10.



Figure 9-10. Pinpointing multiple GameStop locations

Changing the Default Icon

The default red icon used to represent the marker location is fine for most purposes, however replacing it with one more suitable to your website's theme can be a very nice touch. While a tad more complicated to implement than the other features you've encountered so far, chances are you'll spend far more time trying to find or design an appropriate icon than actually incorporating it into your code!

Figure 9-11 displays three instances of the custom icon I used as a replacement for Google's default icon. This icon is part of the awesome Tulliana 2.0 icon set, released under the GNU Lesser General Public License. You can find this set at numerous locations around the Web; just search for it using your favorite search engine.



Figure 9-11. Using a custom icon to represent local GameStop locations

To replace the default icon, instantiate the `GIcon` class as shown on line 05 of the following listing. Lines 06-09 tell the API where to find the replacement icon, the underlying shadow icon, and specify these icons respective sizes in pixels. Line 11 assigns the new icon to the `GMarkerOptions` object (which is instantiated using a syntactical variation known as an object literal). Finally, in lines 16-23 you'll see the `GMarkerOptions` object is passed into the various `GMarker` objects. The result of this listing is shown in Figure 9-11.

```

01 function load() {
02   if (GBrowserIsCompatible()) {
03     var map = new GMap2(document.getElementById("map"));
04
05     var joystickIcon = new GIcon(G_DEFAULT_ICON);
06     joystickIcon.image = "http://localhost/chapter09/joystick.png";
07     joystickIcon.shadow = "http://localhost/chapter09/joystick-shadow.png";
08     joystickIcon.iconSize = new GSize(32,32);
09     joystickIcon.shadowSize = new GSize(32,32);
10
11     markerOptions = { icon:joystickIcon };
12
13     center = new GLatLng(39.995649,-83.007116);
14     map.setCenter(center, 10);
15
16     gateway = new GLatLng(39.995649,-83.007116);
17     map.addOverlay(new GMarker(gateway, markerOptions));
18
19     kingsdale = new GLatLng(40.023309,-83.058443);
20     map.addOverlay(new GMarker(kingsdale, markerOptions));
21
22     georgesville = new GLatLng(39.919828,-83.12046);
23     map.addOverlay(new GMarker(georgesville, markerOptions));
24
25   }
26 }
```

Let's break down this code:

- Line 05 instantiates a **GIcon** object. Following instantiation, we assign two images to the object's **image** and **shadow** attributes, using the **joystick.png** and **joystick-shadow.png** graphics, respectively. Lines 08-09 define the sizes for these two images, so the API knows how to properly size the images on the map.
- Line 11 tells the API to use the new icon in place of the default.
- Lines 17, 20, and 23 each use the new icon, by passing in the **markerOptions** variable as a second parameter to **GMarker**.

Although this is all you need to replace the icon with one of your own, this listing demonstrates just a few of the icon-specific features available via the API. Consult the API documentation for more information.

To create your own icons, check out the Sib icon editor, available for free download at <http://www.sibcode.com>. Also consider installing the GNU Image Manipulation Program, better known as GIMP (<http://www.gimp.org/>). But beyond recommending these great programs, the best advice I can give you is to experiment until you obtain the desired visual outcome. For instance, the joystick icon is 32x32 pixels, as is the shadow, however in the shadow's offset has been adjusted a bit in order to achieve a visually pleasing result. Also, you'll want to make sure the shadow is created with an

opacity setting of roughly 50% and situated above a transparent background so the map contents can be seen beneath. Of course, you can avoid the creation process altogether by taking advantage of the numerous icon libraries available for free and through purchase online. See <http://www.iconarchive.com> and <http://www.iconshock.com> for just a few great examples of what's available. There are also several sites devoted to icons created specifically for mapping purposes. See <http://www.cartosoft.com> and <http://www.gmaplive.com> for some great examples.

Adding an Information Window

As a final exercise in this introduction to the Google Maps API, I'll show you how to link an informational window to each marker, offering the user more information about the location should the location's icon be clicked. As shown in Figure 9-12, these informational windows are like mini HTML pages, capable of rendering any standard HTML you provide.

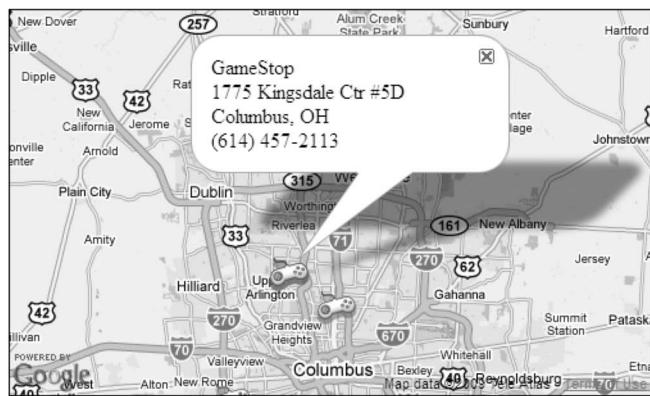


Figure 9-12. Including contact information in an information window

Adding an information window is fairly straightforward, as Listing 9-2 demonstrates.

Listing 9-2. Creating an informational window

```

01 function load() {
02   if (GBrowserIsCompatible()) {
03     var map = new GMap2(document.getElementById("map"));
04
05     var joystickIcon = new GIcon(G_DEFAULT_ICON);
06     joystickIcon.image = "http://www.gamenomad.com/images/joystick.png";
07     joystickIcon.shadow = "http://www.gamenomad.com/images/joystick-shadow.png";
08     joystickIcon.iconSize = new GSize(32,32);
09     joystickIcon.shadowSize = new GSize(32,32);
10
11     joystickIcon.infoWindowAnchor = new GPoint(20,10);
12
13     markerOptions = { icon:joystickIcon };
14
15     center = new GLatLng(39.995649,-83.007116);
16     map.setCenter(center, 10);

```

```

17
18     kingsdale = new GLatLng(40.023309,-83.058443);
19     map.addOverlay(new GMarker(kingsdale, markerOptions));
20
21     kingsdale = new GLatLng(40.023309,-83.058443);
22     kingsdaleMarker = new GMarker(kingsdale, markerOptions);
23     map.addOverlay(kingsdaleMarker);
24
25     var info = "GameStop<br />1775 Kingsdale Ctr #5D<br />Columbus, OH
26             <br />(614) 457-2113";
27     GEvent.addListener(kingsdaleMarker, 'click', function() {
28         kingsdaleMarker.openInfoWindow(info);
29     });
30 }
31 }
```

Let's breakdown this code:

- Line 11 adjusts the location in which the window will appear relative to the top left corner of the icon it's attached to. I wanted the window tip to appear almost at the box' apex and so have moved it 20 pixels to the right of its default location, and 10 pixels downwards.
- Because the window must attach to a named marker object, lines 22-23 create this named object and add the marker to the map. I raise this point because you might notice the approach is slightly different than what was used in previous examples, so please take a moment to compare the two methods.
- Line 25 defines the content which will appear in the message window. Remember you can use any valid HTML in the window, including hyperlinks and images.
- Finally, lines 27-29 connect the window to the marker by attaching an event listener. In this case, clicking on the marker will open the window. However, clicking is just one of several events available to you. For instance, you could open the window when mousing over the icon by replacing `click` with `mouseover`.

Step #2. Adding Mapping Services to Your Website

Interacting with the Google Maps API via the JavaScript library is a great way to familiarize oneself with its fundamental features. As Zend Framework users though, we're all about technology consolidation. Just as we use the `Zend_Db` component to communicate with the database using PHP, it seems natural we'd also want a way to do the same using the Google Maps API, rather than have to deal with passing messages between JavaScript and PHP. While there isn't (yet) a Zend Framework component available for communicating directly with the API, you can easily integrate a third party library into your website which will take care of the integration for you. This library is called the `GoogleMapAPI`, and it's introduced next.

Introducing GoogleMapAPI

GoogleMapAPI is an impressive open source solution for building Google maps using familiar object-oriented PHP syntax. Created by Monte Ohrt (<http://www.phpinsider.com/>), and available under an open source license, GoogleMapAPI removes the need to mix JavaScript and PHP, instead relying on the class to write the JavaScript for you.

Installing and Configuring GoogleMapAPI

To install GoogleMapAPI, head over to <http://www.phpinsider.com/php/code/GoogleMapAPI/>, unzip the `.tar.gz` file, and move the `GoogleMapAPI.class.php` file to a convenient location within your application directory (I prefer placing third-party libraries within `application/library/third-party`). Don't forget to add this directory to your application's include path within the bootstrap file (`application/public/index.php`):

```
set_include_path('..library/third-party' . PATH_SEPARATOR . get_include_path());
```

Next, add a parameter to the `config.ini` file for maintaining your Google Maps API key. I also add a default latitude and longitude coordinates so the map can center on a preset location when the circumstances warrant doing so:

```
; Google parameters
google.map_key          = YOUR_SECRET_API_KEY_Goes_HERE
google.default_latitude  = 39.981856
google.default_longitude = -83.045053
```

Once installed and configured, let's move on to creating our first GoogleMapAPI-driven map!

Creating Your First GoogleMapAPI-driven Map

Put in general terms, to create a GoogleMapAPI-driven map you need to do three things. First, add the header JavaScript used by GoogleMapAPI to create the maps. Second, you'll instantiate the `GoogleMapAPI` class and use the class' methods to orient the map, set an appropriate zoom level, set icons, and informational windows. Finally, you'll render the map within the view. Let's start with the first task, adding the following two lines in between the `<head></head>` tags of the `layout.phtml` file:

```
<?php $this->map->printHeaderJS(); ?>
<?php $this->map->printMapJS(); ?>
```

Of course, because we're referencing `$this->map` within the layout view, a corresponding `$this->view->map` object must be created within a controller. If you only plan on using maps within a specific controller, you can limit the need for this object by creating it within the controller's `init()` method:

```
$this->view->map = new GoogleMapAPI('map');
$this->view->map->setAPIKey($this->config->google->params->map_key);
```

Of course, this also means the `$this->map` object will not exist when the layout is used in conjunction with other controllers, so I wrap the JavaScript calls like so within the `layout.phtml` file:

```
<?php if (Zend_Controller_Front::getInstance()->
    getRequest()->getControllerName() == "marketplace") { ?>
<?php $this->map->printHeaderJS(); ?>
<?php $this->map->printMapJS(); ?>
<?php } ?>
```

This results in the extraneous JavaScript being omitted from the pages where maps won't be used. Of course, if you planned on using maps within multiple controllers, such an approach could quickly become difficult to maintain. In most cases, loading the additional JavaScript into every page won't result in any significant performance degradation anyway; I just thought you might appreciate this particular approach, as it has its occasional application elsewhere. Alternatively, you could make the `$this->map` object available to all controllers and actions by initializing it within an action helper such as the Initializer helper discussed in Chapter 5.

You'll also need to add the `onload()` and `onunload()` event handlers to the `<body>` tag within your layout. As was discussed earlier in this chapter, the Google Maps API requires this method to execute the JavaScript used to create the maps, and properly clean up the resources allocated during the maps' usage, respectively. Again, if you'd like to consolidate the API JavaScript to a single controller, you could use the aforementioned `getInstance()` approach to determine which controller was being used, and execute this event handler accordingly.

Next, within the controller, add the following lines to the desired action:

```
01 $this->view->map->setCenterCoords($this->config->google->default_longitude,
                                         $this->config->google->default_latitude);
02 $this->view->map->setZoomLevel(10);
```

In this example, line 01 centers the map over a particular location. I used the default latitudinal and longitudinal values stored in the `config.ini` file. Line 02 sets the zoom level, which at 10 will be sufficient to give the user a bird's eye view of a large city.

Finally, all you need to do is call a single line from within the action's controller:

```
<?= $this->map->printMap(); ?>
```

Executing the action will produce a map identical to that shown in Figure 9-13.

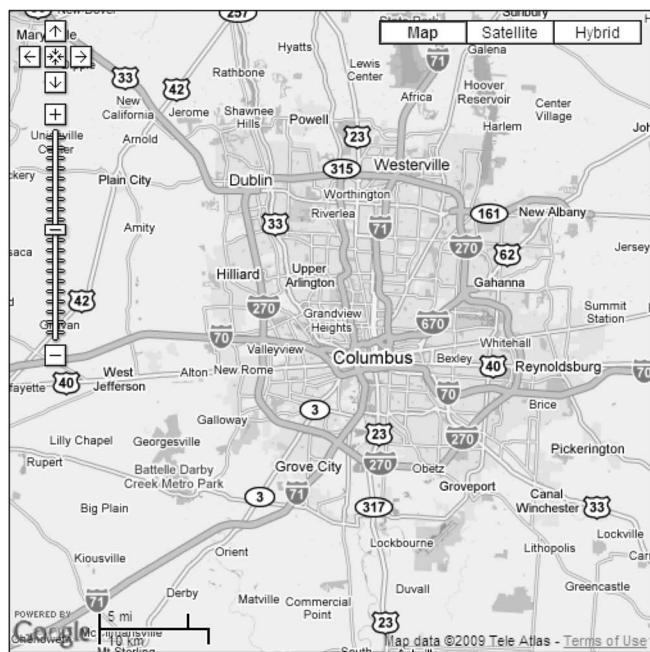


Figure 9-13. Creating a map with the GoogleMapAPI class

Converting an Address to Coordinates

As you've seen, the Google Maps API requires a set of coordinates to function properly. Fortunately, you can retrieve these coordinates (known as *geocoding*) with a simple call to the `getGeocode()` method:

```
$coords = $map->getGeocode('1373 Grandview Avenue Columbus Ohio 43212');
```

The return value is an associative array containing two keys: `lat` and `lon`. Therefore to retrieve the coordinates, use the array like so:

```
echo "Office location: latitude {$coords['lat']} and longitude {$coords['lon']}.";
```

Returning:

```
Office location: latitude 39.9857680 and longitude -83.0448350.
```

Thankfully, the API is very forgiving in terms of how you enter the address. Any of the following versions are parsed without any problems:

```
1373 Grandview Avenue, Columbus, Ohio 43212
1373 Grandview Avenue, Columbus Ohio 43212
1373 Grandview Avenue 43212
```

Note in particular the last example, as it lacks the city and state. In many cases you won't have a

complete mailing address available, however the API is capable of geocoding to a level of accuracy correspondent to the data on hand. For instance, GameNomad currently does not ask the user to enter his mailing address, asking only for the city, state, and zipcode. Following the usual filtering to confirm the data is valid, this information is geocoded like so:

```
if ($this->_request->getPost('state_id') != "") {
    $coords = $map->geoGetCoords($this->_request->getPost('zipcode'));
} else {
    $country = new Country();
    $cnt = $country->getCountryById($this->request->getPost('country_id'));
    $coords = $map->geoGetCoords($this->_request->getPost('city'), $cnt->name);
}
```

Notice how in the case the user is based in the United States, only the zip code is used to carry out the geocoding. In the case of a foreign user, only his country is used because the Google Maps API does not yet offer comprehensive geocoding support for all countries.

Storing Coordinates in the Database

Because you'll likely be mapping certain addresses repeatedly, it makes sense to store coordinates in the database. Use the FLOAT type with a size of 10 and a precision of 6. You can make this change using phpMyAdmin or the mysql client. If you use the latter, the commands will look like this:

```
%>ALTER TABLE users ADD COLUMN latitude FLOAT(10,6) NOT NULL;
%>ALTER TABLE users ADD COLUMN longitude FLOAT(10,6) NOT NULL;
```

Adding Markers and Information Windows

Adding a marker using the GoogleMapAPI library is easy. In fact there are several ways to go about doing so. If you only have the address available, you can pass it to the `addMarkerByAddress()` method, which will automatically do the geocoding prior to creating the marker. For instance, to add a marker identifying the 43212 zip code:

```
$this->view->map->addMarkerByAddress('43212');
```

This will produce a marker placed over the city (Grandview Heights) identified by the 43212 zip code, as shown in Figure 9-14.



Figure 9-14. Adding a marker using the `addMarkerByAddress()` method

To add a marker using coordinates, use the `addMarkerByCoords()` method:

```
$this->view->map->addMarkerByCoords(-83.0448350, 39.9857680);
```

Adding Information Windows

To add an information window which will appear when the user clicks on the marker, just pass a title and window text to the `addMarkerByAddress()` or `addMarkerByCoords()` methods. The result of the following example is shown in Figure 9-15.

```
$message = "12 GameNomad members reside in Grandview Heights, Ohio.";
$this->view->map->addMarkerByCoords(-83.0448350, 39.9857680,
                                         "Grandview Heights, Ohio", $message);
```

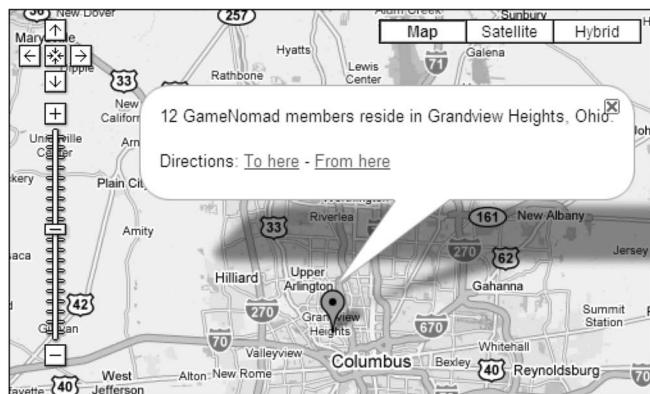


Figure 9-15. Attaching an informational window to a marker

Notice the unexpected addition of links regarding obtaining directions to or from the marker's location. This feature is automatically enabled by GoogleMapAPI. The user can click on the appropriate link, enter an address, and another browser window will open to the <http://maps.google.com> site, with the directions calculated. If you would like to disable this feature, use the `disableDirections()` method.

Other Key Features

Rather than exhaustively recreate many of the examples already introduced in this chapter, I've elected to instead provide you with an overview of the GoogleMapAPI class' key methods, summarized in Table 9-1. This is not intended to be a complete list, so be sure to consult the GoogleMapAPI online documentation for an exhaustive summary.

Table 9-1. The GoogleMapAPI class' commonly used methods

| Method | Description |
|---|--|
| <code>disableMapControls()</code> | Prevent the zoom and move map controls from rendering. |
| <code>enableMapControls()</code> | Render the zoom and move map controls. Enabled by default. |
| <code>getGeocode(\$address)</code> | Retrieves the coordinates of the given address |
| <code>setControlSize(\$size)</code> | Changes the map control size. Setting this to 'large' will include the slider along with the zoom and move controls. Setting it to 'small' will omit the slider. By default the control size is set to large. |
| <code>setHeight(\$height)</code> | Sets the height of the map window, in pixels or in terms of a percentage of the total window size. For instance a value of '300px' would set the map height to 300 pixels, whereas '75%' would create a map of a height equal to 75% of the browser window height. |
| <code>setInfoWindowTrigger(\$type)</code> | Determines how a marker's information window will open. Setting \$type to 'click' will cause the window will open when the marker is clicked (the default). Setting \$type to 'mouseover' will cause it to open when the user passes his mouse over the marker. |
| <code>setMapType(\$type)</code> | Determines the map type. Supported values include 'map', 'satellite', and 'hybrid'. |
| <code>setWidth(\$width)</code> | Sets the width of the map window, in pixels or in terms of a percentage of the total window size. For instance a value of '300px' would set the map width to 300 pixels, whereas '75%' would create a map of a width equal to 75% of the browser window width. |
| <code>setZoomLevel(\$level)</code> | Sets the map's zoom level. Setting \$level to 0 will show the globe on a single map, whereas 19 will offer the finest resolution. |

Conclusion

Spatially-driven websites are getting more popular by the day, and hopefully this chapter shed some insight into why users find them so compelling. With the Google Maps API playing a key role in this exciting new movement, countless developers are blogging about their efforts and extending the API in impressive ways. So be sure to consult another popular Google feature, namely the Google search engine, seeking out ideas and other useful mapping API utilities!

In the next chapter we'll look at another service which has changed the face of the web, namely the Amazon Associates Web Service. Using this popular solution you can build a product catalog of interest to your users, and earn revenues through affiliate fees!

CHAPTER 10

Introducing the Amazon Associates Web Service

Regardless of whether your site discusses books, DVDs, fantasy football teams, or video games, you're going to be judged by the quality of the data you keep. For instance, chances are GameNomad users aren't going to stick around for long if game titles are misspelled, prices are incorrect, or cataloged games aren't properly matched with their supported platforms. At the same time, part of your success is going to hinge upon how quickly you can build a large content catalog; after all, prospective users aren't going to be particularly compelled to register if there's not much to read or otherwise use on your website. So your dilemma is really two-fold. Not only do you need to assemble a large array of content as quickly as possible, but you also need to ensure its accuracy. To be certain, this isn't an easy task, particularly when you're already spending the bulk of your time developing the website.

Thankfully, there are numerous services which can help you accomplish both goals, no matter what your website's particular product focus may be. These services, better known as *web services*, open vast troves of highly-organized data to developers interested in mining, analyzing, or presenting this data in new and interesting ways. In fact, in the last chapter you were using a web service and perhaps didn't even know it. The Google Maps API gives developers access to Google's massive store of geographical data, and will even present that data via it's by now abundantly familiar interface.

With some knowledge and creativity, you can integrate web services into your own websites, building your own compelling services which can rely upon and even enhance these organizations' data stores. Want to build a movie review website? Look to the Netflix API (<http://developer.netflix.com/>) to seed your movie catalog. Dreamed up a compelling way to submit and monitor items for auction on eBay? Check out the eBay API at <http://developer.ebay.com/>. Think you can help music lovers more effectively explore the latest releases? The Last.fm API (<http://www.last.fm/api>) might be exactly what you need to get started.

In this chapter I'm going to introduce you to the web service that makes GameNomad games database tick. The Amazon Associates Web Service is Amazon.com's eponymous web service which opens up the company's enormous database to developers interested in advertising new and used products on their own websites. Developers can use this solution to build custom catalogs around their favorite product category, be it books, toys, groceries, shoes, or video games! By linking these products to Amazon.com, developers can earn a referral fee for every product sold as a result of a user clicking through and purchasing the product on Amazon.com.

CAUTION. You might be wondering why organizations go through the trouble of giving third-party developers access to what most would consider the corporate golden goose, namely the enormous store of well-manicured data. The answer is simple: profit generation. These organizations often require your website to be primarily concerned with sending customers their way, either due to your website raising awareness about a particular product (such as a newly released DVD) or due to an affiliate link clicked upon by an interested user. That said, be sure you're adhering to the API's usage guidelines by seeking out and carefully reading its Terms of Use policy.

Chapter Steps

The goals of this chapter are accomplished in four steps:

- **Step #1. Introducing the Amazon Associates Web Service:** In the chapter's opening step we'll talk more about the Amazon Associates Web Service, and discuss the advantages of creating an Amazon affiliate account. To round out the step, I'll show you how to create an Amazon Associates Web Service account, Amazon Associates Account, and ready your Zend Framework-powered website to begin accessing the Amazon Associates Database.
- **Step #2. Introducing the Zend_Services_Amazon Component:** In this step I'll show you how query the Amazon database using the Zend Framework's Zend_Services_Amazon component. You'll learn how to retrieve product attributes such as prices, images, sales ranks, descriptions, and other data crucial to building a great affiliate service.
- **Step #3. Searching for Products:** In the third step we'll build upon what was learned in Step #2, building more complex queries to mine the Amazon database in new and interesting ways.
- **Step #4. Customer Reviews:** Unbiased customer reviews can hold tremendous sway over a prospective buyer's purchasing decisions. In the fourth and final step of this chapter you'll learn how to retrieve and sort a product's reviews. Along the way we'll create a view helper for turning an integer-based rating value into a string of star icons.

VIDEO: A Tour of Amazon's Web Services

The Amazon Associates Web Service is just one of the several fascinating web services offered by Amazon.com and used by thousands of organizations large and small around the globe. This video introduces you to these services. Watch the video at <http://www.easyphpwebsites.com/zfw/videos/>.

Step #1. Introducing the Amazon Associates Web Service

Having launched their Associates program back in 1996, before much of the world had even heard of the Internet, Amazon.com founder Jeff Bezos and his colleagues clearly grasped at a very early point the power of Web linking. By providing motivated third-parties, known as *associates*, with an easy way to promote Amazon products on their own websites and earn a percentage of any sales occurring as a result of their efforts, Amazon figured they could continue to grow market share in the fledgling e-commerce market. Some 13 years later, the Amazon Associates program is a true online juggernaut, with everybody from large corporations to occasional bloggers taking advantage of the program to enhance revenues.

With over a decade of experience under their belts, Amazon has had plenty of time and opportunities to nurture their Associates program. Early on in the program's lifetime, associates' options were limited to the creation of banners and other basic links, however presently options abound, with associates provided with a wealth of tools for linking to Amazon products using links, banners, widgets, search engines. Users are also provided with powerful sales analysis tools which help them understand what efforts are working and what aren't.

Along the way, they also unveiled the shining gem of the associates program: the *Amazon Associates Web Service*. This service opened up Amazon's enormous product database to developers, giving them the wherewithal to retrieve and manipulate this data in new and creative ways. What's particularly impressive about this service is that Amazon didn't make a half-hearted effort at opening up the database, for instance allowing access to only the product titles and descriptions. Developers have access to *all* of the data they could conceivably need to build a fascinating new solution for perusing Amazon products. Among other things, developers have access to product titles, ASINs (Amazon's internal version of the UPC code), product release dates, prices, manufacturer names, Amazon sales ranks, customer and editorial reviews, product relations (products identified as being similar to one another), images, and much more!

Access to the database is provided by way of a Web Service, which is just a fancy way of calling an API over the Internet. As mentioned, the Google Maps API introduced in the last chapter is an example of a Web service which exposed Google's mapping data via a programmatic interface. Likewise, the Amazon Associates Web Service exposes their product listings through a similar interface. Later in this chapter we'll communicate with that interface using the Zend Framework's `Zend_Service_Amazon` component.

But before you can begin taking advantage of this fantastic service, you'll need to register for two accounts: an Amazon Associates account and an Amazon Associates Web Service account. I'll show you how to do this next.

Joining the Amazon Associates Program

Joining the Amazon Associates Program is free, and only requires you to complete a short registration form in which you'll provide your payment and contact information, web site name, URL and description, in addition to declare agreement to the Amazon Associates operating agreement. To register for the program, head over to <https://affiliate-program.amazon.com/> and click on the **Join now for FREE!** button to start the process.

There's no waiting period following registration, and in fact you'll be immediately provided with your unique Associate ID following submission of the registration form. As you'll soon learn, you'll attach this associate ID to the product URLs so Amazon knows to what account they should credit the potential purchase. At this point you'll also be prompted to identify how you'd like to be paid, either by direct deposit, check, or Amazon gift card.

Creating an Amazon Associates Web Service Account

To gain access to Amazon's database and begin building your catalog, you'll need to create an account. The end result of this free registration process is the provision of an access key, which you'll use to communicate with the service. To obtain this key, head over to <http://aws.amazon.com/> and click the **Create an AWS Account** link located at the top right of the page. You'll be asked to sign in to your existing Amazon.com account, and provide contact information, your company name or website, and the web site URL where you'll be invoking the service. You'll also be asked to read and agree to the AWS Customer Agreement. Once done, Amazon will send a confirmation e-mail. You can then retrieve your key by heading back to <http://aws.amazon.com> and clicking on the **Access Identifi-**

fiers menu item located within the Your Account tab found at the top of the page. Copy the identifier named Your Access Key ID and paste it into your config.ini file, along with your associate ID:

```
; Amazon  
amazon.key = 12345678ABCDEFGHIJK  
amazon.associate_id = gamenomad
```

We'll use this access key to connect to Amazon using the Zend_Service_Amazon component, introduced in the next step.

Creating Your First Product Link

Although the point of this chapter is to show you how to dynamically create product links tied to your Amazon Associate account, it's worth taking a moment to understand the components of these links so you'll understand why we're doing this in a particular way later in this chapter. Proper functioning of your Amazon Associate's account is dependent upon the proper inclusion of your Associate ID within the product link. When using Amazon's automated wizards for creating product links (head over to <https://affiliate-program.amazon.com/> to learn more about these) you'll find these links to be extremely long and decidedly not user-friendly. However, they support a shortcut which allows you to create succinct alternative versions. For instance, the following link will point users to Amazon's product detail page for the video game Halo 3 for the Xbox 360, tying the link to GameNomad's affiliate account:

```
http://www.amazon.com/exec/obidos/ASIN/B000FRU0NU/gamenomad-20
```

As you can see, this link consists of just two pieces of dynamic information: the product's ASIN, and the associate identifier. Don't believe it's this easy? Head on over to the Amazon Associates Link Checker (<https://affiliate-program.amazon.com/gp/associates/network/tools/link-checker/main.html>) and test it out. Enter the link into the form (you'll need to swap out my Associate ID with your own), and press the Load Link button. The page will render within an embedded frame, confirming you're linking to the appropriate product. Once rendered, click the Check Link button to confirm the page is linked to your associate identifier.

Of course, you'll probably want to include much more than a mere link. Using the Amazon Associates Web Service, we can do this on a large scale. This topic is the focus of the remainder of this chapter.

Step #2. Introducing the Zend_Service_Amazon Component

The Zend_Service_Amazon component offers a convenient way to talk to the Amazon Associates Web Service using an object-oriented interface. Using this component, you'll be able to query Amazon.com's database for product information such as descriptions, images, manufacturer names, and prices. You'll also be able to perform more sophisticated queries capable of retrieving users' product reviews, related products, and even users' Listmania! lists.

Interestingly, this component actually includes two separate APIs, which the Zend Framework developers refer to as the *Traditional API* and the *Query API*. While equally capable in terms of their

ability to query the Amazon database, I prefer the Query API due to its particularly intuitive syntax. Accordingly, we'll be using solely the Query API throughout this chapter, although I invite you to at least get acquainted with the Traditional API's syntax by checking out the Zend_Service_Amazon documentation at <http://framework zend.com>.

Per usual, my preferred methodology for learning a technology is by actively experimenting with it. Let's work through a variety of examples involving querying the Amazon product database using this component.

Retrieving a Single Video Game

Amazon.com has long used a custom product identification standard known as the Amazon Standard Identification Number, or ASIN. We can use these 10-digit alphanumerical strings to retrieve a specific product. Of course, you need to know what the product's ASIN is in order to perform such a query, but how do you find it? You can either locate it within the product's URL, or by scrolling down the product's page where it will be identified alongside other information such as the current Amazon sales rank and manufacturer name. For instance, the ASIN for *Halo 3* on the Xbox 360 is B000FRU0NU. With that in hand, we can use the Zend_Services_Amazon component to query Amazon. Within an appropriate controller action add the following code:

```
01 $amazon = new Zend_Service_Amazon_Query($this->config->amazon->key);
02 $amazon->Asin('B000FRU0NU');
03 $this->view->item = $amazon->search();
04 echo "Title: {$this->view->item->Title}<br />";
05 echo "Publisher: {$this->view->item->Manufacturer}<br />";
06 echo "Category: {$this->view->item->ProductGroup}";
```

Although I'd venture a guess this code is self-explanatory, let's nonetheless expand upon some of its finer points:

- Line 01 creates the connection to the Amazon Web Services service, authenticating the user by passing in the assigned access key. Note how the key is retrieved from the configuration file.
- Line 02 indicates we're searching for a specific item as identified by the ASIN B000FRU0NU. As you'll see later in this chapter, we can also perform open-ended searches using criteria such as product title and manufacturer.
- Line 03 performs the search, returning an object which you can subsequently parse and display in the view, or use as the basis for further manipulation in the action.
- Lines 04-06 output the returned product's title, manufacturer, and product group. You can think of the product group as an organizational attribute, like a category. Amazon has many such product groups, including *Books*, *Video Games*, and *Sporting Goods*.

Executing this code returns the following output:

Title: Halo 3
 Publisher: Microsoft
 Category: Video Games

Creating the Affiliate URL

By default, an attribute named `DetailPageURL` was returned along with the other attributes demonstrated in the previous example. However, this URL is not only decidedly user-unfriendly, but it logically does not include your Associate ID. However creating your own custom URL based on the URL template shown earlier in this chapter is easy. Just use the `ASIN` attribute and the `amazon.associate_id` configuration parameter to construct the URL:

```
echo "http://www.amazon.com/exec/obidos/ASIN/{$this->view->item->ASIN}/
      {$this->config->amazon->associate_id}";
```

Executing this line will produce the following URL:

```
http://www.amazon.com/exec/obidos/ASIN/B000FRU0NU/gamenomad-20
```

Setting the Response Group

To maximize efficiency both in terms of bandwidth usage and parsing of the returned object, Amazon empowers you to specify the degree of product detail you'd like returned. When it comes to querying for general product information, typically you'll choose from one of three levels:

- **Small:** The Small group (set by default) contains solely the most fundamental product attributes, including the ASIN, creator (author or manufacturer, for instance), manufacturer, product group (book, video game, or sporting goods, for instance), title, and Amazon.com product URL.
- **Medium:** The Medium group contains everything found in the Small group, in addition to attributes such as the product's price, editorial review, current sales rank, the availability of this item in terms of the number of new, used, collectible, and refurbished units made available through Amazon.com, and links to the images.
- **Large:** The Large group contains everything found in the Medium group, in addition to data such as a list of similar products, the names of tracks if the product group is a CD, a list of product accessories if relevant, and a list of available offers (useful if a product is commonly sold by multiple vendors via Amazon.com). Hopefully it goes without saying that if you're interested in retrieving just the product's fundamental attributes such as the title and price, you should be careful to choose the more streamlined Medium group , as the Large group comes at the cost of significant extraneous bandwidth for such purposes.

If you're interested in retrieving only a specific set of attributes, such as the image URLs or customer reviews, then you might consider one of the many specialized response groups at your disposal.

Among these response groups include `Images`, `SalesRank`, `CustomerReviews`, and `EditorialReview`. As an example, if you'd like to regularly keep tabs of solely a product's latest Amazon sales rank, there's logically no need to retrieve anything more than the rank. To forego retrieving superfluous data, use the `SalesRank` response group:

```
$amazon = new Zend_Service_Amazon_Query($this->config->amazon->key);
$amazon->Asin('B000FRU0NU')->ResponseGroup('SalesRank');
$this->view->item = $amazon->search();
echo "The latest sales rank is: {$this->view->item->SalesRank}";
```

NOTE. Determining which attributes are available to the various response groups can be a tedious affair. To help sort through the details, consider downloading the documentation from <http://aws.amazon.com/documentation/>. Also, I've found the AWS Zone website (<http://www.awszone.com/>) to be very useful, as you can use a Web-based tool to experiment with the various groups and review the results in XML format.

Displaying Product Images

Adding an image to your product listings can greatly improve the visual appeal of your site. If your queries are configured to return a **Medium** or **Large** response group, URLs for three different image sizes (available via the **SmallImage**, **MediumImage**, and **LargeImage** objects) are included in the response. Of course, unless you require something else only available within the **Large** response group, use the **Medium** group, as demonstrated here:

```
$amazon = new Zend_Service_Amazon_Query($this->config->amazon->key);
$amazon->Asin('B000FRU0NU')->ResponseGroup('Medium');
$this->view->item = $amazon->search();
echo $this->view->item->SmallImage->Url;
```

Executing this code returns the following URL:

```
http://ecx.images-amazon.com/images/I/51atrAxEV1L._SL160_.jpg
```

Of course, you'll want to include the image within the view. To do so, all you need to do is pass the URL into an **** tag, like so:

```
_request->getParam('asin');

    // Query AWS
    $amazon = new Zend_Service_Amazon_Query($this->config->ws->amazon_key);
    $amazon->Asin($asin)->ResponseGroup('Medium');
    $this->view->item = $amazon->search();

    // Assign the Associate ID to the view for later use
    $this->view->amazonAssociateID = $this->config->ws->amazonAffiliateAccount;
}
```

Once the query has been returned, all that's left to do is populate the data into the view, as is demonstrated here:

```
01 <span id="game-profile-left">
02
03   <p>
04     
12  </p>
13
14 </span>
15
16 <span id="game-profile-right">
17   <h1><?= $this->item->Title; ?></h1>
18   <p>
19     <b>Platform</b>: <?= $this->item->Platform; ?><br />
20     <b>Release date</b>:
21     <?= date("F d, Y", strtotime($this->item->ReleaseDate)); ?>
22     <br />
23     <b>Latest Price</b>: <?= $this->item->FormattedPrice; ?><br />
24     <b>Latest Sales Rank</b>: <?= number_format($this->item->SalesRank); ?>
25   </p>
26 </span>

```

Like the controller, we're really just connecting the dots regarding what's been learned here and in other chapters. Let's highlight a few key lines in this view:

- Line 04 renders the video game's medium image. Remember you also have a small and large image at your disposal.
- Lines 08-11 create the Amazon Associate-enhanced Amazon.com product link.
- Lines 19-24 output information regarding the game, including the title, platform, release date, price, and sales rank. Note how we're using PHP functions to format this data in more user-friendly terms; the `strtotime()` function converts the date into a format capable of being recognized by the `date()` function, which subsequently converts the value into a user-friendly date. Finally, the `number_format()` function will format a number by grouping it into thousands and placing commas accordingly.

Of course, GameNomad being a live website, I don't actually actively query the web service every time a user would like to learn more about a game. Much of this data is cached locally, and regularly updated in accordance with the terms of service. Nonetheless, the above example nicely demonstrates how to use the web service to pull this data together.

Step #3. Searching for Products

All of the examples provided thus far presume you've an ASIN handy. But manually navigating through the Amazon.com website to find them is a tedious process. In fact, you might not even know the product's specific title, and instead just want to retrieve all products having a particular keyword in the title, or made by a particular manufacturer. In this step you'll learn how to perform searches which help you to learn more about the many products available via Amazon.com.

Searching for Products by Title

What if you wanted to find products according to a particular keyword found in the product title? To do so, you'll need to identify the product category, and then specify the keyword you'd like to use as the basis for searching within that category. The following example demonstrates how to search the `VideoGames` (note the lack of spaces) category for any product having the keyword *Halo* in its title:

```
$amazon = new Zend_Service_Amazon_Query($this->config->ws->amazon_key);

$amazon->Category('VideoGames')->Keywords('Halo');

$this->view->item = $amazon->search();

foreach($this->view->item AS $item) {
    echo "{$item->Title}<br />";
}
```

At the time of this writing (the contents of the Amazon.com catalog are of course subject to change at any time), executing this code produced the following output:

```
Halo: Combat Evolved
Halo 3
Halo, Books 1-3 (The Flood; First Strike; The Fall of Reach)
Halo: Combat Evolved
Halo Wars
Halo 2
Ghosts of Onyx (Halo)
Halo Wars Limited
Halo 2
The Halo Graphic Novel
```

It's worth pointing out that the ten products found in the listing aren't all video games, as the defined category might lead you to believe. For instance, *The Halo Graphic Novel* is a graphic novel (comic book). Why these sorts of seeming inconsistencies occur isn't apparent, although one would presume it has to do with making the product more easily findable on the Amazon.com website and through other outlets.

Incidentally, `VideoGames` is just one of more than 40 categories at your disposal. Try doing searches using categories such as `Music`, `DigitalMusic`, `Watches`, `SportingGoods`, `Photo`, and `OutdoorLiving` for some idea of what's available!

Executing a Blended Search

If you were creating a website dedicated to the *Halo* video game series, chances are you'd want to list much more than just the games! After all, there are *Halo*-specific books, soundtracks, toys, action figures, and even an animated series. But not all of these items are necessarily categorized within `VideoGames`, so how can you be sure to capture them all? You might be tempted to do this:

```
$amazon->Category('VideoGames')->Category('Music')->Keywords('Halo');
```

While logical, the behavior isn't what you'd expect, because the `Music` category designation will just overwrite `VideoGames` rather than be coupled with it. You can however use what's known as a *blended search*. Setting `Category` to `Blended` will result in a combined search of the categories `Apparel`, `Books`, `DVD`, `Electronics`, `GourmetFood`, `Kitchen`, `PCHardware`, `Software`, `SportingGoods`, `Tools`, `Toys`, and `VideoGames`. Although not perfect (at the time of writing performing a blended search using the keyword `Halo` turns up among other things a listing for Nissan Maxima fog lights). Revising the previous line, here's how you'd perform the blended search:

```
$amazon->Category('Blended')->Keywords('Halo');
```

Performing the search anew turns up almost 50 items with *Halo* in the title, the vast majority of which are clearly related to the popular video game brand.

Step #4. Customer Reviews

Amazon's customer reviews undoubtedly have enormous sway over prospective buyers, not to mention can be a great catalyst in terms of sparking debate and springboards for further research among interested readers. You can retrieve the latest five reviews for a particular product simply by identifying the ASIN and the `Review` response group:

```
$amazon = new Zend_Service_Amazon_Query($this->config->ws->amazon_key);
$amazon->ASIN('B000FRU0NU')->ResponseGroup('Reviews');
$this->view->reviews = $amazon->search();
```

The corresponding view looks like this:

```
<h1>The Latest Halo 3 (Xbox 360) Reviews</h1>
<?php foreach($this->reviews->CustomerReviews AS $review) { ?>
    <h3><?= $review->Summary; ?></h3>
    <p>
        <?= $review->Rating; ?> stars<br />
        Posted on <?= date("F d, Y", strtotime($review->Date)); ?><br />
        <?= $review->Content; ?>
    </p>
<?php } ?>
```

Executing this code produces output similar to that shown in Figure 10-2.

The Latest Halo 3 (Xbox 360) Reviews**4 Stars - Not so epic.**

Posted on February 01, 2009

This was the second greatest game in the trilogy, but it lacked length. It took me one day to beat this game and honestly, I was disappointed. I still like Halo 1 better than both 2 and 3. The game is still great, but I do think that if you are new to the series, or even new to shooter games, you should get Halo 1.

5 Stars - Nice trilogy..

Posted on January 30, 2009

A very good third episode of the Halo trilogy. Excellent graphics, the game is a bit short compared with HALO 2, but provides hours of excellent entertainment. If you have an xbox360 you must have this title in your collection.

5 Stars - Amazing game

Posted on January 30, 2009

I've been playing the other two parts of this amazing saga and I love this one. It gets very cool graphics.

5 Stars - Great Online experience

Posted on January 28, 2009

This game is awesome...the single player campaign falls short of the multiplayer experience. That is where this game rocks!!!!

5 Stars - Awsome game, short campaign

Posted on January 25, 2009

If you play call of duty 1-4, you'll notice the first time you play halo 3 that it's way more complex--and fun. You can not own a 360 without halo 3!

Figure 10-2. Displaying the latest Amazon customer product reviews

Paging Reviews

If you ran the code in the previous example, you noticed only five reviews were returned, regardless of how many actually appear on the product's corresponding Amazon.com page. This is because the query by only returns five items per request. To retrieve additional reviews, you'll need to page through these results using the `ReviewPage()` method, like so (this example will return the second page of results):

```
$amazon->ASIN('B000FRU0NU')->ResponseGroup('Reviews')->ReviewPage(2);
```

To determine how many pages are available, reference the `TotalReviews` attribute:

```
echo "Total reviews: {$this->view->reviews->TotalReviews}";
```

Although you can't change the number of reviews returned per page, there's an easy workaround. Because no more than five items can be returned per request, just divide `TotalReviews` by 5. You'll be able to use this number as an upper boundary when navigating from one page to the next.

Sorting Reviews

When querying for product reviews, the default order is descending according to the most recently posted review. You can however change this sort order using other criteria. For instance, to sort returned reviews in descending order according to the highest number of helpful votes received, use this command:

```
$amazon->ASIN('B000FRU0NU')->ResponseGroup('Reviews')->
    ReviewSort('-HelpfulVotes');
```

To sort according to least helpful votes, just remove the minus sign from the front of `HelpfulVotes`. You can also sort according to rating (`OverallRating`) and submission date (`SubmissionDate`).

Creating a View Helper for Rendering Stars

As you saw in the previous two examples, each reviewer's rating is identified by an integer value. But surely there is a more visually appealing way we can represent the ratings? As you learned in Chapter 5, a view helper can do a tremendous amount of work with surprisingly little code. Using the `Stars` helper shown below, we can convert this boring integer value into a string of star icons which considerably improve the page's design, as shown in Figure 10-3.

The Latest Halo 3 (Xbox 360) Reviews

 **Not so epic.**
Posted on February 01, 2009
This was the second greatest game in the trilogy, but it lacked length. It took me one day to beat this game and honestly, I was disappointed. I still like Halo 1 better than both 2 and 3. The game is still great, but I do think that if you are new to the series, or even new to shooter games, you should get Halo 1.

 **Nice trilogy..**
Posted on January 30, 2009
A very good third episode of the Halo trilogy. Excellent graphics, the game is a bit short compared with HALO 2, but provides hours of excellent entertainment. If you have an xbox360 you must have this title in your collection.

 **Amazing game**
Posted on January 30, 2009
I've been playing the other two parts of this amazing saga and I love this one. It gots very cool graphics.

 **Great Online experience**
Posted on January 28, 2009
This game is awesome...the single player campaign falls short of the multiplayer experience. That is where this game rocks!!!!

 **Awsome game, short campaign**
Posted on January 25, 2009
If you play call of duty 1-4, you'll notice the first time you play halo 3 that it's way more complex--and fun. You can not own a 360 without halo 3!

Figure 10-3. Using a view helper to enhance the page

The revised view code follows:

```
<h1>The Latest <?= $this->reviews->Title; ?> (Xbox 360) Reviews</h1>
<?php foreach($this->reviews->CustomerReviews AS $review) { ?>
    <h3><?= $this->Stars($review->Rating); ?>&nbsp;<?= $review->Summary; ?></h3>
    <p>
        Posted on <?= date("F d, Y", strtotime($review->Date)); ?><br />
        <?= $review->Content; ?>
    </p>
<?php } ?>
```

Next you'll find the `Stars` view helper, which translates the `Rating` value into the corresponding number of star icons.

```
01 class My_View_Helper_Stars extends Zend_View_Helper_Abstract
02 {
03
```

```

04  /**
05   * Converts rating integer to star icons
06   *
07   * @param integer $rating
08   * @param string $stars
09   */
10  public function Stars($rating)
11  {
12      $star = "<img src='/images/icons/star.png' />";
13      $stars = "";
14
15      for($i=1; $i <= $rating; $i++)
16      {
17          $stars .= $star;
18      }
19      return $stars;
20  }
21
22 }

```

Let's break down some code:

- Line 12 defines the location of the star icon. This icon happens to be part of the spectacular FAMFAMFAM Mini icon set, available at <http://www.famfamfam.com/lab/icons/min/>. Of course, for organizational reasons you might consider storing the image path in the `config.ini` file.
- Lines 15-18 cycle through the `for` loop a number of times equivalent to the value of the `$rating` input parameter. Each time it loops, another `` tag containing the `star.png` path will be appended to the `$stars` variable.
- Line 19 returns the `$stars` variable to the calling view.

I think this is an ideal example of the power of view helpers because such a small amount of code adds so much to the website.

Conclusion

Although a fair bit of ground was covered, this chapter provided but a mere introduction to what's possible using the powerful Amazon Associates Web Service. To learn more about this excellent technology, download the documentation at <http://aws.amazon.com/documentation/>, and browse the forums at <http://developer.amazonwebservices.com/connect/>.

The next chapter discusses how you can enhance your website with Ajax, a modern web development technique used to create highly interactive interfaces.

CHAPTER 11

Enhancing the User Interface with Ajax

There's no use hiding it; I hate JavaScript. Over the years, its frightful syntax and difficult debugging procedures have brought me to the sheer edge of insanity countless times. I'm not alone; the language is widely acknowledged for its ability to cause even the most level-headed of programmers to spew profanity. To put the scope of the frustration brought about by this language another way, consider the impressive record of non-violence espoused by the likes of John Lennon. I speculate his penchant for pacifism was at least in part attributed to his unfamiliarity with JavaScript.

Yet today there is really no way to call yourself a modern Web developer and not regularly use JavaScript. In fact, while you might over the course of time alternate between several Web frameworks such as Zend Framework, Grails (<http://www.grails.org/>) and Rails (<http://www.rubyonrails.org/>), it's almost a certainty that JavaScript will be the common thread shared by all projects. This is because JavaScript is the special sauce behind the techniques used to create highly interactive web pages collectively known as *AJAX*.

AJAX makes it possible to build websites which behave in a manner similar to desktop applications, which have traditionally offered a far more powerful and diverse array of user interface features such as data grids, autocomplete, interactive graphs, and drag-and-drop. Indeed, users of popular services such as Gmail, Flickr, and Facebook have quickly grown accustomed to these sorts of cutting-edge features. Accordingly, you'll want to integrate similar features into your websites as soon as possible in order to help attract and maintain an audience who has come to consider rich interactivity a requirement rather than a novelty.

This puts us in a bit of a quandary: coding in JavaScript is a drag, but we really need it. Further, because these days we need to use JavaScript to some degree within practically every website, we're tasked with figuring out how to maintain often large bits of code for subsequent reuse. Doesn't this sound a bit like the problems we've encountered in the early chapters of the book, which were subsequently resolved by looking to a framework? Perhaps not surprisingly, the answer to our JavaScript woes is... a JavaScript framework!

In this chapter I'll introduce you JavaScript and the Ajax development paradigm, focusing on the popular Prototype and Script.aculo.us JavaScript libraries, two solutions which are so easy to use they (almost) make JavaScript development fun! Along the way, we'll review several examples which demonstrate JavaScript's ability to perform tasks such as forms validation and visual effects.

Chapter Steps

The goals of this chapter are broken down just two steps:

- **Step #1. Introducing JavaScript:** In this opening step I'll introduce you to the JavaScript language. Rather than endure an exhaustive review of basic syntax, we'll instead focus on key JavaScript features which help you to quickly begin implementing modern interactive features into your website.

- **Step #2. Introducing Ajax:** In order to implement these features, I'll introduce you to Prototype, a powerful JavaScript framework which although not officially supported by the Zend Framework, integrates very well nonetheless. In this step I'll also introduce you to Script.aculo.us, which is a set of user interface-oriented JavaScript libraries which extend the Prototype framework in fascinating ways. In this step I'll show you why organizations such as NASA, Gucci, and Apple have embraced Script.aculo.us, and show you how to integrate it into the Zend Framework.

Step #1. Introducing JavaScript

Because JavaScript is interpreted and executed by the browser, you need to make that code available to the browser, done by either embedding it directly within the web page, or referencing a separate file much as you've done with CSS or images. The cleanest way to do so is using the latter approach, allowing you to maintain and reuse the code separate from the HTML file. The following example demonstrates how an external JavaScript can be referenced:

```
01 <html>
02   <head>
03     <script type="text/javascript" src="/javascript/myjavascript.js"></script>
04   </head>
05   <body>
06     ...
07   </body>
08 </html>
```

In the context of the Zend Framework, the `javascript` directory referenced in line 03 would reside within the `/public/javascript/` directory, so if this directory doesn't exist, go ahead and create it now. Next create the `myjavascript.js` file, and place it in the directory. Within that file, add just a single line:

```
alert("This is my first JavaScript script!");
```

Load the page into the browser, and you'll see the message appear over the browser window, as shown in Figure 11-1.



Figure 11-1. Creating a JavaScript alert window

While the approach of referencing an external script is recommended, for testing purposes you might prefer to just directly embed the JavaScript into the HTML like so:

```
01 <html>
02   <head>
```

```
03 <script type="text/javascript">
04   alert("I love video games!");
05 </script>
06 </head>
07 <body>
08 ...
09 </body>
10 </html>
```

Basic Syntax

JavaScript is a vast language sporting countless features, leading me to believe any attempt to cover even the fundamentals within a single chapter, let alone a single section, would be rather self-delusional (see the note for a list of my two favorite JavaScript books). For instance, within the JavaScript language you'll find support for variables, arrays, looping mechanisms, mathematical calculations, string manipulation, and much more. Accordingly, I've given this section quite a bit of thought, and in fact have since cut out a great deal of original draft material in an attempt to provide you with only what's necessary to make the most of this chapter's ultimate goal, which is to show you how to take advantage of a JavaScript framework within the context of the Zend Framework. From there, consider continuing your learning through the numerous online JavaScript tutorials, or by picking up one of the books mentioned in the below note.

NOTE. I've long kept two JavaScript books close to the desk. The first is "Beginning JavaScript", currently in its third edition and co-authored by Paul Wilton and Jeremy McPeak. The second is a more recent title called "Beginning JavaScript with DOM Scripting and Ajax", authored by Christian Heilmann.

Creating and Using Variables

Like PHP, you'll want to regularly create and reference variables within JavaScript. You can formally define variables by declaring them using the `var` statement, like so:

```
var message;
```

JavaScript is a case-sensitive language, meaning that `message` and `Message` are treated as two separate variables. You can also assign the newly declared variable a default value at creation time, like so:

```
var message = "I love video games!";
```

Alternatively, JavaScript will automatically declare variables simply by the act of assigning a value to one, like so:

```
message = "I love video games!";
```

I suggest using the former approach, declaring your variables at the top of the script when possible, perhaps accompanied by a JavaScript comment, which looks like this:

```
// Declare the default message
var message = "I love video games!";
```

Creating Functions

For the typical organizational reasons, you'll want to segment JavaScript code into reusable functions. Like PHP these functions can accept parameters and return results. For instance, let's create a reusable function which displays the above-declared message:

```
01 <html>
02   <head>
03     <script type="text/javascript">
04
05       // Displays a message via an alert box
06       function message()
07     {
08       // Declare the default message
09       var message = "I love video games!";
10
11       // Present the alert box
12       alert(message);
13
14     }
15   </script>
16 </head>
17 <body>
18 ...
19 </body>
20 </html>
```

As you can see, the function's declaration and enclosure look very similar to standard PHP syntax. Of course, like PHP the `message()` function won't execute until called, so insert the following line after line 14:

```
message();
```

Reloading the page will produce the same result shown in Figure 11-1.

You can pass input parameters into a JavaScript function just as you do with PHP; when defining the function just specify the name of the variable as it will be used within the function body. For instance, let's modify the `message()` method to pass along a revised statement:

```
01 // Displays a message via an alert box
02 function message(user, pastime)
03 {
04   // Present the alert box
05   alert(user + " is the " + pastime + " player of the year!");
06 }
```

You can then pass along a user's name and their favorite pastime to create a custom message:

```
message("Jason", "Euchre");
```

Reloading the browser window produces an alert box identical to that shown in Figure 11-2.



Figure 11-2. Using a custom function

TIP. Like PHP, JavaScript comes with quite a few built-in functions. You can peruse a directory of these functions here: <http://www.javascriptkit.com/jsref/>.

Working with Events

Much of your time working with JavaScript will be spent figuring out how to make it do something in reaction to a user action, for instance validating a form when a user presses a submit button. In fact, in Chapter 9 you learned how to make JavaScript do something simply by loading the page by embedding the `onload()` event handler into the page. For instance, you can tell our custom `message()` function to execute when the page is loaded by modifying the `<body>` element:

```
01 <html>
02   <head>
03     <script type="text/javascript">
04
05       // Displays a message via an alert box
06       function message()
07     {
08       // Declare the default message
09       var message = "I love video games!";
10
11       // Present the alert box
12       alert(message);
13     }
14   </script>
15 </head>
16 <body onload="message()">
17 ...
18 </body>
19 </html>
```

Reload this example, and you'll see the alert window immediately appear, just as it did when you explicitly called the `message()` function from within the embedded JavaScript section.

So how do you cause JavaScript to execute based on some other user action, such as clicking a submit

button? In addition to `onload()`, JavaScript supports numerous other event handlers such as `onclick()`, which will cause a JavaScript function to execute when an element attached to the event handler is clicked. Add the following code within the `<body>` tag (and remove the `onload()` function from the `<body>` element) for an example:

```
<input type="submit" name="submit" value="Click Me!" onclick="message();">
```

The button and window which pops up once the button is clicked is shown in Figure 11-3.



Figure 11-3. Executing an action based on some user event

The same behavior occurs when using a simple hyperlink, an image, or almost any other element. For instance, try adding the following lines to the page and clicking on the corresponding elements:

```
<a href="#" onclick="message();">Click me right now!</a><br />
<h1 onclick="message();">I'm not a link but click me anyway!</h1>
```

See Table 11-1 for a list of other useful JavaScript handlers. Try swapping out the `onclick` handler used in the previous examples with handlers found in this table to watch their behavior in action.

Table 11-1. Useful JavaScript Event Handlers

| Event Handler | Description |
|--------------------------|--|
| <code>onblur</code> | Executes when focus is removed from a select, text, or textarea form field. |
| <code>onchange</code> | Executes when the text in an input form field is changed. |
| <code>onclick</code> | Executes when the element is clicked upon. |
| <code>onfocus</code> | Executes when the element is placed into focus (typically an input form field) |
| <code>onload</code> | Executes when the element is loaded |
| <code>onmouseover</code> | Executes when the mouse pointer is moved over an element. |
| <code>onmouseout</code> | Executes when the mouse pointer is moved away from an element. |
| <code>onselect</code> | Executes when text within a text or textarea form field is selected. |
| <code>onsubmit</code> | Executes when a form is submitted. |
| <code>onunload</code> | Executes when the user navigates away or closes the page. |

Incidentally, in the unlikely case you're planning on using JavaScript in a specific location, with no plans of reusing the JavaScript elsewhere, there's no need to go to the trouble of creating a custom function. Instead, you can just embed the JavaScript directly into the HTML, like so:

```
<h1 onclick="alert('I love video games!');">Click me!</h1>
```

Forms Validation

Let's consider one more example involving an HTML form. Suppose you wanted to ensure the user doesn't leave any fields empty when posting a video game review to your website. According to what's available in Table 11-1, it sounds like the `onsubmit` event handler will do the trick nicely. But first we have to create the JavaScript function to ensure the form field isn't blank upon submission:

```
01 function isNotEmpty(formfield)
02 {
03   if (formfield == "")
04   {
05     return false;
06   } else {
07     return true;
08   }
09 }
```

This function is in many ways practically identical to the simple validation functions we first created back in Chapter 3. In short, if the `formfield` parameter is blank, `false` is returned, otherwise `true` is returned.

From here, you can reuse this function as many times as you please by referencing it within another function, which we'll call `validate()`:

```
01 function validate()
02 {
03   // Retrieve the form's title field
04   title = document.getElementById("title").value;
05
06   // Retrieve the form's review field
07   review = document.getElementById("review").value;
08
09   // Verify neither field is empty
10   if (isNotEmpty(title) && isNotEmpty(review))
11   {
12     return true;
13   } else {
14     alert("All form fields are required.");
15     return false;
16   }
}
```

As this is the most complex example we've encountered thus far, let's break down the code:

- Lines 04 and 06 use something called the Document Object Model (DOM) to retrieve the values of the elements identified by the `title` and `review` identifiers. The DOM is a very powerful tool, and one I'll introduce in detail in the next section.

- Line 10 uses the custom `isNotEmpty()` function to examine the contents of the `title` and `review` variables. If both variables are indeed not empty, true is returned which will cause the form's designated action to be requested. Otherwise an error message is displayed and FALSE is returned, causing the form submission process to halt.

Finally, construct the HTML form, attaching the `onsubmit` event handler to the `<form>` element:

```
<form action="/reviews/post" method="POST" onsubmit="return validate();">
  <p>
    <label name="title">Please title your review:</label><br />
    <input type="text" id="title" name="title" value="" size="50" />
  </p>
  <p>
    <label name="review">Enter your review below</label><br />
    <textarea name="review" id="review" rows="10" cols="35"></textarea>
  </p>
  <p>
    <input type="submit" name="submit" value="Post review" />
  </p>
</form>
```

Should the user neglect to enter one or both of the form fields, output similar to that shown in Figure 11-4 will be presented.

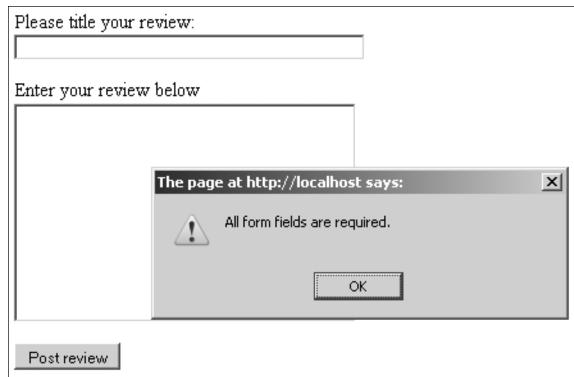


Figure 11-4. Validating form fields with JavaScript

The use of the Document Object Model (DOM) to easily retrieve parts of an HTML document, as well as user input, is a crucial part of today's JavaScript-driven features. In the next section I'll formally introduce this feature.

Introducing the Document Object Model

Relying upon an event handler to display an alert window can be useful, however events can do so much more. Most notably, we can use them in conjunction with a programming interface known as the Document Object Model (DOM) to manipulate the HTML document in interesting ways. The DOM is a standard specification built into all modern browsers which makes it trivial for you to

reference a very specific part of a web page, such as the `<title>` tag, an `<input>` tag with an id of `email`, or all `` tags. You can also refer to properties such as `innerHTML` to retrieve and replace the contents of a particular tag. Further, it's possible to perform all manner of analytical and manipulative tasks, such as determining the number of `` entries residing within a `` enclosure.

JavaScript provides an easy interface for interacting with the DOM, done by using a series of built-in methods and properties. For instance, suppose you wanted to retrieve the contents of a `<p>` tag (known as an *element* in DOM parlance) having an id of `message`. The element and surrounding HTML might look something like this:

```
01 ...
02 <p id="message">Your profile has been loaded.</p>
03 <h1 id="gamertag">wjjgilmore</h1>
04 Location: <b id="city">Columbus</b>, <b id="state">Ohio</b>
05 ...
```

To retrieve the text found within the `<p>` element (line 02), you would use the following JavaScript command:

```
<script type="text/javascript">
  message = document.getElementById("message").innerHTML;
</script>
```

You can prove the text was indeed retrieved by passing the `message` variable into an alert box in a line that follows:

```
alert("Message retrieved: " + message);
```

Adding the `alert()` function produces the alert box containing the message "Your profile has been loaded.".

Retrieving the text is interesting, but changing the text would be even more so. Using the DOM and JavaScript, doing so is amazingly easy. Just retrieve the element ID and assign new text to the `innerHTML` property!

```
document.getElementById("message").innerHTML = "Your profile has been updated!";
```

Simply adding this to the embedded code doesn't make sense, because doing so will change the text from the original to the updated version before you really have a chance to see the behavior in action. Therefore let's tie this to an event by way of creating a new function:

```
function changetext()
{
  document.getElementById("message").innerHTML = "Your profile has been updated!";
}
```

Next, within the HTML body just tie the function to an `onclick` event handler as done earlier:

```
<a href="#" onclick="changetext();">Click here to change the text</a>
```

Everything you've learned so far lays the foundation for integrating Ajax-oriented features into your website. In the next step I'll show you how to query the server and update a webpage with new data, all without having to actually reload the page!

Step #2. Introducing Ajax

You might be wondering why I chose to name this section title "Introducing Ajax". After all, haven't we been doing Ajax programming in many of the prior examples? Actually, what we've been doing is fancy JavaScript programming involving HTML, CSS and the DOM. As defined by the coiner of the term Ajax Jesse James Garrett (<http://www.adaptivepath.com/ideas/essays/archives/000385.php>), several other requisite technologies are needed to complete the picture: *XML*, *XSLT*, and the *XMLHttpRequest object*. With the additional technologies thrown into the mix, we're able to harness the true power of this programming technique, which involves being able to communicate with the Web server in order to retrieve and even update data found or submitted through the existing web page, *without having to reload the entire page!*

By now you've seen the power of Ajax in action countless times using popular websites such as Facebook, Gmail, and Yahoo!, so I don't think I need to belabor the advantages of this feature. At the same time, it's doubtful an in-depth discussion regarding how all of these technologies work together is even practical, particularly because we can take advantage of them without having to understand what's under the covers, much in the same way we can use for example the Zend_Db component without being privy to the gory implementational details. The facility in which we can implement Ajax-driven features can be attributed to the many JavaScript frameworks which were borne out of developers' need to do so quickly and efficiently. In this step I'll introduce you to two of my favorite such solutions, namely Prototype and Script.aculo.us, along the way demonstrating several useful examples.

NOTE. The Zend Framework team has already adopted an official Ajax toolkit known as Dojo (<http://www.dojotoolkit.org/>). However this move was primarily made to provide those users not interested in evaluating the various existing solutions with one which will integrate well with the Zend Framework (<http://devzone.zend.com/article/3545-Dojo-and-Zend-Framework-Partnership-Announcement>). This does not however prevent you from adopting any other of the existing solutions! In fact, one of the reasons I decided to discuss Prototype and Script.aculo.us rather than Dojo was precisely to show you just how easy it is to adopt another solution.

Introducing Prototype

Prototype (<http://www.prototypejs.org/>) is a particularly powerful JavaScript framework offering a wide array of tools capable of interacting with the DOM, performing Ajax tasks, and much, much more. It also greatly streamlines the amount of code required to perform various tasks. For instance earlier in this chapter we used the following command to retrieve the contents of an element having the ID of `message`:

```
message = document.getElementById("message").innerHTML;
```

Using Prototype, you can retrieve the same data using the following command:

```
message = $('message').getValue();
```

This interesting behavior extends in a variety of fascinating ways. For example, suppose you wanted to hide a particular `div` element when the user clicks upon it, a feature which might be useful for displaying a news flash on the home page. Once read, the user can click on the news flash to make it disappear. To create this feature, all you need to do is tie the `div`'s event handler to the following function:

```
function hideNewsFlash()
{
    $('message').hide();
}
```

Now that's easy!

Prototype also supports another great shortcut for interacting with form fields. Just as `$()` is useful for retrieving elements by their ID, you can use `$F()` to return the values of form input elements (you can also use `$()` to retrieve form input elements, but using `$F()` helps to further clarify the coder's intent). For example, suppose you wanted to use Prototype to validate an e-mail address prior to submitting the form. If this e-mail address field's ID is `email`, you can retrieve it with the following function:

```
$F('email');
```

Later in the section we'll build full-blown validation and Ajax-driven examples using Prototype. But you'll need to integrate Prototype into your Zend Framework-powered website, which we'll do first.

VIDEO: Introducing Prototype

Want to implement complex Ajax features but hate JavaScript? The Prototype framework abstracts many of these features, leaving you to focus solely upon creating a great website rather than dealing with frustrating JavaScript syntax. This video introduces several of Prototype's key features. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Installing Prototype

To download Prototype, head over to <http://prototypejs.org/download> and right click on the latest version, saving the file to your Zend Framework-powered website's `/public/javascript/` directory. Because all of Prototype's code is found within a single file, the download link points directly to the raw JavaScript file (`.js` extension) rather than the typical zip file found with other downloads.

Once the `prototype.js` file is in place, to begin using Prototype within the Zend Framework, all you need to do is add the following line to your `layout.phtml` file:

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
```

Determining Whether a Registrant's User Name is Already Taken

Popular online services such as Twitter require users to create a unique user name, ensuring each user

has a distinct address available simply by appending the user's user name to www.twitter.com. For example, my Twitter address is <http://www.twitter.com/wjgilmore>. In order to implement this behavior, Twitter has to ensure any new user doesn't inadvertently choose a user name already taken by an existing user. At the same time, Twitter must strive to make the registration process extremely easy, ensuring that a user doesn't become frustrated and navigate away from the registration form, even if he were to happen to be denied several times due to choosing existing user names.

The solution is to perform real-time user name verification, determining whether the registering user's desired user name is taken before he even completes the registration form. Figure 11-5 shows Twitter's registration behavior in action.

The screenshot shows the Twitter registration interface. At the top, it says "Join the Conversation" and "Already on Twitter? Sign in.". Below that, it asks "Already use Twitter on your phone? Finish signup now." The registration form fields are as follows:

- Full name:** John Smith (with a green checkmark icon and the word "ok" next to it)
- Username:** wjgilmore (with a red error message: "username has already been taken")
- Your URL:** <http://twitter.com/wjgilmore>
- Password:** (empty field)
- Email:** (empty field)
- I want the inside scoop—please send me email updates!** (checkbox)
- Type the words above:** 97 gallop (with a reCAPTCHA challenge)
- Create my account** (large button)

Figure 11-5. John is trying to choose my Twitter user name!

Let's implement similar behavior using the `users` table presented in Chapter 8. The form and examples of the validation behavior in action is shown in Figure 11-6.

| | |
|--|----------------|
| Choose a User Name: <input type="text" value="johnnyutah"/> | Available! |
| <input type="button" value="Register!"/> | |
| Choose a User Name: <input type="text" value="wjgilmore"/> | Already taken! |
| <input type="button" value="Register!"/> | |

Figure 11-6. Determining whether a user name is available

First, let's create the registration form. To keep things simple, we'll just create a form consisting of the user name field, and a submit button. This form (rendered in Figure 11-6) is shown in Listing 11-1.

Listing 11-1. The registration form

```

01 <form>
02   <p>
03     <label>Choose a User Name:</label><br />
04     <input type="text" id="username" value="" size="35"
05       onblur="checkUsername($F(this));" />
06     <span id="username-error"></span><br />
07     <input type="submit" id="submit" name="submit" value="Register!" />
08   </p>
09 </form>

```

Because much of this is new, let's break down the code:

- Lines 04-05 define the input field used to accept the user name. Because we want to let the user know as soon as possible whether the user name is available, we'll use the `onblur()` event handler, which will cause the `checkUsername()` function to execute.
- Line 06 defines an element which we'll use to display an appropriate message regarding whether the desired user name is available. As you'll see later, we'll use a JavaScript function in conjunction with a DOM call to populate this element.

Listing 11-2. The checkUsername() function

```

01 function checkUsername(username)
02 {
03   var myAjax = new Ajax.Request('checkusername',
04     {method: 'get', parameters: {username: username},
05      onComplete: postResponse});
06 }

```

Let's break down some more code:

- Prototype's `Ajax.Request` method handles all of the otherwise gory details involved in using JavaScript to communicate with the server and receive the response. All that's required is passing along the proper parameters.
- The first parameter, `checkusername`, identifies the name of the controller action we want to contact. Because in this case the action resides in the same controller serving the registration form, so there's no need for a path. Otherwise, be sure to include the proper path pointing to the controller and corresponding action.
- The second parameter determines whether you want to send the parameters to the `checkusername` action using the POST or GET method. Refer back to Chapter 3 for a refresher on the purposes of these methods.
- The third parameter identifies the parameters we'll be passing to the `checkusername` action. In this case we're passing along just one parameter named `username`. Although seemingly confusing at first, for matters of consistency I used an identical name for the parameter passed into the `checkUsername()` function. To eliminate any confusion, think of the parameter pair-

ing like this: `server : javascript`, with `server` representing the name of the parameter is the action will see it, and `javascript` representing the name of the parameter as it appears in the JavaScript function scope. If you want to pass multiple parameters, just delimit each pairing with a comma.

- Finally, the fourth parameter determines what should happen once the server action has completed. In this case, we're going to execute another JavaScript function named `postResponse()`.

As stated in the final bullet-point of the above code breakdown, the `postResponse()` function will execute once the server action is complete. This function is presented in Listing 11-3, followed by the usual code breakdown.

Listing 11-3. The `postResponse()` function

```
01 function postResponse(transport)
02 {
03     $('username-error').innerHTML = transport.responseText;
04 }
```

The `postResponse()` function has just one purpose: to send a response back to the user, typically by updating an element. The `transport` object is automatically passed in from the `Ajax.Request` call made in the `checkUsername()` function, and this object's `responseText` attribute contains whatever data was passed back from the controller action.

Finally, let's create the `checkusername()` controller action used to consult the database and determine whether the desired user name has already been taken. This action is shown in Listing 11-4.

Listing 11-4. The `checkusername()` action

```
01 public function checkusernameAction()
02 {
03     // No layouts needed for these methods
04     $this->_helper->layout()->disableLayout();
05     Zend_Controller_Front::getInstance()->setParam('noViewRenderer', true);
06
07     $username = $this->_request->getParam('username');
08
09     $user = new User();
10     $exists = $user->findByUsername($username);
11
12     if (count($exists) == 1)
13     {
14         echo "Already taken!";
15     } else {
16         echo "Available!";
17     }
18 }
19 }
```

By now you should be very well-acquainted with the general operation of a typical controller action. However I'll take the liberty of pointing out a few key items:

- Because this action is intended to be solely accessed through an Ajax call, it shouldn't render the layout or an action-specific view. Thus lines 04-05 disable both behaviors.
- Line 07 retrieves the `username` parameter, passed in using the GET method.
- Line 10 determines whether the desired user name has already been taken. The `findByUsername()` method is provided in Listing 11-5.
- If it has been determined the desired user name has already been taken, an appropriate message will be returned (line 14). Otherwise, the user will be provided with a confirmatory message indicating this user name is still available (line 16). Take note the output is being echoed rather than returned.

Finally, let's create the `User` model method used to determine whether the user name already exists. The `findByUsername()` method is shown in Listing 11-5. I'll forego any explanation as it should be pretty obvious what's happening in this method.

Listing 11-5. The User model's `findByUsername()` method

```
01 function findByUsername($username)
02 {
03     $query = $this->select();
04     $query->where('username = ?', $handle);
05     $result = $this->fetchRow($query);
06     return $result;
07 }
```

Congratulations! You've just implemented a pretty slick feature which greatly enhances the usability of your website. But if you've been following along and executing the code, it seems there's room for improvement by sprucing up the notification messages with some additional eye candy. For instance, perhaps we could use a special effect when displaying the message? Doing so is a breeze using a Prototype add-on called `Script.aculo.us`, which I'll introduce next.

Introducing `Script.aculo.us`

Prototype does a fantastic job of manipulating the DOM and helping developers carry out otherwise complex Ajax operations. In fact, built-in helpers for creating special graphical effects are perhaps the only area where Prototype is lacking. However, an enterprising developer named Thomas Fuchs (<http://script.aculo.us/thomas/>) created a library which extends the Prototype framework which gives developers the ability to create some pretty amazing effects. Using `Script.aculo.us` you'll be able to dazzle your users with special effects capable of making elements pulse, shake, fade, grow, shrink, and more. `Script.aculo.us` also offers built-in autocompletion controls, in-place editors, and drag-and-drop capabilities. In short, if you're looking to incorporate some serious eye candy into your website, look no further.

Script.aculo.us offers another significant advantage in that because it extends Prototype, even the novice Prototype user will find Script.aculo.us' syntax instantly familiar. For instance, to cause an element to shake, just attach the `shake()` method when retrieving the element by its ID:

```
$(‘message’).shake();
```

In many cases you can pass along parameters which will tweak the effect's default behavior. For instance, the default shake duration is just 0.5 seconds. What if you wanted to make a real statement, shaking the element for a full 3 seconds? Just use the `duration` parameter like so:

```
<div onclick="$(this).shake({duration: 3.0})">
Click here to shake it!
</div>
```

Quite a few effects are available, so be sure to check out the script.aculo.us documentation (<http://script.aculo.us/>) and the Chapter 11 code download for more examples!

VIDEO: Introducing Script.aculo.us

If Prototype is the cupcake, Script.aculo.us is the icing, adding an amazing array of visual effects to your JavaScript toolbox. This video shows you how to implement several of these effects. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Installing Script.aculo.us

Like Prototype, Script.aculo.us is free software released under the MIT license. To download Script.aculo.us, head over to <http://script.aculo.us/downloads> and download the latest version in the compression format of your choice. Decompress the download and you'll see several directories and a few files. Presuming you've already placed the `prototype.js` file within your `/public/javascript/` directory as described earlier in this step, all you need to do is enter the download's `/src/` directory and copy the eight files (`builder.js`, `controls.js`, `dragdrop.js`, `effects.js`, `scriptaculous.js`, `slider.js`, `sound.js`, and `unittest.js`) into your `/public/javascript/` directory. If you haven't already placed the `prototype.js` file into your `/public/javascript/` directory, enter the download's `/lib/` directory and copy the `prototype.js` file found there into the `/public/javascript/` directory.

To begin using Script.aculo.us within your website, just add the following two lines within the `<head></head>` tags of your `layout.php` file (the first line might already be there if you followed along with previous exercises):

```
<script src="/javascript/prototype.js" type="text/javascript"></script>
<script src="/javascript/scriptaculous.js" type="text/javascript"></script>
```

The `scriptaculous.js` library doesn't actually do anything other than determine which of the other seven files are loaded, which by default is all of them. You can increase performance by omitting those libraries you don't plan on using by identifying only those you want through the `load` parameter. For instance, suppose you only want to load the `effects.js` library:

```
<script src="/javascript/scriptaculous.js?load=effects">
```

```
></script>
```

The purpose of each library is explained in Table 11-2.

Table 11-2. Script.aculo.us' JavaScript Libraries

| Library | Purpose |
|------------------|--|
| builder.js | Facilitates in the dynamic creation of DOM elements |
| controls.js | Provides in-place editing and autocompletion controls |
| dragdrop.js | Offers drag-and-drop capabilities |
| effects.js | Animation effects |
| scriptaculous.js | Loads the various Script.aculo.us libraries |
| slider.js | Offers a slider control for selecting from a range of values |
| sound.js | Playback of audio from within the browser |
| unittest.js | Performs JavaScript unit testing |

Animating the Form Validation Process

Animating the `username-error` span element used in the user name example is easy. Just update the `postResponse()` function defined in Listing 11-3 to perform the special effect! In the following example I've used Script.aculo.us' `pulsate()` method to cause the element to pulsate each time it's updated:

```
function postResponse(transport)
{
    $('username-error').innerHTML = transport.responseText;
    $('username-error').pulsate();
}
```

Try swapping the `pulsate()` method with other effects, and also change the underlying CSS to devise an even more appealing result!

Conclusion

This chapter introduced one of the hottest development paradigms out there today, which rather ironically relies upon one of the Web's earliest technologies. Using libraries such as Prototype and Script.aculo.us, you can implement these features with minimal inconvenience, allowing your users to start taking advantage of them almost instantaneously!

In the next chapter we're going to examine two more very popular technologies, namely Real Simple Syndication (RSS) and Facebook application development. You'll be able to use both to greatly expand your website's sphere of influence, generating more traffic in the process!

CHAPTER 12

Extend Your Website with RSS and Facebook

Back in the early days of the web, companies new and old were obsessed with "monetizing eyeballs"; in other words, encouraging as many people to come to their website as possible. This encouragement often came in the form of lavish parties, product giveaways, and daring Super Bowl ads. Following the burst of the Internet bubble, coupled with the rise of massive social networking hubs such as Facebook and MySpace, companies started considering more pragmatic tactics when it comes to reaching the user. The most successful of these tactics have focused on making data more conveniently accessible to the user through a variety of destinations and formats, rather than forcing the user to repeatedly return to the company website.

Given today's challenge of vying for a share of users' increasingly divided attention, it pays to devise methods for packaging information for publication in a variety of outlets, giving the user the opportunity to choose the outlet most convenient for his consumption. Two such outlets are via RSS feeds and the Facebook Platform. In this chapter I'll introduce you to both formats, showing you how to create and publish RSS feeds, as well as build a Facebook application which integrates a user's GameNomad profile and game collection into their Facebook profile.

Chapter Steps

The goals of this chapter are accomplished in nine action-packed steps:

- **Step #1. Creating and Publishing RSS Feeds:** In this opening step you'll learn how to both create and publish RSS feeds, giving users the ability to quickly scan the most popular games tracked by GameNomad as determined by Amazon.com sales trends.
- **Step #2. Introducing the Facebook Platform:** In the second step you'll be introduced to the Facebook Platform, Facebook Connect, and other key Facebook technologies which make Facebook application development possible, including Facebook Markup Language, Facebook Query Language, and Facebook JavaScript.
- **Step #3. Building Your First Facebook Application:** In this third step we'll register and configure a new Facebook application, a process which doesn't involve any programming but does require you to have a general understanding of at least some of the many configuration settings available when building your application.
- **Step #4. Introducing the Facebook PHP Client Library:** While quite a few language libraries exist for creating Facebook applications, PHP developers are particularly blessed in that the actual Facebook team wrote the official PHP client library. In this fourth step I'll introduce you to this library.
- **Step #5. Creating a Facebook Platform Controller:** All third-party Facebook application logic is hosted and managed by the developer, giving you the flexibility to maintain your application in any manner you please. In this step we'll create a Zend Framework controller and begin integrating the Facebook client library into the website.

- **Step #6: Creating a Profile Tab:** While the primary purpose of GameNomad's Facebook application is to grant registered users an easy way to access their GameNomad profile and collection, it would also be great if Facebook friends had a way to keep tabs on a user's collection. In this step I'll show you how to create a profile tab which can be added to the user's profile page.
- **Step #7. Sending a Facebook User Notification:** Users and their friends can be notified when a particular event occurs, such as when a friend adds a new game to his collection. In this step I'll show you how to send these notifications.
- **Step #8. Adding Facebook Status Updates:** Updating your Facebook status is one of the most interesting, and indeed addictive aspects of this social networking platform. In this step you'll learn how to send updates to a user's account via your application.
- **Step #9. Deploying the Facebook Application:** In this ninth and final step I'll show you how to deploy your Facebook application, and discuss steps you might consider taking to promote the application and track its popularity.

Step #1. Building and Publishing RSS Feeds

Short for *Real Simple Syndication*, RSS provides users with a convenient way to keep tabs on favorite websites without having to actually visit the site. Instead, users view RSS feeds within an application (often Web-based) known as an *RSS aggregator*. These feeds are regularly updated with the latest content published on the corresponding website, typically including the content title, publication date, a short summary, and a link to the content. As the user scans his feeds, he can easily click on any link of interest, thereby eliminating the need to tediously visit each website for the latest news. If you're not familiar with RSS aggregators, Figure 12-1 presents a screenshot of my RSS aggregator, opened to the small business category, which contains seven different RSS feeds. Just think about how much less time it takes to scan these headlines rather than manually visit each site!

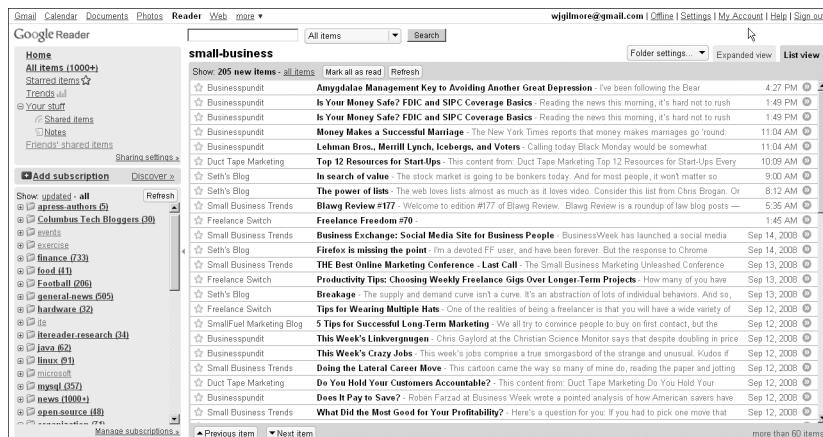


Figure 12-1. Scanning feeds with the Google RSS Reader (<http://reader.google.com/>)

TIP. If you haven't started using an RSS aggregator, check out Google Reader (<http://reader.google.com/>). It's a Web-based aggregator which gives you the ability to read, manage, and search feeds, and even analyze your feed-reading behavior.

Of course, RSS feeds aren't restricted to providing information solely about the daily news or the latest blog entries. These days, organizations are using feeds to keep customers informed about their latest sales, as summaries of the latest technical support tickets, and for a variety of other interesting purposes. In the case of GameNomad, RSS feeds summarize the following content:

- The most recently added games: <http://www.gamenomad.com/rss/recentgames.xml>
- The most popular games (as determined by recent Amazon.com sales trends): <http://www.gamenomad.com/rss/populargames.xml>

In this section I'll show you how to create these RSS feeds using the Zend_Feed component.

VIDEO. Introducing RSS

Web users have come to expect the ability to monitor website updates at a time and place of their choosing, rather than have to tediously return to a given website on a regular basis. RSS has made this possible, giving website developers a solution for packaging updates in a digest format, capable of being parsed and formatted by third-party applications such as <http://reader.google.com/>. This video introduces RSS, and shows you how to use the Zend_Feed component to build and distribute RSS feeds. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Creating Feeds with Zend_Feed

In many ways, the Zend_Feed component is the exemplar of a useful component because it completely abstracts what would otherwise be the rather tedious process of assembling a valid *XML* file. An XML file you ask? What's that? Put simply, XML is a text format used to describe the meaning of content found in a document. For instance, HTML is a dialect of XML in that its tags are used to convey meaning to the various elements comprising an HTML page, **hello** telling the browser to render the word hello in bold text for example. RSS is also an XML dialect, with tags used to identify the feed's title, items, item authors, item publication dates, and other useful pieces of information. A simplified sample feed looks like this:

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>GameNomad: Popular Games</title>
    <description>The latest games added to the GameNomad database.</description>
    <language>en-us</language>
    <pubDate>Tue, 10 Dec 2008 10:42:01 GMT</pubDate>
      <item>
        <title>Tiger Woods PGA Tour 08 (Nintendo Wii)</title>
        <link>http://www.gamenomad.com/games/show/B000P0QIP6</link>
        <description>Sales Rank: #45</description>
      </item>
      <item>
```

```

<title>Cosmic Family (Nintendo Wii)</title>
<link>http://www.gamenomad.com/games/show/B000ME252U</link>
<description>Sales Rank: #321</description>
</item>
<item>
    <title>Ace Combat 6: Fires of Liberation (Xbox 360)</title>
    <link> http://www.gamenomad.com/games/show/ B000P297JS</link>
    <description>Sales Rank: #120</description>
</item>
</channel>
</rss>
```

As you can imagine, programmatically creating a feed such as this can be a difficult and error-prone affair! This is where the Zend_Feed component comes into play, as it will create the feed for you based on the data you provide. While Zend_Feed offers a number of solutions for going about creating a feed, I'll introduce the approach I've found to be the easiest, namely that which passes an array to Zend_Feed for conversion into a feed.

The following action is found in the RSS Controller (`RssController.php`). It retrieves the top 25 most popular games according to the latest Amazon.com sales trends, and converts those games, along with some general RSS information, into an array which is subsequently passed to the Zend_Feed object. This feed is then saved to a file whose name and location is defined by the `$this->config->feed->popular_games` configuration variable (found in `config.ini`).

```

01 /**
02  * Creates an RSS feed containing the most popular games in the GameNomad
03  * database as determined by recent Amazon.com sales trends
04 *
05 */
06 public function popularAction()
07 {
08
09     // Create array to store the RSS feed entries
10     $entries = array();
11
12     // Retrieve the 25 most popular games
13     $game = new Game();
14     $rankings = $game->getLatestSalesRanks(25);
15
16     // Cycle through the rankings, creating an array storing each, and push
17     // the array onto the $entries array
18     foreach ($rankings AS $ranking) {
19         $entry = array(
20             'title'      => "{$ranking->title}
21                                ({$ranking->findParentRow('Platform')->name})",
22             'link'       => "http://www.gamenomad.com/games/{$ranking->asin}",
23             'description' => "Sales Rank: #{$ranking->rank}",
24         );
25         array_push($entries, $entry);

```

```
26     }
27
28 // Create the RSS array
29 $rss = array(
30     'title'    => 'GameNomad: Popular Games',
31     'link'     => 'http://www.gamenomad.com/games/ranks',
32     'charset'  => 'ISO-8859-1',
33     'entries'   => $entries
34 );
35
36 // Import the array
37 $feed = Zend_Feed::importArray($rss, 'rss');
38
39 // Write the feed to a variable
40 $rssFeed = $feed->saveXML();
41
42 // Write the feed to a file residing in /public/rss
43 $fh = fopen($this->config->feed->popular_games, "w");
44 fwrite($fh, $rssFeed);
45 fclose($fh);
46
47 }
```

A breakdown of the example's finer points follows:

- After retrieving a list of the 25 currently most popular games (Line 14), the titles, platform, link, and rank are each stored in an array, which is subsequently pushed onto the end of the `$entries` array. The `$entries` array is in turn inserted into the `$rss` array, which contains these elements along with general RSS information such as the feed title. Keep in mind these array elements are representative of only the required elements; you actually have quite a few other feed options at your disposal. Consult the `Zend_Feed` documentation for more information.
- Line 37 converts this array into an RSS feed. If you're familiar with Atom feeds and prefer that format over RSS, you can substitute `atom` for `rss`.
- Line 40 stores the feed in a variable named `$rssFeed`.
- Lines 43-45 write the feed to a file whose name and location are identified by the `popular_games` configuration variable, which is stored in the `config.ini` file.

Subscribing to this feed via a feed aggregator (<http://www.gamenomad.com/rss/populargames.xml>) produces output similar to that shown in Figure 12-2.

The screenshot shows a Google Reader interface with the following details:

- Title:** GameNomad: Popular Games
- Feed settings...** button
- Expanded view** button
- List view** button
- Show:** 25 new items - all items | Mark all as read | Refresh
- Subscription:** You have subscribed to "GameNomad: Popular Games." | Add to a folder... ▾
- Items:**
 - LittleBigPlanet (Playstation 3) - Sales Rank: #3
 - Mario Kart Wii with Wii Wheel (Nintendo Wii) - Sales Rank: #6
 - Wii Play with Wii Remote (Nintendo Wii) - Sales Rank: #7
 - Fable 2 Limited Edition (XBOX 360) - Sales Rank: #13
 - Wii Music (Nintendo Wii) - Sales Rank: #14
 - Fable 2 (XBOX 360) - Sales Rank: #14
 - Super Mario Galaxy (Nintendo Wii) - Sales Rank: #17
 - Gears of War 2 (XBOX 360) - Sales Rank: #18
 - We Ski (Nintendo Wii) - Sales Rank: #18
 - Pokemon Ranger: Shadows of Almia (Nintendo DS) - Sales Rank: #21
 - Active Life Outdoor Challenge (Nintendo Wii) - Sales Rank: #22
 - Gears of War 2 Limited Edition (XBOX 360) - Sales Rank: #23
 - NBA 2K9 (XBOX 360) - Sales Rank: #24
 - Star Wars: The Force Unleashed (Nintendo Wii) - Sales Rank: #26
 - Mario Kart DS (Nintendo DS) - Sales Rank: #28
 - Star Wars: The Force Unleashed (XBOX 360) - Sales Rank: #29
 - Fallout 3 (XBOX 360) - Sales Rank: #30
 - Mystery Case Files: MillionHeir (Nintendo DS) - Sales Rank: #32
 - Rock Band 2 (XBOX 360) - Sales Rank: #32
 - Brain Age: Train Your Brain in Minutes a Day! (Nintendo DS) - Sales Rank: #33
 - New Super Mario Bros. (Nintendo DS) - Sales Rank: #34
 - NBA 2K9 (Playstation 3) - Sales Rank: #35
 - LEGO Batman (Nintendo Wii) - Sales Rank: #39
 - Carnival Games (Nintendo Wii) - Sales Rank: #40
 - Brain Age 2: More Training in Minutes a Day! (Nintendo DS) - Sales Rank: #43
- Timestamps:** All items were added at 3:32 PM.

Figure 12-2. Viewing GameNomad's Most Popular Games in Google Reader

Scheduling Feed Creation Scripts

There are a number of strategies you could employ regarding the feed update frequency. For instance, you might integrate the feed creation task directly alongside the database update actions, recreating the feed every time a new game is added to the GameNomad website. Or you could recreate the feeds using a predefined time interval such as every six hours. For the purposes of introducing you to a new programmatic concept known as the *cron job* I'll opt for the latter approach.

cron is a scheduling service available on the Linux operating system, providing users with the ability to execute scripts at any predefined date, time or interval. Using a configuration file known as the *crontab*, you define the schedule and identify the script to be executed. The operating system reads the crontab every minute, determining whether a job is scheduled to execute at that time.

To edit the crontab file, login into your hosting account and execute the following command:

```
%>crontab -e
```

Presuming you've not used crontab before, this file will be empty. Next, add the following lines to the file:

```
0 0,6,12,18 * * * wget --http-user=jason --http-password=secret
http://www.gamenomad.com/rss/popular
```

This command tells the Linux `wget` command to access the respective URLs at 12:00am, 6:00am, 12:00pm, and 6:00pm on every day of every month. At first glance this may seem a bit cryptic, however the fields are actually pretty straightforward:

- Field 1: Defines the minute (0-59) of the hour, with the first minute represented by 0
- Field 2: Defines the hour (0-23) of the day, with 1am represented by 0
- Field 3: Defines the day (1-31) of the month
- Field 4: Defines the month (1-12) of the year
- Field 5: Defines the day (0-6) of the week, with Sunday represented by 0

If you wanted the command to execute within every possible value of a given field, use an asterisk as that field's placeholder.

VIDEO. Accessing Password-protected Controller Actions Using cron and wget

Using cron and `wget` to access controller actions is a great way to maintain all of your application's code within the framework rather than have to create and maintain separate scripts. Of course, you'll want to adequately protect these actions so they can't be accessed by an outside user. This video introduces you to these utilities, and shows you how to both password-protect a specific controller and subsequently access it in an automated fashion. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Step #2. Introducing the Facebook Platform

The meteoric rise of Facebook from a college student's project to the world's 4th most heavily-trafficked websites is truly an impressive story. With more than 120 million active users, availability in 35 languages, and an enormous community infrastructure (more than 10 billion hosted photos and more than 6 million user groups within it), Facebook presents a tremendous opportunity to not only attract a larger audience to your website, but actually expand your website's capabilities.

NOTE. If you don't have a Facebook account, I encourage you to sign up and add me as your first friend!

In this step I'll show you how to build an application atop the Facebook platform which allows Facebook users to integrate a number of interesting GameNomad features directly into their Facebook account. These features will give users the ability to:

- Add the GameNomad application to their Facebook account, and easily access it using the application toolbar and right-side navigation menu.
- Access their GameNomad profile and game collection via their Facebook account.

- Add a Facebook profile tab which gives Facebook friends an easy way to view the user's games.
- Post status updates for their Facebook friends' benefit whenever a new game is added to the user's collection, and whenever the user borrows a game from another GameNomad user.
- Receive notifications whenever a friend adds a game to his collection, or borrows a game.

An ambitious agenda, to be certain! However, before moving into the implementation stage, there are a few preliminary topics to discuss.

VIDEO: Building a PHP-driven Facebook Application

Building a Facebook application using the Facebook Platform is actually a fairly easy process, although one consisting of numerous steps. This video guides you through the process, showing you how to register and configure a new application, and use the Facebook PHP client and Facebook Platform to build a number of interesting features into your application. Watch the video at <http://www.easyphpwebsites.com/zfw/videos/>.

The Facebook Platform

Hosting more than 120 million active users, Facebook has unquestionably become the mother of all social networks. But even in spite of the tremendous mindshare employed by the company, the Facebook team quickly realized the importance of giving developers the ability to extend the Facebook website in ways not otherwise dreamed. These efforts effectively transformed Facebook from a tightly controlled website to an organic platform in which developers can build a whole new array of interesting social applications.

Navigating to Facebook's Application Directory (<http://www.facebook.com/apps/>), you'll find Facebook applications suiting a wide variety of purposes, ranging from simple utilities such as Birthday Alert, which sends you e-mail alerts regarding your friends' upcoming birthdays to games such as Texas Holdem Poker which makes it possible for you to play the popular casino game with your network. You'll also find a healthy dose of light-hearted applications such as Likeness, which you use to take a quiz in order to determine what famous person you most resemble, or Pillow Fight, which allows you to hold a virtual pillow fight on the Facebook website.

These days you'll find startups fighting for attention alongside corporate behemoths such as Pizza Hut, Hasbro, and American Greetings, a sure signal Facebook's transformational strategy is working.

Introducing Facebook Connect

Facebook's platform initiative proved so popular that the team soon started seeking out other ways to expand its offerings. These efforts have most recently culminated with the latest version of the Facebook Platform, dubbed *Facebook Connect*. Facebook Connect not only continues giving developers the ability to create applications which integrate into the Facebook.com website, but additionally extends Facebook's authentication, identity, network, and privacy features to third-party websites!

Why might this be useful? For starters, you could create a mechanism for linking your website users' accounts to their Facebook accounts, giving you the ability to retrieve their Facebook profile

and network information. With the user's permission you'll also be able to automatically update their Facebook status, send Facebook notifications, and determine whether other members of their Facebook network are also members of your website (and act accordingly, such as prompting the user to connect with the member likewise on your site).

This is accomplished by integrating select Facebook JavaScript libraries and other extensions into your website. These work with the underlying technology of your website to connect the accounts, retrieve the user's Facebook profile and network information, and send information to the user's Facebook account and network. For instance, Figure 12-3 shows what you would encounter when connecting a website account with your Facebook account.

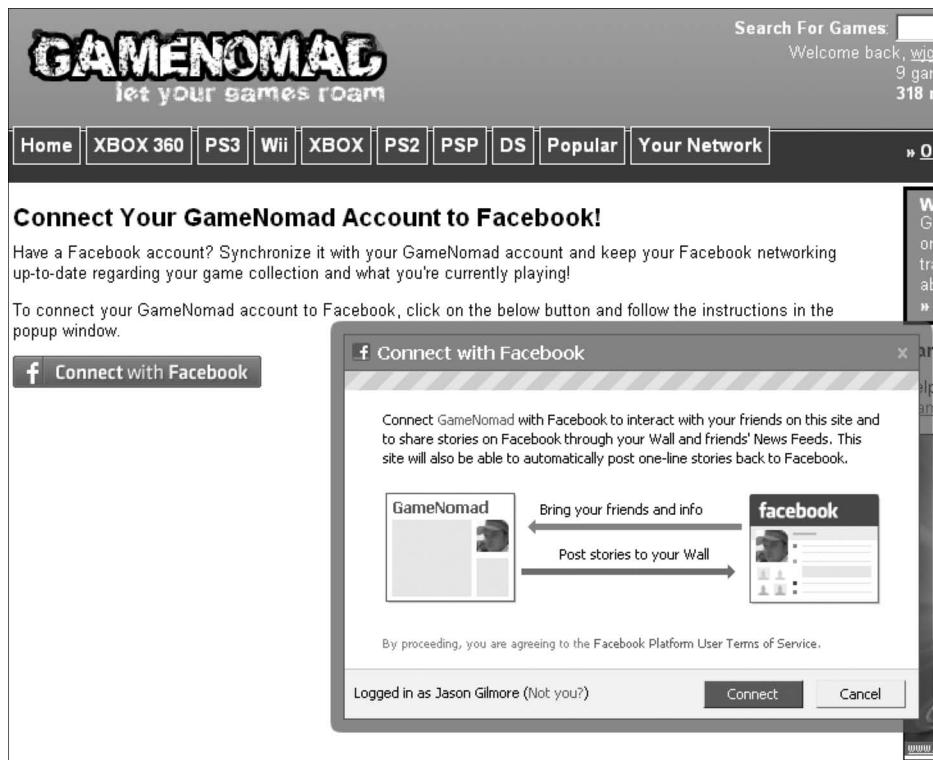


Figure 12-3. Connecting a Facebook User to GameNomad

Once connected, the website can begin initiating a variety of other actions, such as prompting the user to allow the site to post status updates on the user's behalf. You'll learn how to request these additional privileges, not to mention connect accounts, later in this step.

NOTE. Facebook requires you to closely follow several defined policies before your Facebook Connect-driven application can be publicly released. Facebook ensures you've conformed by requiring you to submit your website to the Facebook team for approval. Most of these policies involve making it clear to the user that he is currently logged into your website and therefore connected to Facebook, for instance by displaying the user's Facebook profile icon and name on each page. At the time of this writing these policies were still in a state of flux, so be sure to Facebook documentation for the latest information.

At the time of writing, Facebook Connect is still an emerging technology, albeit one with great promise. For the purposes of this discussion, we'll stick with creating a "traditional" Facebook application, an approach still embraced by the vast majority of Facebook developers. Nonetheless, be sure to stay atop the latest developments regarding this fascinating new feature.

Key Facebook Technologies

The Facebook team has taken the pragmatic approach of building the Facebook website based upon proven, scalable technologies and open standards. Additionally, they've wisely made it possible for third-party developers to build Facebook applications using similarly proven solutions. For instance, although the Facebook website is powered by PHP and MySQL, it's possible to build applications using any of the mainstream programming languages, PHP, C#, Ruby, Perl, and Java among them.

While you could build a working application using only a programming language and HTML, the Facebook developers have granted you access to a few other technologies which make the development process even easier. In this section I'll introduce you to these five technologies, explaining how each facilitates the application development process.

The Facebook REST API

In the previous paragraph, I mentioned it's possible to create a working Facebook application using a chosen programming language and HTML. This is possible thanks to Facebook's REST API, which exposes a series of methods accessible using the HTTP protocol. For instance, one method retrieves all of the friends tied to a specific user, while another determines whether a user has granted your application permission to perform a specific task, such as update his Facebook status. Enterprising developers have created wrappers in their favorite language to make calling Facebook's API easier.

The specifics regarding the API, not to mention how exactly these language-specific implementations are created in the first place, is out of the scope of this book, however its suffice to say that these collective efforts go a long way towards making your life much easier as a developer. Later in this scenario I'll introduce you to the most popular PHP implementation, namely Facebook's official PHP client library.

Facebook Markup Language (FBML)

Like RSS, Facebook Markup Language (FBML) is an XML-dialect; it's comprised of a set of tags which convey some sort of special meaning, much like the `` tag tells a browser to format anything found within the tag using a bold-face font. For instance, the following tag will render within your application the first name of the Facebook user identified by the user ID 12345678:

```
<fb:name uid="12345678" firstnameonly="true" />
```

This is just one of over 120 available tags, capable of not only retrieving and formatting user-related information but also creating interfaces which closely resemble the look-and-feel of the Facebook website. You'll even find tags capable of performing rather esoteric tasks such as showing contained content only in the case it's April Fool's Day (`<fb:is-it-april-fools></fb:is-it-april-fools>`).

FBML for Facebook Connect (XFBML)

As you learned earlier in this chapter, Facebook Connect makes it possible to extend the Facebook platform to your website by integrating the platform's identity, network, news, and privacy features into your own solution. To render Facebook-specific user data and interface elements to your Facebook Connect-associated website, you'll use XFBML, which is a subset of supported FBML tags.

In actuality, these tags are often practically identical to their FBML-counterparts. For instance, displaying the user's first name uses the same `<fb:name>` tag as introduced earlier, the only difference being that XFBML tags require a closing tag:

```
<fb:name uid="12345678" firstnameonly="true" /></fb:name>
```

Facebook Query Language (FQL)

For the SQL-minded among us, Facebook offers an alternative solution for querying user data using SQL syntax. This syntax is almost identical to standard SQL syntax, and allows you to retrieve only select parts of a data structure rather than all of it, thereby increasing performance. For instance, the following query retrieves the first name of the Facebook user identified by the user ID 12345678:

```
SELECT first_name FROM user WHERE uid = 12345678
```

You'll also be pleased to learn FQL supports a number of functions and operators such as `concat()`, `lower()`, `upper()`, `rand()`, `AND`, `OR`, and `NOT`, and even supports subqueries (with some restrictions). FQL however isn't a full-blown query language, and is missing quite a few features you've probably come to depend on within other database-driven applications. For instance, while you can use clauses such as `WHERE`, `ORDER BY`, and `LIMIT`, open ended queries are not allowed; instead, you must restrict the query to one of the table columns identified as indexable. For instance, you can execute the above query because `uid` is identified as being indexable, however this query would return an error message:

```
SELECT first_name FROM user WHERE religion = "Protestant"
```

The reason for such restrictions might be evident. Executing these sorts of queries on a database the size of Facebook's would place an enormous strain on the system, thereby reducing usability for all users.

Facebook JavaScript (FBJS)

These days, most developers use JavaScript (judiciously or otherwise) to enhance website usability. Recognizing this trend, the Facebook developers allow you to use JavaScript within your Facebook applications, albeit a special version known as Facebook JavaScript (FBJS). This modified version eliminates common security threats such as cross-site scripting (XSS) attacks.

For the most part, FBJS is no different from standard JavaScript. The general syntax and behavior is identical, however differences begin to arise when it comes to DOM manipulation, one of the key tenets of AJAX. These differences are largely syntactical however, so getting used to the slightly different naming conventions seems a small price to pay in return for the ability to integrate cutting-edge JavaScript features into your Facebook application.

Step #3. Building Your First Facebook Application

To get started, navigate to <http://www.facebook.com/developers/> and allow the Facebook Developer application to access your Facebook profile. Click the "Set Up New Application" button to begin creating your application.

The first task is to assign a unique name to your application (Figure 12-4). While application names can consist of up to 50 characters, I recommend keeping the name brief so users can easily remember it. If the application is intended to serve as a companion to your web application, consider giving it a name identical to the website. For instance, I called the GameNomad application "GameNomad". At any rate, enter the application name, check the box confirming you agree to Facebook's Terms of Service, and click the Submit button to continue. Congratulations, your Facebook application has been created!

The screenshot shows the 'New Application' setup page. At the top left is a 'Back to Developer Home' link. Below it is a 'New Application' section with a 'Required Fields' header. Under 'Application Name' (with a note '(Limit: 50 characters)'), there is an input field containing 'GameNomad'. Below it is a checkbox labeled 'Check here to indicate that you have read and agree to the terms of the Facebook Platform.' At the bottom left is an 'Optional Fields' section, and at the bottom right are 'Submit' and 'Cancel' buttons.

Figure 12-4. Assigning a unique name to your Facebook application

Next, click the **Edit Settings** link to begin configuring your application. As you can see, there are a large number of adjustable fields. Rather than exhaustively list and define every option, I'll instead touch upon those which you'll likely want to immediately adjust:

- **Developer Contact E-mail:** Facebook will use this address to contact you about any application problems or updates.
- **User Support E-mail:** Each Facebook application page includes a "Contact" link at the bottom of the screen. Clicking on this link produces a contact form in which users can use to contact you with questions or comments. That correspondence will be e-mailed directly to the address you provide here.
- **Callback URL:** This is the location in which your Facebook application logic resides. In the case of GameNomad, that URL is <http://www.gamenomad.com/facebook>. I recommend using the same name as the one you chose for your application. Later in this step I'll explain the important role this particular URL plays in the application.
- **Base Domain:** By default Facebook Connect ties applications to a very specific domain identifier. For instance, www.gamenomad.com is different from gamenomad.com. However, larger websites often use multiple sub domains for scaling or organizational purposes. For instance,

GameNomad might one day configure a second server to manage users hailing from Europe as opposed to visitors hailing from the U.S. separating them using the domains `europe.gamenomad.com` and `us.gamenomad.com`, respectively. Of course, you'd still want to use the same Facebook application for all users, meaning the base domain would have to be set to `gamenomad.com` in order for both domains to work.

- **Canvas Page URL:** This defines the address where your Facebook application will reside. I recommend using the same name as the application. For instance, GameNomad's is `http://apps.facebook.com/gamenomad/`.
- **Profile Tab URL:** It's possible to add a tab at the top of the user's Facebook page, alongside the "Wall", "Info", "Photos", and other tabs. This tab won't appear by default; it needs to be enabled by the user through the Facebook application settings interface. However, if you'd like to give the user the ability to do so, assign a URL here. For instance, the GameNomad Profile Tab URL is `http://apps.facebook.com/gamenomad/games`. I'll talk more about this topic in the section "Creating the Tab View".
- **Profile Tab Name:** If you decide to create a tab, assign the tab title here.
- **Application Type:** Facebook supports two application types: Website- and desktop-based. We'll be covering the former type, so select `Website`.
- **Can your application be added on Facebook?**: The answer to this question is presumably yes, unless you'd like to temporarily disable the ability to do so.
- **TOS URL:** If you'd like to force users to accept a Terms of Service agreement before adding the application, specify the URL here.
- **Developers:** When developing an application, you want to grant access to a select set of developers. Clearly you're one of these developers however if you'd like to grant developer privileges to another Facebook member, you can do so here.
- **Icon:** The icon is used to identify the application within the Applications menu found at the bottom left of your Facebook account. It can be a maximum of 16x16 pixels, and must be uploaded in GIF, PNG, or JPEG format.
- **Logo:** This larger icon (maximum of 75x75 pixels) is used within the application directory listing and select other locations within the Facebook website. Like the Icon, this must also be uploaded in GIF, PNG, or JPEG format.
- **Who can add your application to their Facebook account?**: Most applications are intended for users' use, therefore leave the Users checkbox enabled if that's your intention. Additionally, you can optionally allow page administrators to add the application, done by selecting "All Pages" or "Some Pages". If you select the latter, a list of categories will appear. Choose those page categories which you'd like to open application access to.
- **Application Description:** This short description will describe your application to users considering adding it to their Facebook account.

- **Developer Mode:** When developing your application, it's a wise idea to enable Developer Mode which will prevent anybody but those users identified by the Developers field from adding it to their Facebook account. Once you've worked out all of the kinks, uncheck this box to open the application up to the Facebook community. Remember, if you're planning on taking advantage of Facebook Connect, you'll also need to obtain approval by submitting the application for review.

You'll see quite a few other options within this settings list. Feel free to learn about any that interests you, keeping in mind you can always return to this page later and make further adjustments as necessary. Click the Save button to save your changes.

Once saved, you'll be returned to the application's development home page. Locate the API and Secret keys available on this page. These keys are used to identify your application when connecting to the Facebook platform, so you should add them to your configuration file (config.ini) for reference within the code. An example of what this addition might look like follows (using contrived keys):

```
; Facebook
fb.apikey = 72c7a8dc05cbf07f4c1c7796bci83107
fb.secretkey = d1dc31c44cc6a8bbb92bf5862714be8d
```

Step #4. Introducing the Facebook PHP Client Library

Because Facebook heavily relies upon the PHP language to power the website, they've put special emphasis on the language's suitability for building Facebook applications. To download this client, navigate to your application's settings page and scroll to the bottom. You'll see a link titled "Download the Client Library". Click this link to download Facebook's PHP-specific client library. Unzip this file and copy all of the files found in the php directory into a directory named `facebook` located in the `/library/GameNomad` directory where you placed the `index.php` file. Finally, add the `facebook` directory to PHP's include path by either modifying the `php.ini` file, or adding it to your application's `index.php` file:

```
set_include_path('..../library/GameNomad/facebook' . PATH_SEPARATOR .
    get_include_path());
```

Once the library is accessible to PHP, you're ready to begin using it. But what exactly does this library do? In a sentence, the Facebook PHP client uses the PHP language to implement many of the methods defined by the previously introduced Facebook REST API. This negates the need for you to have to deal with the complexities otherwise involved in communicating with Facebook's REST service and parsing the resulting XML, freeing you to concentrate almost exclusively upon creating the Facebook application. Among other tasks, you can use the client to connect to the Facebook platform, retrieve user information such as his name and user ID, as well as a list of friends in his network, and even retrieve information about a user's photo albums.

While the client is extremely useful, it is curiously not feature complete. For instance, it currently lacks the ability to update a user's Facebook status, which is clearly a particularly useful feature. However, one of the great advantages of having the source code at your disposal is the ability to modify it and add desired features! In fact, in a later section ("Adding Status Updates"), we'll modify

the client to add this important feature.

NOTE. The Facebook team is so PHP-oriented that they even generate a PHP-specific code example tailored specifically to your application. To retrieve this code, scroll down your application's configuration page and click on the "example code" link located under the Sample Code section. A window will pop open, containing installation instructions and the code. Copy and paste that code into a file named `index.php`, saving it to the same directory you identified as the location where your Facebook application logic will reside. Executing this code will result in the display of a list of up to 25 friends found in your Facebook network. Note you'll need to have the PHP client installed and available to the script in order to execute this script.

In the coming sections you'll become well acquainted with the Facebook client syntax as we build out the Facebook application and add interesting features.

Step #5. Creating a Facebook Platform Controller

One of the truly compelling aspects of developing applications for the Facebook platform is that it exposes your project to potentially millions of new users by integrating the application into the Facebook infrastructure while not requiring you to sacrifice undue flexibility while developing and maintaining your application. This is because your application is hosted at a location of your choice, and integrated into the Facebook site using what's called the *canvas page*. Your application is rendered in a frame within this canvas page, its contents retrieved from the location as defined by the *Callback URL* (defined within the application settings interface).

More precisely, this Callback URL defines the application's home page, which in the case of GameNomad is defined as `http://www.gamenomad.com/facebook/`. However, any other page falling within this URL path can also be served within the Facebook frame. For instance, `http://www.gamenomad.com/facebook/privacy`, `http://www.gamenomad.com/facebook/friends`, and `http://www.gamenomad.com/facebook/popular` are all accessible through the Facebook frame, although `http://www.gamenomad.com/contact/` and `http://www.gamenomad.com/about/contact` are not. Therefore one simple way to create and manage the application is to define all pages comprising the Facebook application within the context of a controller named Facebook and proceed from there.

Creating the Facebook Controller

Create a new controller, assigning it an appropriate name such as `FacebookController.php`. We'll use the controller's `init()` method to integrate the Facebook client and perform several other tasks:

```
01 function init()
02 {
03
04     // Grab the Facebook client
05     require_once 'facebook.php';
06
07     // No layouts needed for these methods
08     $this->_helper->layout()->disableLayout();
09 }
```

```

10 // Retrieve the configuration information
11 $this->config = Zend_Registry::get('config');
12
13 $user = new User();
14 $liu = Zend_Auth::getInstance()->getIdentity();
15 if (isset($liu)) {
16     $this->view->user = $user->getUserByEmail($liu);
17 } else {
18     $this->view->user = "";
19 }
20
21 // Retrieve the Facebook API and secret keys, and connect to the platform
22 $appapikey = $config->fb->apikey;
23 $appsecret = $config->fb->secretkey;
24 $this->facebook = new Facebook($appapikey, $appsecret);
25
26 // Require user to be logged-in before using the application.
27 $this->facebook->require_login();
28
29 }

```

A breakdown of the relevant code follows:

- Line 05 retrieves the Facebook client. Remember you'll need to add the `/library/GameNomad/facebook` directory to PHP's include path in order to be able to reference the `facebook.php` script directly.
- Line 08 disables the application layout for all methods in this controller. This is because the method views will be integrated into the Facebook website, meaning we'll need to strip away everything but the data and simple formatting shown in each view.
- Lines 13-19 verify the user is authenticated per usual. Authentication is required in order to retrieve the user's profile and game collection.
- Lines 22-24 initiate the Facebook platform connection process by passing the API key and secret key to the Facebook class constructor. As you might imagine, this is a necessary first step in order to retrieve a user's Facebook profile, and carry out status updates and notifications.
- Line 27 requires the user to be logged-in to his Facebook account in order to use the application.

Next we'll create the application's home page.

Creating the Application's Home Page

GameNomad's Facebook application home page is actually used solely to determine whether the user is already logged into GameNomad. If so, the user is redirected to the profile page (introduced in the

next section), otherwise, a login prompt is rendered (Figure 12-5) and subsequently processed.

The screenshot shows a login form for the GameNomad application. At the top, it says "To add the GameNomad application, you need to login to your GameNomad account." Below this are two input fields: "Your E-mail Address:" and "Your Password:". Underneath the password field is a "Login!" button. At the bottom of the form is a link: "Not a GameNomad member? Registering only takes a moment, and it's free!"

Figure 12-5. Logging into the GameNomad Facebook Application

The first time the Facebook user logs into GameNomad we want to record his Facebook user ID within the GameNomad users table. This user ID is the Facebook user's primary key, used to uniquely identify the user no matter what other parts of his profile subsequently changes. In order to perform tasks such as send notifications and status updates whenever a user updates his profile or game collection from the GameNomad website, we need to retrieve this user ID and associate it with the user's GameNomad account. This ID will be stored in the users table, so let's begin by altering this table.

Modifying the Users Table

The Facebook user ID is a large integer value (for instance mine is 501632489), and so you should use the BIGINT data type to represent it within the user's table. Modify the table using phpMyAdmin or using the ALTER TABLE statement from the MySQL client:

```
ALTER TABLE users ADD COLUMN facebook_id BIGINT UNSIGNED DEFAULT 0;
```

Creating the Login Form and Logic

The login form (Figure 12-5) is just a copy of that discussed in Chapter 8, modified to include a smaller GameNomad logo and Facebook-adapted surrounding text. Other than these cosmetic changes, the form's action points to /gamenomad/login, which will handle the login process.

The login logic (found in the Facebook controller's `indexAction()` method) is also almost identical to that found in the Gamers controller, save for that we want to store the user's Facebook user ID if this is the user's first time logging into the application:

```
// If the user's Facebook ID hasn't been stored,  
// grab it and update the users table  
if ($updateLogin->facebook_id == 0) {  
    $fbUser = $this->facebook->get_loggedin_user();  
    $updateLogin->facebook_id = $fbUser;  
}
```

You'll also need to modify the redirection mechanism to point to /facebook/profile on successful login. We'll create the profile method and view next.

Creating the Profile Page

The profile page summarizes the user's key characteristics and game collection. Figure 12-6 depicts the profile.

The screenshot shows a Facebook application profile for a user named Jason Gilmore. At the top, there's a navigation bar with links: Manage Your Games, Manage Your Friends, Manage Your GameNomad on Facebook Privacy Settings, Check Out the Most Popular Games, and Logout. Below the navigation, Jason's profile information is displayed: "Jason Gilmore is gaming out of Columbus, Ohio". It shows his member since date (October 31, 2008) and last login time (December 01, 2008 @ 04:55:57). A section titled "Greatest Game Ever" lists "Wii bowling because of its ability to bring everybody in the room into the game." There are links to edit the declaration and manage the game collection. The "Game Collection" section indicates Jason has 13 games in his collection. A "Currently Playing" section lists three games: Call of Duty 4: Modern Warfare on the XBOX 360, Mario Kart Wii with Wii Wheel on the Nintendo Wii, and Fallout 3 on the XBOX 360. A "Available to Borrow" section lists ten more games. The entire profile is contained within a light blue box.

Figure 12-6. GameNomad's Facebook application profile view

We're able to easily retrieve the user's profile because the user was required to login to GameNomad, meaning the automatically created user object (in the `init()` method) is easily accessible, and can therefore be used to retrieve the user's profile, games, and friends in precisely the same manner as that used to retrieve his profile from within the GameNomad website.

Step #6. Creating a Facebook Profile Tab

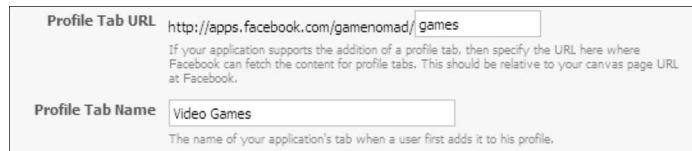
Creating an application of the sort we've been working on so far is useful for the typical GameNomad user who happens to login to Facebook on a regular basis, because it eliminates the hassle of navigating from one website to the next in order to keep track of not only your game collection, but those of your friends. But what if you wanted an easy way to let a user's Facebook friends who haven't yet signed up for a GameNomad account monitor the user's game collection? Using the Facebook Platform, you can easily add a tab to the top of the user's Facebook profile (Figure 12-7).

The screenshot shows a Facebook profile for Jason Gilmore. At the top, there's a navigation bar with tabs: Wall, Info, Photos, Video Games, and a plus sign. Below the tabs, there are social sharing options: Update Status, Share Link, Add Photos, Add Video, Write Note, and a dropdown menu. A text input field says "What are you doing right now?" with a "Post" button. The main content area of the profile page is visible below these elements.

Figure 12-7. Adding GameNomad's Video Games Tab to a Facebook Profile

Configuring the Facebook Application to Support a Tab

In the interests of reducing redundant code, we're going to reuse the Facebook Controller's profile view which comprises the tools we gave to the logged-in user. To do this, just set the **Profile Tab URL** value (Figure 12-8) within the application settings to <http://apps.facebook.com/gamenomad/games>. Next, set the **Profile Tab Name** value to **Video Games**.

**Figure 12-8. Configuring the tab URL and name in the application settings**

Disabling the Authorization Requirement for Tab View

Because any Facebook friend could conceivably view the Video Games tab, we need to figure out how to disable authorization when any user is viewing the Facebook controller's games method. To do this, we'll need to first modify the Facebook Controller's `init()` method to forego authorization of the user when in tabbed mode. Doing so is surprisingly easy, because the Facebook platform will automatically set the `$_POST['fb_sig_in_profile_tab']` variable should a tab page be requested. Therefore all you have to do to forego authorization is check that variable. If not set, perform the authorization:

```
$this->view->onTab = isset($_POST['fb_sig_in_profile_tab']);
if (! $this->view->onTab) {
    $this->facebook->require_login();
}
```

Rather than just create a local variable, the `$onTab` variable was set in the view context in order to later be able to reference it within the Facebook controller's `games` view. I use the variable to display a special message introducing GameNomad and inviting users to join, as well as hide the logged-in user's tool navigation menu.

Enabling the Profile Tab

Even when configured within the application's settings interface, it's not possible to automatically add a tab to the user's profile. Instead, the user must explicitly enable it through the application settings interface. To access this interface, navigate to your applications menu by clicking on the **Applications** button at the bottom left of your Facebook page. From there, click on **Edit**, and in application listings page which appears, click on the **Edit Settings** link next to your application. In the window which pops up, click on the **Profile** tab (Figure 12-9).

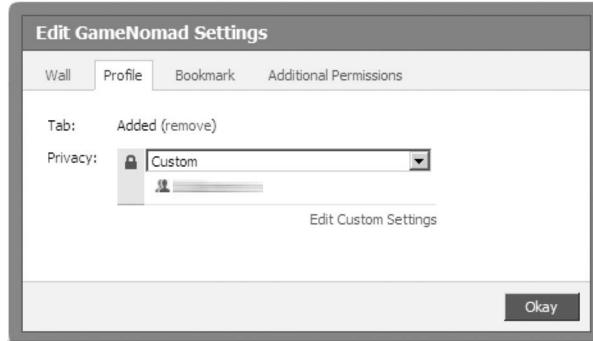


Figure 12-9. Tab addition authorization takes place in the application settings interface

Within this tab the user will first need to first specify the tab should be added to your profile page, and in the Privacy drop-down, he can set an appropriate visibility level, specifying the Video Games tab be visible to on the user, only select users (Custom), only friends, all friends and their friends, or all of the user's networks and friends. Once added, the tab and its contents will be immediately visible to the allowed parties.

Incidentally, when creating the application, it's useful to set the Privacy setting to Custom, and select from your friends a list of fellow developers and other friends interested in testing the application. This way you can safely test the profile tab and its contents without opening up the application to others before it's complete.

Step #7. Sending a Facebook User Notification

It might be useful to send notifications to a user's Facebook friends who are also members of GameNomad whenever the user performs some sort of newsworthy action, such as adding a new game to his collection. An example of such a notification is shown in Figure 12-10.



Figure 12-10. Notifying a user of a friend's game collection addition

To send these notifications, call the Facebook client's `notifications_send()` method. For instance, the following call will send a notification about a new addition to Scott's game collection to my Facebook account:

```
$this->facebook->api_client->notifications_send('501632489',
    'Scott added Call of Duty: World at War (Xbox 360) to his collection',
    'user_to_user');
```

To incorporate this feature, all you need to do is add the method call at the appropriate location.

Of course, sending out too many notifications could turn this informative feature into an annoying one, so consider reserving this feature for only the most important of updates. Facebook makes it possible for users to disable notifications altogether for a specific application through the notifications manager, and in cases of perceived abuse of the system, can identify an application as a spammer.

Step #8. Adding Facebook Status Updates

One of Facebook's killer features is the status update, used to let friends know what's currently going on in your life. These days, there seems to be no more effective way to keep tabs on what flavored coffees your network is drinking, or whose children are currently suffering a bout of pinkeye. Facebook's status update feature can serve somewhat more practical purposes though, such as letting your network know about a new game you're currently playing.

Granting Permissions

Because posting an update on a user's behalf has such visible repercussions, the user must explicitly grant GameNomad permission to perform this task. GameNomad does this via the Facebook controller's privacy method, but what's so convenient about Facebook permissions is that we don't have to bother with managing the permissions because Facebook will manage them for us. In fact, using a bit of FBML, we can dynamically include a link prompting the user to grant GameNomad permission to post status updates. What's particularly nice about using FBML is that if the user has already granted permission, Facebook will automatically override rendering the FBML! What's more, using a Facebook client call, you can determine whether the user has already granted the permission, and if so display a message explaining how to disable the permission. The following listing demonstrates these concepts:

```

01 <?php
02 if (! $this->facebook->api_client->users_hasAppPermission('status_update')) {
03 ?>
04
05 <p>
06 You've granted permission to GameNomad to post status updates on your behalf.
07 To remove this permission, navigate to your GameNomad
08 <a href='http://www.facebook.com/editapps.php'>application settings</a>.
09 </p>
10
11 <?php } ?>
12
13 <fb:prompt-permission perms="status_update">
14 Grant permission for status updates
15 </fb:prompt-permission>
```

A breakdown of the above listing follows:

- Line 02 determines whether the user has already granted the application permission to perform status updates on the user's behalf. This is just one of several available permissions;

check the Facebook documentation for more information. If the user has granted permission, lines 05-09 are output. Note this method doesn't exist in the official client! Later in this section I'll show you how to add it to the class.

- Lines 13-15 display a link prompting the user to grant permission to GameNomad to perform status updates. If the user has already granted this permission, Facebook will conveniently forego displaying the link altogether!

When the user decides to enable updates, he'll be greeted with a prompt asking him to confirm this request (Figure 12-11).

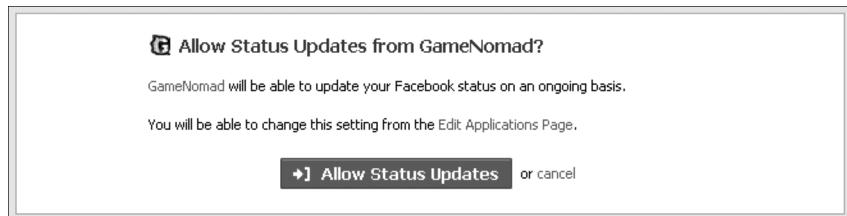


Figure 12-11. Enabling Status Updates

If the user does later decide to disable status updates, he'll have to navigate to <http://www.facebook.com/editapps.php> and edit GameNomad's settings. The application's settings window will appear (Figure 12-12). From there, he can click the Additional Permissions tab, and disable status updates.

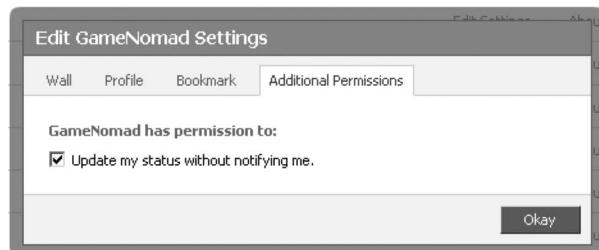


Figure 12-12. Disabling Status Updates

Publishing Status Updates

Oddly, the Facebook PHP client does not currently support the ability to post status updates. However, modifying the client to support this ability is easy! Open up the `facebookapi_php5_restlib.php` file residing in the Facebook client directory, and add the following two methods at an appropriate location.

```
/**  
 * Sets a user's Facebook status  
 *  
 * @param string $status The status update to post  
 * @param boolean $clear Clear an existing message matching $status
```

```

 * @return boolean
 */
public function users_setStatus($status,$clear)
{
    return $this->call_method('facebook.users.setStatus',
                                array('status' => $status,'clear' => $clear));
}

/**
 * Determines whether a user has granted a particular Facebook permission
 *
 * @param string $permission The permission in question
 * @return boolean
 */
public function users_hasAppPermission($permission)
{
    return $this->call_method('facebook.users.hasAppPermission',
                                array('ext_perm' => $permission));
}

```

The code comments should be suffice to explain the purpose of each new method. So how do you subsequently send an update to the user's Facebook profile? The following example shows how:

```

if (! $this->facebook->api_client->users_hasAppPermission('status_update')) {
    $this->facebook->api_client->users_setStatus('added Halo 3 to his game
                                                    collection.', FALSE);
}

```

Step #9. Deploying Your Facebook Application

You've thoroughly tested your new application, braced your Web server for the onslaught of new users, and told your family they won't be seeing you for a few days. It must be application launch time! Launching your Facebook application is a very easy process; just navigate to <http://www.facebook.com/developers/>, choose the application you'd like to launch under the "My Application" section, and click the submit button! Before the application is added to the Facebook directory though, you're required to provide five pieces of information if you haven't already done so, including the application name, contact e-mail address, a short application description, logo, and whether the application uses Facebook's mobile platform.

Furthermore, your application must have at least five users before Facebook will accept it. I'd imagine this provides at least a modicum of proof that you're serious about the application, and have recruited friends and fellow developers to thoroughly test the project before launching it.

Once launched, it's time to get the word out! Be sure to update your Facebook status to point your Facebook network to the new venture/feature, not to mention let your existing website community know about the new feature. Ask your friends and colleagues to mention the application on their respective Facebook networks. Before you know it, the users will be streaming in!

Conclusion

This chapter introduced RSS and the Facebook Platform, two very hot technologies which each play a major role in connecting with your users. In fact, in terms of establishing a long term relationship with your community, the topics discussed in this chapter may be the most important in the entire book.

Speaking of maintaining connections, the next chapter discusses the Google Analytics traffic analysis service, and the Google AdWords and Google AdSense advertising services. These crucial technologies which will no doubt have a significant impact on your website's operations, both from a technological and business standpoint.

CHAPTER 13

Monitor Traffic and Manage Ads with Google

Sales and marketing guru Orvel Ray Wilson famously opined, "Customers buy for their reasons, not yours." Your challenge is to figure out what those reasons are, and then construct, communicate, and refine a message that echoes those reasons. As applied to the web, you'll specifically need to conduct online marketing campaigns and measure the effectiveness of website content by monitoring visitor traffic and navigation behavior. To help businesses carry out these important tasks, numerous utilities and services have been created for helping companies effectively undertake these complex tasks.

In this chapter I'll introduce you to Google Analytics and Google AdWords, two de facto solutions for analyzing visitor traffic and managing online advertising. I'll also show you how you can start earning revenue by publishing advertisements on your own website using the popular Google AdSense advertising service.

Chapter Steps

The goals of this chapter are accomplished in three steps:

- **Step #1. Monitoring Traffic with Google Analytics:** In this opening step you'll learn how to analyze website traffic using the powerful Google Analytics package.
- **Step #2. Advertising with Google AdWords:** You'll eventually want to begin spreading the word about your website in an effort to increase traffic. One of the most effective ways for doing so is by advertising online through the Google AdWords network, and in this step you'll learn all about it.
- **Step #3: Earn Money Using Google AdSense:** Google's AdSense service offers a simple and effective solution for publishing ads on your website, and in this step you'll learn how.

Step #1: Monitoring Traffic with Google Analytics

The philosopher George Santayana once observed, "Those who cannot remember the past are condemned to repeat it." The same reasoning applies to your website traffic; if you are unable to monitor and analyze the successes and failures of your website content, how can you plan for the future?

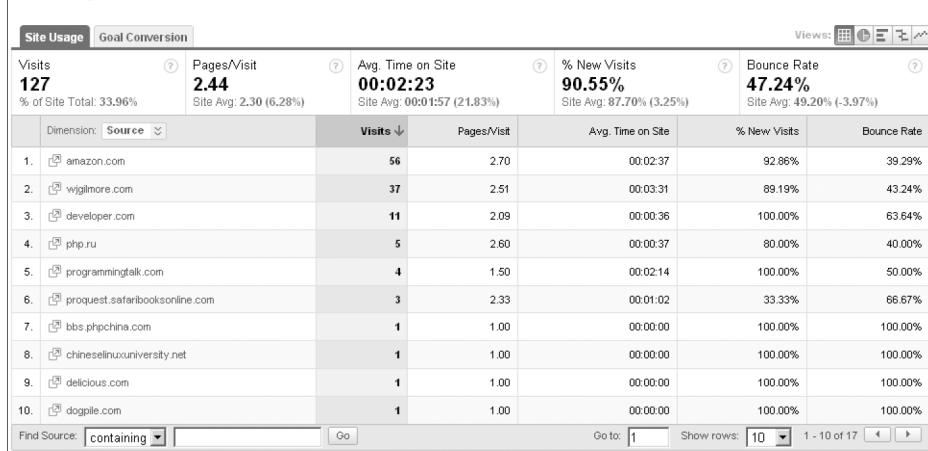
For instance, if you launch a new website feature such as a downloadable PDF newsletter, you'll logically want to actively monitor visitors' interest in the newsletter, most notably by tracking the number of times it's downloaded. However, even this raw number is limited in utility. Wouldn't it be fantastic to know what parts of the world these visitors hailed from, even broken down to the state and city? How about what languages they speak? The number of times these visitors return over a defined time range? Using Google's traffic analysis service, known as Google Analytics, you can do all of this and much, much, more. Once configured, you'll have one of the most powerful traffic analysis solutions in the world at your fingertips, capable of quantifying nearly every measurable aspect of your user traffic. For instance, using Google Analytics you can track:

- **Overall traffic trends:** Keep tabs on the total number of visitors, page views, pages navigated per visit, and average time visiting the site according to numerous temporal ranges, including all-time, per-month, per-week, and per-day basis.
- **Visitor locales:** Know where your visitors are coming from by sifting through traffic records according to content, country, continental regions, U.S. states, and cities. Analytics will also examine the users' browser character encoding to determine their preferred language. See Figure 13-1 for a screenshot of a website's recent traffic broken down according to city.
- **Visitor technical profiles:** Visitors are segmented by technical characteristics such as browser type, screen resolution, connection speed, operating system, and whether their computer supports Java and Flash.
- **Visitor loyalty trends:** Learn about first time and returning visitors, the length of visitation, and the number of pages visited.
- **Traffic sources:** Identify the origin of your traffic by monitoring search keywords resulting in visitation, as well as top referrers. See Figure 13-2 for a screenshot.
- **Content popularity:** Know what content works and what doesn't by monitoring the visitation frequency of every page on your site.

NOTE. Google Analytics can tie into another popular Google service, Google AdWords. By taking advantage of this possibility, you can measure the effectiveness of your advertising campaigns taking place over the Web, print, radio, and television. You can also configure and monitor sophisticated conversion goals, determining whether and how often a visitor completes a specific task such as making a purchase or downloading a file.



Figure 13-1. Identifying your visitors' locations by city

Referring sites sent 127 visits via 17 sources**Figure 13-2. Understanding Inbound Traffic Sources**

Although Analytics' features are so vast that an entire book could be written on the topic, this section should provide you with enough information to not only convince you of the solution's merits, but also help you to quickly begin exploiting several of Analytics' compelling options.

VIDEO. A Tour of Google Analytics

After all the time you put into building a great website, you're going to want to effectively monitor important matters such as site traffic and user demographics. There's perhaps no better available tool to do so than Google Analytics. This video introduces this fantastic service, showing you how to plug it into your website in just minutes, and guiding you through its key features. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Configuring Google Analytics

Beyond the valuable features, one reason for Google Analytics enormous popularity is the incredibly easy installation process. In this section I'll show you how to integrate Analytics into your website in mere minutes.

Creating a Google Account

Google requires you to create an account before using many of its services, Google Analytics included. If you don't yet have an account, navigate to <https://www.google.com/accounts/NewAccount> to create one. You'll have to confirm your e-mail address by clicking on a confirmation link sent to you following registration, so be sure to provide a valid address. Once your account has been created and confirmed, proceed to the next section.

Adding Your Website to Google Analytics

To begin monitoring your website traffic with Google Analytics, head over to <http://www.google.com>.

com/analytics/ and login with your Google account. Once logged in, you'll be prompted to create a new Google Analytics account by providing key information such as your website's URL, a desired account name, your name, country, and time zone. Finally, you'll be prompted to read and accept the terms of service, before being presented with a block of JavaScript code. This code is what Google uses to track and analyze user traffic, and so must be present on *every* page of your website!

Of course, the easy solution is to just paste the code into your website's `layout.phtml` file. I tend to place the code right before the closing HTML tag, like so:

```
<script type="text/javascript">
  var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." :
                  "http://www.");
  document.write(unescape("%3Cscript src='" + gaJsHost +
    "google-analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
</script>

<script type="text/javascript">
try {
  var pageTracker = _gat._getTracker("UA-XXXXXXX-X");
  pageTracker._trackPageview();
} catch(err) {}
</script>

</body>
</html>
```

Once inserted, it will take up to 24 hours before data collection will begin, so don't fret if your reports are initially blank. If you're copying directly from the book, be sure to replace the UA-XXXXXXX-X with your unique analytics code.

Using Google Analytics

The Google Analytics' interface is very user-friendly, and frankly you're going to quickly become proficient simply by logging in and playing around with the service. There are however a few features which you'll probably want to begin exploiting immediately, yet how to do so might not initially be so obvious. In this section I'll introduce several of these features, and show you how to implement each.

Granting Access to Your Colleagues

Particularly when working in a team environment, you're probably not going to be the only one wishing to view the Analytics data. You can give other users access to the service in just a few steps:

1. Login to Analytics
2. Choose the desired account by clicking on its name within the Accounts table.
3. Click the User Manager link, located at the bottom of the page.

4. Click the User Manager link at the bottom of the page. On the page that appears, click the Add User link located at the top right of the Existing Access table.
5. Insert the user's e-mail address, and select view-only or administrator access. This e-mail address must be tied to a Google account.
6. Grant the user access to the desired website profiles by highlighting their names and clicking the Add button. If you've identified the user as an account administrator, he will be automatically granted administrator access to all reports in this account.
7. Click the Save Changes button

Next, direct the user to <http://www.google.com/analytics>, where he can login and begin accessing the reports in accordance with the level of privilege granted by you.

Exporting Reports

Analytics supports exporting the data found on any page using a variety of formats, including PDF, XML, CSV (comma-separated values), and TSV (tab-separated values). These alternative formats are useful if you'd like to pass the data to your own custom reporting programs for further processing.

To export a page's data, click on the ubiquitous Export button located at the top of each page, and choose the appropriate format, as shown in Figure 13-3.



Figure 13-3. Selecting an export format

Once the data has been exported, your browser's download prompt will appear. From here you can choose to open or save the file.

Emailing Reports

If you're dealing with the pointy-headed boss, chances are you're going to e-mail an exported report. Analytics takes this inevitability into account by providing you with a mechanism for e-mailing an exported file directly from the website. To e-mail an exported file, click the ubiquitous E-mail button located at the top of each page, and complete the form (shown in Figure 13-4).

The screenshot shows the 'Email' interface for Google Analytics. At the top, there are three buttons: 'Send Now', 'Schedule', and 'Add to Existing'. Below these are fields for 'Send to others' (with a note about comma-separated addresses), 'Subject', 'Description', and 'Format' (PDF is selected). A 'Word Verification' section contains a CAPTCHA image showing 'ansklua' and a text input field. At the bottom right is a 'Send' button.

Figure 13-4. Emailing an exported file

You can also schedule reports to be emailed on a daily, weekly, monthly or quarterly basis. To do so, click the aforementioned **Email** button to retrieve the e-mail interface. From there, click the **Schedule** tab located at the top of the interface. Complete the form (shown in Figure 13-5) and press the **Schedule** button to schedule the e-mail.

The screenshot shows the 'Schedule' interface for Google Analytics. It includes fields for 'Send to others' (with an example address), 'Subject' ('Latest Analytics Report for BeginningPHPandMySQL.com'), 'Description' (a message about the traffic report), 'Format' (PDF is selected), and 'Date Range/Schedule' ('Weekly (sent each Monday)'). There is also a 'Include date comparison:' checkbox and a 'Schedule' button at the bottom right.

Figure 13-5. Scheduling an e-mail for weekly delivery

Blocking Your IP Address

Because you'll logically regularly peruse your own website to ensure everything is working properly, surfing with such frequency could skew Analytics' traffic analysis. I recommend filtering your IP or office's IP range from the results. To do so, follow these instructions:

1. Login to Analytics
2. Choose the desired account by clicking on its name within the Accounts table.
3. Click the **Filter Manager** link, located at the bottom right of the page
4. Click the **Add Filter** link
5. Create a filter name (for instance "Jason Home IP Address") and set the filter type to "Exclude all traffic from an IP address". Insert your *external* IP address, and then specify which websites should heed the filter.

NOTE. These days you'll rarely be operating a computer having a direct connection to the Internet, meaning your computer will likely be assigned an IP address belonging to an internal network, such as 192.168.1.101. You can however easily determine your external IP address by navigating to <http://myip.wjgilmore.com>.

Tracking Download Frequencies

Suppose your marketing team created a downloadable brochure highlighting a new array of products. By default Analytics doesn't track downloads, but can do so with a simple modification to the downloadable file's hyperlink. This modification involves creating a custom pageview title which Google will use as the placeholder for monitoring traffic to that download. For instance, suppose the brochure was titled `whizbang.pdf` and located at `http://www.example.com/files/whizbang.pdf`. To begin tracking the download frequency, you would modify the PDF download link to look like this:

```
<a href="/files/whizbang.pdf"
  onClick="javascript:pageTracker._trackPageview('/downloads/whizbang'); />
```

Of course, the Analytics tracking code must also be present on this page.

Tracking Outbound Clicks

Many websites partner with others in a collaborative effort to build traffic by trading links. You can track the frequency in which visitors access these links by modifying the URL to include a custom pageview title, using the same syntax as that found in the previous section. For instance, to track the number of users navigating to the outgoing link `http://www.wjgilmore.com/`, you would modify that hyperlink like so:

```
<a href="http://www.wjgilmore.com/"
  onClick="javascript:pageTracker._trackPageView('/outgoing/wjgilmore.com'); />
```

Be sure the Analytics tracking code is also present on this page so the `_trackPageView` method can be called.

Step #2. Advertising with Google AdWords

The constant prodding of family members, friends and colleagues to let others know about your website has been pretty effective, and using Google Analytics you're watching the traffic grow by the day. However, given your boundless ambition, you won't be satisfied until your website ranks among the most popular in the world. To obtain such lofty heights, you'll need to reach beyond your inner circle and target a larger potential audience.

One of the most effective ways to do this is through online advertising, and there's no greater online advertising solution than Google AdWords. Launched in 2000, Google AdWords has since become the largest online advertising network in the world, and is responsible for the lion's share of Google's revenue. Its success is due to the solution's tremendous power, flexibility, and cost-effectiveness, offering features such as:

- **Powerful campaign management:** Whether you're interested in using AdWords to advertise a single product or service, or want to create a complex promotional campaign involving banner variations, keyword filters, and regional targeting, AdWords offers the tools required to effectively create and manage a successful campaign.
- **Cost controls:** Newcomers to online advertising are often wary of advertising expenses, wondering how much it costs to display an ad, and whether it costs more if the advertisement is clicked. AdWords removes such concerns by allowing the advertiser to specify a maximum desired cost-per-click, with willingness to pay a higher fee for the click playing a factor in the ad's position. Advertisers can also set a maximum daily budget which when reached will halt the campaign until the start of the next day.
- **Advanced reporting solutions:** Rather than login to the AdWords site and sift through the lengthy benchmarking data, you can create reports highlighting the performance of campaigns, keywords, search queries, and demographics. You can also arrange for these reports to be e-mailed to one or several recipients attached as a CSV, HTML, TSV, or XML document.
- **Multiple online advertising formats:** AdWords supports a variety of advertising formats, including the default text-, image-, and video-based ads.
- **Localized Placement:** If your goal is to drive individuals to a physical business location, you can take advantage of AdWords' local business ad placement service. This feature places your ad on Google Maps when appropriate, and will also display your ad when users perform context- and location-specific searches, or when Google identifies the user's originating IP address as being closely situated to your business address.
- **Regional and Language-specific Placement:** While certain international organizations wish to seek a worldwide following through online advertising, AdWords recognizes the desire for many advertisers to limit advertising to select regions or languages. In addition to giving advertisers the ability to limit their campaign to one or several of 40 supported languages, it's

also possible to constrain the display of advertisements to users hailing from certain continents, countries, states, cities, and even zip codes. It's even possible specify a certain radius limitation surrounding a particular location.

- **Multiple advertising venues:** Although AdWords was originally created as an online advertising-specific network, Google has since expanded the program to support radio and television advertising. For instance, Google Audio Ads program makes it possible for you to advertise on over 1,600 FM and AM radio stations around the country.

In this step I'll show you how to get started with Google AdWords, introducing you to the program's key features and showing you how to tie AdWords into your Google Analytics account in order to better gauge an ad's effectiveness.

Creating an AdWords Account

Creating an AdWords account is free; payment is only due after your advertising campaign has launched, and visitors begin clicking on your advertisements. Before you can begin using the service however, you'll need to create an account. Begin by heading over to <http://adwords.google.com/> and clicking on the **Start now** button. Next you'll be prompted to choose from one of two AdWords programs:

- **Starter Edition:** The Starter Edition is geared towards advertisers interested in promoting a single item or service. Chances are you'll want to choose this edition, as it offers the easiest path for getting started. This version does however only offer text-based ads, whereas the Standard Edition offers numerous advertising formats. You can always upgrade to the Standard Edition as your advertising requirements grow in size and complexity. The information provided throughout the remainder of this step is based on the presumption you've chosen this edition.
- **Standard Edition:** The Standard Edition is ideal for more experienced advertisers interested in promoting multiple products or services and wishing to manage potentially multiple advertising initiatives. Among other features not available through the Starter Edition, the Standard Edition gives you the ability to identify specific websites for ad placement.

Once you've chosen a program, you'll next be optionally prompted to provide your country and phone number. If your business is already listed in the Google business directory (<http://www.google.com/local/add>), Google will use the information to cut down on the amount of information you'd otherwise need to redundantly provide when completing the AdWords registration process.

In the next step you'll start creating the first advertisement! You'll be asked to provide several key pieces of information, such as the location of your customers, and the advertisement's language. This information is important because it will help Google to determine who will see the advertisement. For instance, if your organization sells athletics memorabilia for teams based in the state of Ohio, Google will not show your advertisements to individuals residing in Europe, provided you identify your customer base as residing in Ohio.

You'll also identify the URL your advertisement will link to, the advertisement's headline and corresponding message, and the context keywords which will cause your advertisement to appear with the

Google advertising network. The wizard will automatically render an example of what the text-based advertisement will look like, as depicted in Figure 13-6.

2. Write your ad

What site will your ad link to?
Users who click your ad will be sent to this web page.

Example: <http://www.AdWordsExample.com/products/item.htm>

What will your ad say?
All text ads contain a title, two lines of descriptive text, and a display URL. Make sure to include information that will help customers understand your business.
[The five keys to powerful ads](#) | [Editorial Guidelines](#)

Easy PHP Websites
Learn PHP fast with the ultimate book + video package. Just \$35!
www.easyphpwebsites.com

This is how your ad will look.

Easy PHP Websites [25 max]
Learn PHP fast with the ultimate [35 max]
book + video package. Just \$35! [35 max]

<http://www.easyphpwebsites.com> [35 max]

Figure 13-6. Creating your advertisement text is easy using the wizard

To complete this step, you'll be prompted to identify the currency you'll use to pay for the advertising, and your monthly advertising budget. You can set this to whatever monetary value monetary value you'd like, and change the value later as your budget fluctuates.

Finally you'll need to setup an account tied to this advertisement. You'll be prompted to either sign into your existing Google account, or identify a new e-mail address and password for logging into this account. For reasons of convenience I suggest using the same account you used for the Analytics account.

Activating Your Account

Once you've created the account, Google will send a verification e-mail to the address you've provided. If you created a new account, you'll need to click on a URL provided in the e-mail to activate the account. If the account already existed prior to registering for AdWords, you'll just need to login by navigating to <https://adwords.google.com/>. In either case, you'll be transported you back to the Google AdWords website where you'll complete the registration process by providing your billing information. In these final steps you'll be prompted to determine whether you'd like to prepay or post-pay for the advertising. I suggest the latter option, using a credit card or direct debit to be debited the cost of your advertising on a 30-day billing cycle. Regardless of which payment option you choose, Google will charge you a one-time \$5 setup fee once the activation process is complete.

Next you'll be prompted to agree to Google's Advertising Program Terms. Take some time to read this document, and if acceptable, click the appropriate radio button to signify your agreement and click the **Continue** button to continue.

Finally, you'll be prompted to provide your billing information. This information will vary slightly based on the payment format, however presuming you chose a credit card you'll be prompted to

identify the credit card type, number, verification code, card holder's name, and expiration date. You'll also need to provide the billing address, and answer a few optional questions regarding the purpose of your advertising. Once completed, click the **Save and Activate** button, at which point your advertising campaign will immediately begin!

Managing Your Campaign

The Google AdWords campaign manager provides you with a powerful set of tools for monitoring the campaign performance on a per-keyword basis, and tallying the total costs accrued during the current billing period. The main interface for managing a campaign is shown in Figure 13-7. I've annotated this screenshot with numbered callouts to key elements you'll want to understand. Following the screenshot I've summarized these callouts.

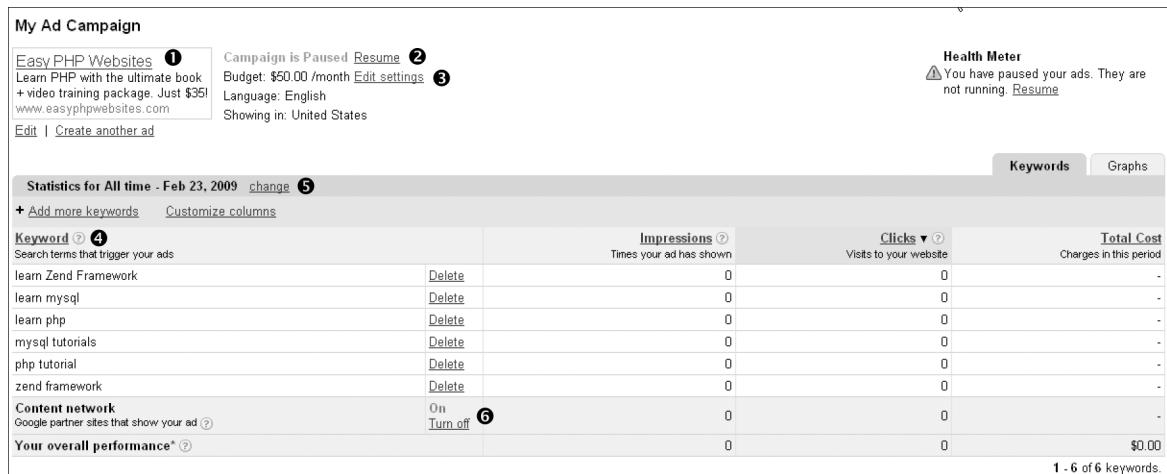


Figure 13-7. Monitoring keyword efficiency of a new campaign

There are six callouts to key features of this manager interface, each of which is explained here:

- **Callout #1. The advertisement:** This callout identifies the advertisement exactly as it appears to users perusing the Google content network. Be sure to periodically review your ad to ensure there are no spelling errors or other misstatements. To edit the advertisement's URL destination, or edit the ad title or text, click the **Edit** button located below the advertisement. If you click on the **Create another ad** link, you'll be prompted to do one of two things: create a variation of the existing ad, which will run in accordance with the existing set of keywords, or create an entirely new ad intended to promote another product. If you choose the latter, you'll need to upgrade to Standard Edition, which is free to do but comes at the cost of a slightly steeper learning curve.
- **Callout #2. The toggle switch:** You can pause and resume your campaign by clicking the context-sensitive link located here. As you can see, this campaign is currently paused. To resume it, just click the link, at which point Google AdWords will immediately begin serving the ad anew.

- **Callout #3. Campaign settings:** Although oddly placed next to the campaign budget specification, clicking the **Edit settings** link will actually take you to a page controlling not only budgetary matters, but also the specified native language and location of the users who will view the advertisement. I'll talk more about how to make the most of your budget later in this section.
- **Callout #4. The campaign keywords:** The beauty of Google's AdWords program is that advertisements are served in context-specific settings, not only in accordance with predefined language and location constraints, but also with the content being currently viewed by the consumer. Logically, if the advertisements are somehow related to the material currently being viewed by the consumer, it's more likely the consumer will be interested and investigate the advertisement. As an advertiser, you can use this interface to monitor and manage those keywords which define the context. From here you can add and delete keywords, learn more about each keyword's efficacy by comparing the number of triggered impressions to the number of clicks, and monitor the total cost incurred due to using these keywords for the specified period.
- **Callout #5. The date range:** You can change the range of dates used to summarize keyword efficacy and cost by clicking on the **change** link next to the date.
- **Callout #6. The content network:** By default your advertisement will not only appear as appropriate within the ubiquitous vertical column located to the right of Google search results, but also within relevant sites comprising Google's enormous content network, including AOL, Ask.com, Netscape Netcenter, The Food Network, About.com, and thousands of other websites run by users taking part in the Google AdSense program (discussed in Step #3). You can force AdWords to exclusively show your advertisement within searches executed on Google.com by turning the content network off here.

Tweaking the Budget

When using the Starter Edition, Google AdWords helps new users control costs by promising to never exceed the budget you identified when setting up the AdWords account. You can change this maximum monthly expenditure by clicking on the **Edit settings** link identified by callout #3 of Figure 13-7. Using a feature known as the *Budget Optimizer*, Google AdWords will autonomously manage your cost-per-click in an attempt to achieve the maximum number of clicks possible within the constraints of your pre-defined monthly budget.

You can however play a more active role in your cost-per-click exposure by clicking on the **Edit settings** link, disabling the Budget Optimizer, and setting a maximum cost-per-click value in its place. You might want to do so in an attempt to ensure you have submitted the highest-possible bid for a keyword, thereby solidifying your advertisement's position at the top of the ads competing for space, and subsequently increasing the likelihood it will be clicked by an interested user. What's particularly nice about this feature is that you will not always pay the maximum specified value with each click. Instead, Google AdWords will determine the amount of the second highest bid, and adjust your bid to just \$0.01 above that bid! Regardless, before disabling the Budget Optimizer, I suggest taking some time to learn more about the nuances of keyword advertising and the Google AdWords network.

If you find this idea of tweaking the cost-per-click intriguing, I suggest upgrading to the Standard Edition as it offers far more powerful tools for managing per-click expenses than are available with the Starter Edition. Among other things you can set per-click bids for each keyword, thereby wielding total control over the conditions under which your advertisement will be served.

Tying Google AdWords to Your Google Analytics Account

The Google Analytics and Google AdWords services each offer tremendously powerful solutions for analyzing website traffic and managing online advertising campaigns, respectively. However, the Google team has taken great pains to ensure these services can be used together to produce a truly compelling service which helps you to maximize all of your online efforts. By integrating AdWords into your Analytics account, you'll be able to closely monitor the demographics and behavior of users who visited your website by way of an ad campaign.

Configuration Prerequisites

In order to link your Google AdWords and Google Analytics accounts, you'll need to first upgrade your AdWords account to the standard edition if you haven't already done so. Doing so is easy; just login to your AdWords account and click on the [Graduate to Standard Edition](#) link located at the bottom left-hand corner of the page. Doing so doesn't delete any of the advertisements or analytical data collected thus far; migrating just grants you several additional capabilities, among them the ability to link AdWords data to the Analytics platform.

If you want to link your AdWords to an existing Analytics account, you'll also have either used the same Google account for both services, or have added your AdWords account to the Analytics account as an administrator. I showed you how to add an administrator account in Step #1, in the section "Granting Access to Your Colleagues".

Once you've upgraded to the Standard Edition, click on the [Analytics](#) tab located at the top of the page. From here you'll be able to create a new Google Analytics account if you don't already have one, or specify you've already created the account and would like to link to it. Presuming the latter, you'll next be prompted to identify the Analytics account you'd like to link to, and also optionally enable two options:

- **Destination URL Auto-tagging:** Enabling this feature (the default) will cause AdWords to automatically append a unique ID to each advertisement's destination URL, helping you to more effectively analyze the efficacy of the advertisement. Alternatively you can choose to specify custom tracking parameters by disabling this feature and appending the necessary data to the advertisement's destination URL. I suggest leaving this option enabled unless you have specific needs regarding URL formatting.
- **Apply Cost Data:** Enabling this feature (the default) will also integrate your AdWords cost data into Analytics, helping you to further understand the cost-efficiency of each campaign. I suggest leaving this option enabled.

Once enabled, you'll be able to begin tracking all sorts of useful information regarding your AdWords campaigns, a process made even more streamlined because the Analytics interface is now integrated

directly into your AdWords interface. Navigate to your website's AdWords analytical overview by clicking on the Traffic Sources tab located on the left-hand menu, then on the AdWords submenu, as shown in Figure 13-8.

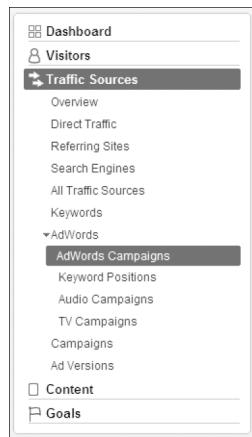


Figure 13-8. You'll find links to AdWords data under the Traffic Sources section

Clicking on the AdWords Campaigns menu item, you'll be able to analyze traffic originating from your advertisements just as you would any other page using Google Analytics. A sample page of such analysis is shown in Figure 13-9.

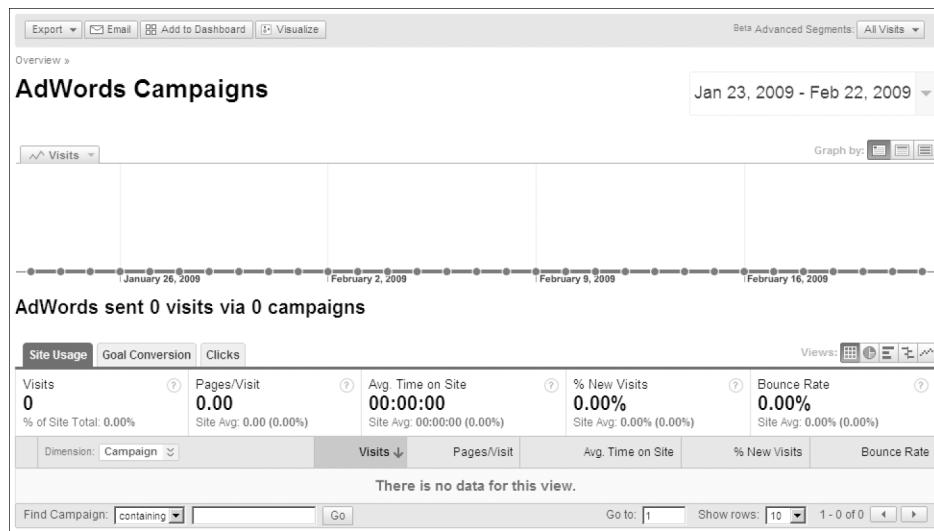


Figure 13-9. Monitoring campaign effectiveness is easy using Analytics

Clicking on the Keyword Positions menu item, you'll be able to keep tabs on the clickthrough efficiency of keywords according to the advertisement's position within search results pages. Among other things, this will help you to determine whether your maximum keyword bid should be adjusted in order to achieve the maximum possible return on investment.

Step #3: Earning Money with Google AdSense

Although your website's popularity continues to soar, you've yet to make a dime from it. Something seems wrong with this picture, doesn't it? The remainder of this book seeks to change that by devoting the remainder of this chapter to showing you how to make money through advertising, and in the next chapter by accepting payments for products using PayPal. Turning to the former effort, in this step I'll show you how to integrate relevant, non-intrusive advertising into your website by taking advantage of the world's most popular advertising network, namely Google AdSense.

I'd attribute Google AdSense's meteoric rise in popularity to two key characteristics. First, it serves only context-specific advertisements on your website, meaning your visitors are more likely to pay attention to them. For instance, on GameNomad.com you'll find advertisements pertaining to topics such as online gaming websites, gaming hardware and consoles, and science fiction. Second, the advertisements are non-intrusive. Annoying pop-ups and auto-expanding advertisements are not allowed, and you can even control which types of advertisements are published on your site (choosing from text-based, image, link units, video, and gadgets).

Publishers are most commonly paid on a per-click basis, meaning you'll be paid every time a visitor clicks on an advertisement. The payment rate varies, determined by the rate bid by way of a competitive bidding process. Payments are issued when the AdSense user's account accrues more than \$100 in unpaid earnings.

Tip. If you're operating in a particularly competitive industry, you can even use the AdSense Competitive Ad Filter to prevent your competition from displaying advertisements on your website!

Joining the AdSense Program

Joining Google AdSense is free, and only takes a few minutes to do. To register, navigate to <http://www.google.com/adsense> and click the Sign up now button to begin the registration process. You'll be required to provide the domain which will serve the advertising, along with payment information and various reassurances such as your promise to not serve advertisements on pornographic websites, nor click on the advertisements in an effort to boost your revenue share.

Once submitted, Google will review your application and follow up via e-mail with instructions. According to the AdSense documentation, turnaround time is typically 2-3 days, although it could take up to a week to respond. My experience has shown a faster turnaround time of less than 24 hours.

TIP. Google will not approve applications for websites they deem to still be under construction. Therefore be sure to apply for your AdSense account only after launching a reasonably complete version of your website.

Adding Advertisements to Your Website

Once approved, you'll be able to login to the AdSense control panel (<http://www.google.com/adsense/>) to begin managing your advertisements. AdSense currently offers six advertising products:

- **AdSense for Content:** The most popular of all the products, AdSense for Content allows you to publish advertisements matching the content of your website and general interests of your audience.
- **AdSense for Search:** If your website includes a search feature, you can use this product to present advertisements matching the context of the user's search keywords.
- **AdSense for Feeds:** If your website offers RSS content, you can use this product to integrate text-based advertising into the feed. This is accomplished by passing along the feed's current URL, which Google will then use to create an ad-infused version which you'll subsequently point future users towards. If you're using the popular FeedBurner RSS service, Google also offers an easy way to move those feeds to Google.
- **AdSense for Domains:** If you're currently doing nothing with that brilliant domain name you purchased after a night out at the bar a few months back, you can use the AdSense for Domains product to publish advertisements on the domain until you find a better use for it.
- **AdSense for Video:** One of AdSense's newest offerings, AdSense for Video offers advertisers a cutting-edge way to communicate with consumers by way of video content. If you think your website audience may have an affinity for video-based material, this may be an interesting way to capture new revenue.
- **AdSense for Mobile Content:** With the evolution of popular internet-enabled phones such as the Apple iPhone, and Google G1, chances are you'll soon be considering building a mobile version of your website, if you aren't already doing so. AdSense for Mobile Content gives you the opportunity to host advertising on your mobile website, using a format conducive to the smaller screen form factors and lower bandwidth requirements.

While I invite you to investigate the latter five options, the remainder of this chapter focuses on what is generally the most popular of the bunch: AdSense for Content. Click the AdSense for Content link to begin the process of integrating ads into your website.

Google's AdSense for Content feature allows you to choose from one of two advertising units: Ads or Links. *Ad units* (Figure 13-10) comprise the text- and image-based ads you've surely grown accustomed to seeing while navigating the Web, whereas *Link units* (Figure 13-11) are the less commonplace lists of relevant topics based on the page context (Figures 13-10 and 13-11 are sample ads located at https://www.google.com/adsense/static/en_US/AdFormats.html). Clicking on a topic takes the user to a page of related advertising. Presuming you're interested in Ad units, select the corresponding radio button and click the Continue button to proceed. You also have the option of selecting the ad format, which defaults to text and image ads.

TIP. First introduced in Chapter 1, the Firefox Web Developer plugin (<https://addons.mozilla.org/en-US/firefox/addon/60>) ranks among the most valuable utilities in any Web developer's toolbox. Among many other features, you can use it to quickly gauge how much space is available for publishing advertisements by clicking on the **Miscellaneous** option and selecting **Display Ruler**. Your mouse pointer will turn into a cross-hair, which you can drag over a specific area to determine the exact pixel space available for the advertisement.



Figure 13-10. Sample image-based 250x250 pixel Ad unit



Figure 13-11. Sample 160x90 pixel Link unit

After determining the space in which you'd like to place the ad, choose the most size-appropriate format by selecting it from the drop-down list, of which numerous sizes are available (see Figure 13-12). If you'd like to customize the ad display to more closely correspond to your website's theme, use the Colors section to modify the ad's border, title, background, text, and URL colors.

Choose Ad Format and Colors

You can customize your ads to fit in with your pages. Use the options below to specify ad size, style, and more.

Format

Ad units come in a variety of sizes - view all the options on our [Ad Formats page](#).

Colors

Choose from one of our pre-designed color palettes, or create your own palette. [Tips](#)

* Some options apply to text ads only.

Sample

Linked Title
Advertiser's ad text here
www.advertiser-url.com

Ads by Google

Figure 13-12. Choosing an ad format and adjusting the colors

Once you've created the desired ad format, you'll be provided with a text box containing the code you'll need to integrate into your website. For example, here's sample code for a text-based 782x90 pixel (skyscraper) advertisement:

```
<script type="text/javascript"><!--
google_ad_client = "pub-1234567890123456";
/* 728x90, created 2/24/09 */
google_ad_slot = "6446198789";
google_ad_width = 728;
google_ad_height = 90;
//-->
</script>
<script type="text/javascript"
src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
</script>
```

Just paste this into the desired location within your website source code, save the changes and post the updated page to your server. You'll immediately begin serving ads with the next page reload!

Conclusion

In the end, you can write code all day, but if you're not also actively thinking about key business operations such as traffic analysis, advertising, and revenue generation, chances are your project is going to slowly die on the vine. Thankfully, the Google Analytics, AdWords, and AdSense triumvirate go a long way towards remedying these needs with amazingly little overhead.

In the next and final chapter, I'll introduce you to PayPal, another revenue-related solution which can be integrated into your website in mere minutes. Using this fantastic service, you'll be able to accept payments in return for products sold through your website!

CHAPTER 14

Accepting Online Payments with PayPal

By now the project has evolved into a mature website offering with all of the bells and whistles the typical user has come to expect. And thanks to some positive buzz generated by the media an increasingly successful Google AdWords initiative, the traffic is really starting to roll in. With the website doing so well, you've started considering expanding the growing empire into other areas, most notably e-commerce. Having long had an interest in history and writing, you've decided to publish a historical guide to the battles fought throughout the video game, "Call of Duty: World at War" (<http://www.callofduty.com/>). You've already written the book and printed an initial 100 copies using one of the online self-publishing services. All that's left to do is figure out how to sell the book through your website!

In this fourteenth chapter of the book, I'll show you how to accept payments using PayPal, an electronic payment processing service capable of managing the global transfer of funds originating from credit cards and bank accounts. By acting as a middleman, PayPal both eliminates potential distrust from new users that you might mishandle credit card information, while removing the need for you to implement a full-blown e-commerce solution.

VIDEO. An Introduction to E-Commerce

Companies around the globe consider e-commerce to be the golden goose capable of ensuring perpetual profitability. But there are many moving parts which need to be assembled in order to launch even a simple e-commerce website. This video introduces you to these parts, going into detail regarding their respective costs, and offers insight into where to purchase for instance the cheapest SSL certificates. We'll also review several of the most popular e-commerce solutions, discussing the pros and cons of each. Watch the video at <http://www.easypHPwebsites.com/zfw/videos/>.

Chapter Steps

The goals of this chapter are accomplished in two steps:

- **Step #1. Integrating PayPal Website Payments Standard:** If you're interested in selling just one item, PayPal Checkout offers a particularly straightforward solution. In this first step I'll show you how to start selling in just minutes using this solution.
- **Step #2. Exploring Third-Party E-Commerce Solutions:** Should you desire to implement a full-blown e-commerce solution including features such as electronic downloads, product categories, wish lists, bundled and product offers, I recommend seeking out a full featured third-party e-commerce solution rather than attempting to build your own. In this step we'll survey some of the most popular solutions, namely Magento, Ubercart, and Shopify.

Step #1. Integrating PayPal Website Payments Standard

Mention the term "online store" to executives of organizations large and small, and chances are they'll

be unable to contain their excitement. After all, the prospect of opening up the company's product and services catalog to a worldwide audience is undeniably appealing. But many newcomers to the world of e-commerce fail to comprehend the significant overhead an organization will incur in the administration of an online store. This often leaves resource-strapped businesses burdened with the additional costs of maintaining a complex online effort which had a short time ago sounded so appealing. Indeed, countless businesses could save considerable time and effort while still benefiting from increased revenues through online sales by employing the KISS principle to its fullest extent when considering their online e-commerce strategy.

For example, if you're interested in selling solely one or a few products, there's really no reason whatsoever to invest in a full-blown e-commerce solution. Instead, why not turn to one of the many third-party solutions capable of managing the process for you? E-junkie (<http://www.e-junkie.com/>) offers a tremendously easy way to integrate a shopping cart (see Figure 14-1) and sell both digital and physical goods through your website, with no programming required. E-junkie supports all of the most popular payment solutions, among them PayPal, Google Checkout, 2CheckOut, and Authorize. Net. All of these services are offered for a fee (and no transaction costs!) starting at just \$5 per month.



Figure 14-1. E-junkie's cart interface can be integrated into your website (source: e-junkie.com)

But what if you don't even need a shopping cart? What if you simply need a way to securely accept and be notified of payments? Once notified of the payment, you'll package the product and ship it to the purchaser's provided address. There's perhaps no easier way to do this than with PayPal.

By taking advantage of their *Website Payments Standard* program, payments will be submitted directly to your PayPal account, which you can subsequently transfer to your bank account as necessary. In exchange for providing this service, PayPal will collect a \$0.30 + 1.9% to 2.9% fee per transaction (the percentage rate varies according to the total monthly income). To begin accepting payments, all you have to do is sign up for a PayPal account, use the PayPal button creation wizard (Figure 14-2) to create the button which the user will click to initiate the transaction, and embed the generated HTML into your website!

▼ Step 1: Choose button type and enter payment details

Accept payments for

Note: [Go to My saved buttons](#) to create a new button similar to an existing one.

Do you want your customers to buy multiple products before they check out?

Yes; create an "Add to Cart" button. [Learn more](#)

No; create a "Buy Now" button. [Learn more](#)

Item name Item ID (optional) [What's this?](#)

Price Currency [Need multiple prices?](#)

| | |
|---|---------------------|
| Customize button | Buyer's View |
| <input type="checkbox"/> Add drop-down menu with price/option Example <input type="checkbox"/> Add drop-down menu without prices Example <input type="checkbox"/> Add text field Example ▶ Customize appearance | |

Shipping

Use specific amount: USD

Tax

Use tax rate %

Merchant ID for purchase transactions: [What's this?](#)

Secure merchant account ID [Why is this secure?](#)

Plain text e-mail address:

▶ Step 2: Track inventory, profit & loss (optional)
▶ Step 3: Customize advanced features (optional)

Figure 14-2. Creating a custom PayPal button with the wizard

Take note of some of the interesting options within the wizard. If you're selling multiple items, you have the option of creating an "Add to Cart" button, which will allow the customer to aggregate items together within a single order before checking out. If your product comes in several versions, such as multiple sizes, you can include a drop-down menu with variant descriptions and differing prices. You can also optionally track inventory, customize the checkout pages, and specify a post-checkout URL which the customer will be sent to following the purchase. Once submitting the form, you'll be greeted with the interface shown in Figure 14-3.

You are viewing your button code

1. Click **Select Code** to select all the button code.
2. **Copy** the code (CTRL+C for Windows, CMD+C for Mac -- or right-click and choose Copy).
3. **Paste** the code into your own code (CTRL+V for Windows, CMD+V for Mac -- or right-click and choose Paste).

Tutorial

The screenshot shows the PayPal button creation interface. At the top, there are tabs for "Website" and "Email". Below them, a large text area contains the HTML code for the button. To the right of the code is a preview window titled "Buyer's View" showing a "Buy Now" button with payment method icons (VISA, MasterCard, American Express). At the bottom left of the code area are two buttons: "Select Code" and "Go back to edit this button".

Create more buttons



Figure 14-3. Copy and paste the button code into your website

All you need to do is copy and paste the button code into an appropriate location within your website (presumably next to the corresponding product's description). When the user clicks on the button, he'll be transported to the PayPal website, and prompted to either login to their PayPal account or paying using a credit card. The latter option thereby eliminates any requirement for the user to be bothered with registering for a PayPal account should he not already have one. Presuming the customer does have a PayPal account, he'll complete the transaction by clicking on the Pay Now button shown in Figure 14-4.

Review your payment

 Secure Payments

If the information below is correct, click **Pay Now** to complete your payment.

[View PayPal policies and your payment source rights.](#)

| Description | Unit Price | Quantity | Amount |
|--|------------|----------|---------|
| Beginning PHP and MySQL, Third Edition | \$26.99 | 1 | \$26.99 |

Total: \$26.99 USD

[Enter gift certificate, reward, or discount](#)

Pay Now

Payment Method:

- Credit Card Visa XXXX-XXXX-XXXX-XXXX
 Link and use your bank account for this purchase

This credit card transaction will appear on your bill as "WJ GILMORE".

[Change](#)

Ship to:



United States
[Change](#)

Contact Information:

@hotmail.com

Pay Now

Figure 14-4. Completing a transaction initiated with PayPal Website Payments Standard

If you're looking for a quick-and-easy way to begin accepting payments, I'd imagine this section made it abundantly clear PayPal's Website Payments Standard service is a prime candidate for doing so. With the convenience however does come some potential drawbacks, perhaps most notably it requires the customer to leave your website to complete the transaction. If you're looking for a tightly integrated and branded solution, check out PayPal's Website Payments Pro service, which gives you total control over the entire purchasing process, while still giving you the advantage of using PayPal for the payment processing. Of course, this does come with quite a bit of additional overhead, both in terms of development and other costs such as obtaining an SSL certificate in order to ensure a secure purchasing environment.

Step #2. Exploring Third-Party E-Commerce Solutions

The e-commerce implementations discussed so far in this chapter are fantastic if your ambitions are relatively modest, largely limited to accepting online payments for a select set of physical products. However, if you have more lofty aspirations for selling products online, you may want to look towards one of the many available turnkey e-commerce solutions. These solutions offer many of the features shoppers have come to expect, such as customer reviews, order history, product tagging, cross-promotions, and coupon codes. Likewise, store owners can take advantage of features such as template management, sophisticated reporting, support for multiple payment backends, shipping and

In this step I'll introduce you to three particularly promising e-commerce solutions which have gained quite a following over the past year or so. Offering many, if not all of the aforementioned features, these solutions can give you a major boost towards your goal of competing with the e-commerce goliaths in surprisingly little time.

Magento

Magento (<http://www.magentocommerce.com/>) is a recent entry to the crowded e-commerce solutions market, with version 1.0 released on March 31, 2008. Despite being a relative newcomer, Magento has already experienced a meteoric rise in popularity, boasting almost 74,000 registered users and having won the award of best new open source project at the 2008 Sourceforge.net Community Choice Awards.

Magento's recipe for success can be attributed to several key ingredients:

- **Open source:** Magento is released under the Open Software License, meaning you can download, customize, deploy, and even distribute the software provided you adhere to the terms set forth in this license.
- **Cutting-edge:** In addition to all of the usual features one would expect from an e-commerce platform, you'll also find RSS feeds for customer wish lists, advertising new products and special offers, the ability to retrieve real-time shipping quotes from the United States Postal Service and others, and extensive integration with all of the mainstream payment gateways. Recent and forthcoming enhancements include a theme optimized for the iPhone, integration support with third-party shipping and accounting solutions, and a gift registry.
- **Community-oriented:** Like any successful open source project, Magento boasts a vibrant user community who has thus far contributed almost 500 extensions via Magento's open source and commercial extension marketplace known as Magento Connect (<http://www.magentocommerce.com/magento-connect>).

Although not even a year old, Magento is quickly gathering an impressive list of users. Check out the following stores for some inspiration regarding what's possible with this impressive platform.

- Poster.com (<http://www.poster.com/>): Magento's ability to manage an enormous catalog is particularly apparent here, as Poster.com is the world's largest vendor of posters and prints, boasting more than 200,000 products.
- TCHO (<http://www.tcho.com/store/>): A chocolate manufacturer founded by Wired Magazine co-founder Louis Rossetto, TCHO uses Magento to sell hot chocolate and various chocolate tasting packs.
- Time Out (<http://www.timeout.com/shop/>): Travel publisher Time Out vends its array of travel guides and magazines via their Magento-powered online shop.

Ubercart

Ubercart (<http://www.ubercart.org/>) is an open source e-commerce extension for the popular PHP-powered content management system Drupal (<http://www.drupal.org/>). Like Magento, it's a full-featured solution, supporting multiple payment systems, downloadable products, subscription-based revenue models, shipping calculators, and a mature Web-based administration console (see Figure 14-5).

Because Ubercart is Drupal-dependent, users unfamiliar will have the additional chore of learning how to install, configure and manage the CMS before even beginning to explore Ubercart. This may however be well worth your time, as Drupal is a very impressive product, with adept users taking advantage of it to manage even the largest and most complex of websites. Further, in addition to Ubercart, you'll find thousands of other modules capable of fulfilling feature requests such as user reviews, search, event calendars, blogs, user authentication, and much more.

The screenshot shows the Ubercart administration interface. At the top left, it says "Home » Administer" and "Store administration". Below this is a menu bar with six items: "Orders", "Customers", "Products", "Reports", "Configuration", and "Help". Each item has a small icon and a "- Show links -" link. Underneath the menu is a section titled "Status messages:" which contains two entries:

| Title | Description |
|--------------------|---|
| Catalog vocabulary | Vocabulary Product Catalog has been identified as the Übercart catalog. |
| Unlisted Products | All products are currently listed in the catalog. |

Figure 14-5. Ubercart's administration interface

Ubercart has been embraced by thousands of organizations around the globe. While researching the material for this section I encountered three particularly interesting implementations:

- Cornerstone Cookie Company (<http://www.cornerstonedecookiegifts.com/>): The Cornerstone Cookie Company's beautiful Ubercart-driven website offers dozens of cookie-related products for corporate occasions and holidays.
- Nowt Records (<http://www.nowtrecords.com/>): Non-profit indie musician promoter Nowt records relies on Ubercart to sell MP3s, CDs, and DVDs produced by its group of eclectic musicians.
- Park Avenue Coffee (<http://www.parkavenuecoffee.com/>): St. Louis-based Park Avenue Coffee uses Ubercart to sell a selection of locally-roasted coffees, coffee-related equipment and accessories, and a particularly delicious-looking pastry they call Gooey Butter Cake.

Shopify

Shopify (<http://www.shopify.com/>) is a particularly interesting e-commerce solution in that it's what's known as a *hosted solution*. Offering many of the features you'd find within the installable solutions, but lacking the administrative overhead and occasional configuration frustrations encountered when managing the software locally.

Unlike many other hosted solutions, Shopify transparently integrates into your existing website, done by pointing a subdomain such as <http://shop.gamenomad.com/> to Shopify's servers. You'll then manage your store's design, products, and other details using a Web-based administration console.

| Product | Inventory |
|--|---|
| Blue Tee Basic Tshirt American Apparel | Extra Small: 3, Small: 12, Medium: 18, Large: 2 |
| Green Tee Basic Tshirt American Apparel | Extra Small: 3, Small: 12, Medium: 18, Large: 2 |
| Khaki Fleece Zipper Zipper American Apparel | Extra Small: ∞, Small: ∞, Medium: ∞, Large: ∞ |
| Red Tank Top Tank Top American Apparel | Extra Small: 5, Small: 9, Medium: 12, Large: 0 |
| Striped Tank Top Blue/Orange Tank Top American Apparel | Small: 12, Medium: 10, Large: 3 |
| Striped Tank Top Red/White Tank Top American Apparel | Small: 12, Medium: 10, Large: 3 |
| White Tank Top Tank Top American Apparel | Extra Small: 2, Small: 5, Medium: 0, Large: 2 |

Figure 14-6. Managing products with Shopify

Shopify currently offers several tiered plans, ranging from \$24/month and a 2.0% transaction fee to \$299/month and a 0.5% transaction fee. You can also change plans as necessary, starting small and upgrading as your needs grow.

Like Magento and Ubercart, Shopify boasts a growing list of interesting clients, including:

- Tesla Motors (<http://shop.teslamotors.com/>): The company behind the electric Tesla Roadster uses Shopify to sell shirts, hats, and various travel accessories emblazoned with the Tesla Motors logo.
- The Candle Company (<http://www.thecandlecompanyonline.com/>): This New Jersey-based company uses their online store to sell more than 200 candle-related products.
- The Indianapolis Star (<http://www.indystarstore.com/>): Indianapolis' largest newspaper uses Shopify to sell a variety of products pertinent to the state of Indiana. Within hours of the Colts' 2007 Superbowl win, the store successfully handled hundreds of thousands of visitors, and processed over \$100,000 in sales (*Calgary Herald*: <http://tinyurl.com/df5a2f>).

If you're looking for a high-powered e-commerce solution but don't want to deal with the maintenance overhead, Shopify may well be worth some additional investigation.

Conclusion

The Web turned the world of commerce on its head, making it possible for even the smallest of companies to open their doors to an audience of billions. Whether you'd like to build a custom PayPal-backed solution or would rather rely on a turnkey solution such as Magento, Ubercart or Shopify, your options abound for building a killer e-commerce website.

CHAPTER 15

Introducing Zend_Tool

The programming community seems to be equally split between those wed to the command-line and others who swear by a point-and-click interface. Yet with so many great tools available in both camps these days, why would you possibly limit your solutions? For example, I've been a longtime user of the PHP Developer Tools IDE, but wouldn't trade the MySQL command-line client for any of the many graphical solutions.

No matter which camp you might fall into, chances are you'll want to take advantage of the newly available Zend_Tool component. Available in beta version as part of the Zend Framework 1.7 release, and an official part of the framework as of 1.8, Zend_Tool is a great utility which removes much of the tedium involved when creating your Zend Framework-powered website. Providing a command-line interface for creating projects, controllers, views, and the other project-related resources, Zend_Tool will greatly reduce the time required when otherwise creating these files and directory structure manually.

Of course, Zend_Tool will also greatly reduce the pain endured by many newcomers to the Zend Framework, including most notably the confusion surrounding where to place the various files such as the bootstrapper, front controller, and configuration file, and how to organize the controllers, models, and views.

NOTE. Zend_Tool is actually much more than a command-line utility for creating project resources; you can extend the component to automate the tedium of other project-related tasks. It will be really interesting to watch where the community takes this tool in the coming months.

Chapter Steps

The goals of this chapter are accomplished in just two steps:

- **Step #1. Configuring Zend_Tool:** Before you can use Zend_Tool you'll need to carry out a few configuration tasks. In this step I'll show you what to do.
- **Step #2. Creating a Project with Zend_Tool:** This step will guide you through the creation of a typical Zend Framework project using Zend_Tool.

Step #1. Configuring Zend_Tool

Although a beta version of Zend_Tool has been included in the Zend Framework since version 1.7, as Zend Framework 1.8 was recently released I'll assume you've already upgraded or intend to do so in the near future (the assumption is important because the configuration process is different for version 1.7). Within the Zend 1.8 download you'll find a directory named `bin`, containing three files, `zf.bat`, `zf.php`, and `zf.sh`.

On Windows, copy the `zf.bat` and `zf.php` files into the same directory where your `php.exe` file is located. Next, make sure the directory where `php.exe` is located has been added to your system path. This will make it possible for you to execute the `zf` command (the command Zend_Tool uses to carry out tasks) from any location within your file system. Finally, place the Zend Framework library directory (this directory contains all of the files necessary for the Zend Framework to operate) within PHP's `include_path`.

On Linux the process is much the same; just copy `zf.sh` and `zf.php` to the same location where your PHP binary resides. To determine the location you can typically execute the following command:

```
%>which php  
/usr/bin/php
```

For both operating systems, instead of adding the Zend Framework library directory to your `include_path`, you have the option of prepending the directory path to your `include_path` directory using the `ZEND_TOOL_INCLUDE_PATH-prepend` system variable, or overwriting the `include_path` directory altogether when Zend_Tool is being used by assigning the `library` directory path to `ZEND_TOOL_INCLUDE_PATH`.

Finally, confirm Zend_Tool is working properly by executing the following command from your command-line:

```
%>zf show version  
Zend Framework Version: 1.8.1
```

If you do not see your framework version number, and instead receive an error, it's almost certainly because the wrong path was used within the system path variable or when defining the Zend library directory's location within the `include_path` directive. So be sure to double-check those settings if you encounter a problem. Otherwise, presuming your framework version number has indeed been output, move on to Step #2 to learn how to use Zend_Tool!

TIP. Windows' built-in command prompt (accessible by entering `cmd` from the Run... console) leaves much to be desired. As you'll presumably be spending quite a bit of time using Zend_Tool, using this prompt will quickly become tiresome. Consider installing Console2 (<http://sourceforge.net/projects/console/>), a fantastic command prompt replacement which lets you run multiple prompts using a tabbed interface, easily change the font size and color, and easily resize the window.

Step #2. Creating a Project with Zend_Tool

Once configured, you can begin using Zend_Tool to create and maintain your projects. In this section, I'll guide you through the creation and management of a simple Zend Framework-powered website named `corporate` using this tool.

Creating a New Project

To create a project named `corporate`, using the `create project` command like so:

```
%>zf create project corporate
```

After a moment you'll be notified of the successful project creation with a message similar to this:

```
Creating project at C:/apache/htdocs/corporate
```

Go ahead and enter this directory and you'll see four directories (`application`, `library`, `public`, `tests`) and a file named `.zfproject.xml` have been created. Believe it or not, this is all that's required to setup the new project. Set Apache's document root to point to the application's public directory (in the case of this example the path would be `C:/apache/htdocs/corporate/public`), restart Apache, and navigate to `http://localhost`. You'll see the page shown in Figure 15-1.

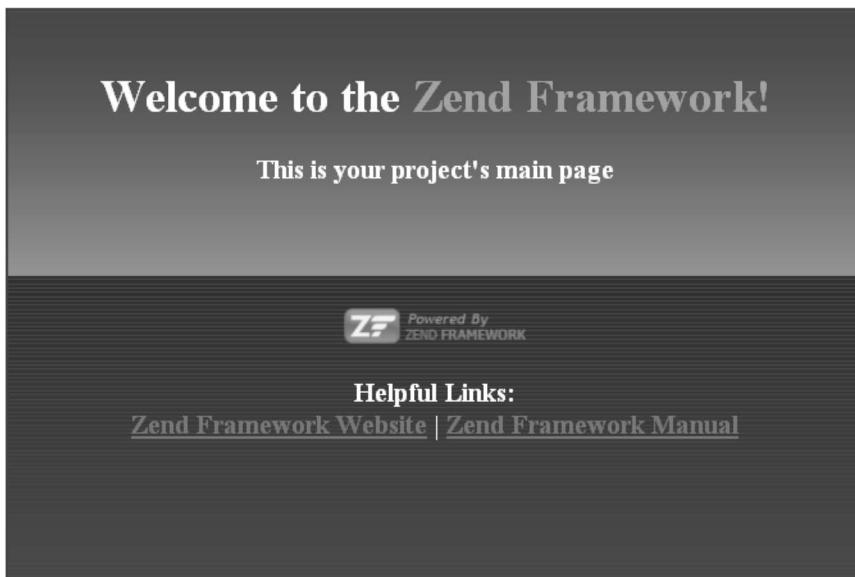


Figure 15-1. A placeholder index controller is created which each new project

I'd imagine you're familiar with the purpose of the four directories automatically created as a result of executing the `create project` command. However the `.zfproject.xml` file is new. This file is used by Zend_Tool to keep track of the project's structure, so be sure not to remove or modify it unless you know what you're doing.

If you've created other Zend Framework-driven websites in the past, I also strongly suggest taking a moment to navigate around the newly created project structure, because you'll notice several departures from the previously used organizational structure. For instance, you'll notice that the configuration file is now located in its own directory (named `configs`). Also, notice that while a `library` directory has been created, ostensibly to host the Zend Framework-specific files, this directory is actually empty because the project will refer to the `library` directory you identified when configuring Zend_Tool. Keep this in mind when migrating your project to the production server as obviously

it won't work without these files. As before, you can change the organization of your project to your liking, so if you don't like the new structure consult the Zend Framework documentation for some direction regarding how to override the default settings.

Creating a Controller

When the project is created, Zend_Tool will automatically create the `Index` and `Error` controllers, so you can go about modifying the `Index` controller right away. However, you'll also likely want to create a few other controllers. For instance, we might create a controller named `About` which will visitors a bit more about the corporation. To do this, use the `create controller` command:

```
%>zf create controller about
```

Executing this command will set quite a bit of action into motion, resulting in the creation of the `About` controller complete with `init()` method and `index` action skeletons, a corresponding `index` view and an `About` controller test file. The project profile (`.zfproject.xml`) is also updated to reflect the latest changes to the project.

```
Creating a controller at C:\apache\htdocs\ corporate/application/controllers  
AboutController.php  
Creating an index action method in controller about  
Creating a view script for the index action method at C:\apache\htdocs\zf18test  
corporate/application/views/scripts/about/index.phtml  
Creating a controller test file at C:\apache\htdocs\ corporate/tests/application  
controllers/AboutControllerTest.php  
Updating project profile 'C:\apache\htdocs\corporate/.zfproject.xml'
```

If you want Zend_Tool to forego creation of an `index` view, pass a second parameter of 0 to the `create controller`, like so:

```
%>zf create controller about 0
```

Creating Actions

You can add an action to an existing controller using the `create action` command. For instance, to add an action named `contact` to the `About` controller, use the following command:

```
%>zf create action contact about
```

This will also create the corresponding `contact.phtml` view. To forego creation of the view, pass a third parameter of 0 like so:

```
%>zf create action contact about 0
```

Creating Views

You can use the `create view` command to create new views. At the time of writing, the Zend_Tool help interface seems to incorrectly indicate how this command should run, and in fact the command

does not run at all without fixing a bug located in the Zend_Tool code. However, I suspect the issue will be fixed within days after the release of this chapter, meaning you'll be able to create the contact view within the About controller like this:

```
%>zf create view contact about
```

Keep in mind this command only creates the view. If you need to create the action, use the `create action` command.

Creating a Module

Chances are your website will be updated using some sort of administration interface. When using the Zend Framework you'll typically manage these administration tools in a separate *module*. You can think of a module as a website within a website, containing its own set of controllers, actions, models, and views, but able to take advantage of the other parts of the website, including the configuration file. To create a module named `admin`, execute the `create module` command, like so:

```
%>zf create module admin
```

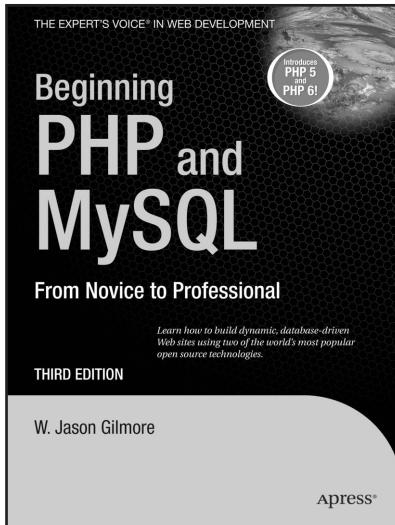
Once created, some output will ensue explaining what has taken place. As you can see from the following example output, the `admin` module has its own controllers, models, and other necessary resources:

```
Creating the following module and artifacts:  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\controllers  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\models  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\views  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\views\scripts  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\views\helpers  
C:\apache\htdocs\zf18test\corporate\application\modules\admin\views\filters  
Updating project profile 'C:\apache\htdocs\zf18test\corporate/.zfproject.xml'
```

Conclusion

The Zend_Tool component represents a significant step forward not only in terms of helping experienced Zend Framework developers quickly lay the groundwork for new websites, but also in terms of helping newcomers quickly surpass what have historically been the most significant roadblocks. Even more exciting is Zend_Tool's extendability, as we'll surely see even more interesting features stemming from this component in the coming months.

1,044 PAGES of PHP and MySQL Instruction



Beginning PHP and MySQL, Third Edition is the definitive guide to two of the world's most prominent open source technologies: the PHP scripting language and the MySQL database server. Packed with over 500 downloadable examples, this book will serve as a handy reference for years to come!

Buy **Beginning PHP and MySQL, Third Edition** direct from the author for only \$28. (shipping and handling included!) from <http://www.easypHPwebsites.com/>. When checking out be sure to use the code **easypHPPromo**.

WJ Gilmore, LLC

Due to shipping costs, offer valid only to U.S. residents.

