Wrox Programmer to Programmer™

PHP and MySQL®

Create - Modify - Reuse

Timothy Boronczyk with Martin E. Psinas

WROX

Updates, source code, and Wrox technical support at **www.wrox.com**

PHP and MySQL®
Create-Modify-Reuse

Tim Boronczyk

with

Martin E. Psinas

WILEY

Wiley Publishing, Inc.

PHP and MySQL®
Create-Modify-Reuse

Introduction xi

Chapter 1: User Registration 1
Chapter 2: Community Forum 31
Chapter 3: Mailing List 63
Chapter 4: Search Engine 87
Chapter 5: Personal Calendar 113
Chapter 6: Ajax File Manager 137
Chapter 7: Online Photo Album 177
Chapter 8: Shopping Cart 195
Chapter 9: Web Site Statistics 239
Chapter 10: News/Blog System 265
Chapter 11: Shell Scripts 291
Chapter 12: Security and Logging 315

Index 333

PHP and MySQL®
Create-Modify-Reuse

Tim Boronczyk

with

Martin E. Psinas

WILEY

Wiley Publishing, Inc.

PHP and MySQL® : Create-Modify-Reuse

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-0-470-19242-9

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Boronczyk, Tim, 1979-
PHP and MySQL : create-modify-reuse / Tim Boronczyk with Martin E. Psinas.
p. cm.
Includes index.
ISBN 978-0-470-19242-9 (paper/website)
1. MySQL (Electronic resource) 2. PHP (Computer program language) 3. Web sites—Design.
I. Psinas, Martin E. II. Title.
QA76.73.P224B64 2008
006.7'6—dc22

2008011996

About the Author

Timothy Boronczyk is a native of Syracuse, NY, where he works as a freelance developer, programmer and technical editor. He has been involved in web design since 1998 and over the years has written several articles and tutorials on PHP programming. Timothy holds a degree in software application programming and recently started his first business venture, Salt City Tech (`www.saltcitytech.com`). In his spare time, he enjoys photography, hanging out with friends, and sleeping with his feet hanging off the end of his bed. He's easily distracted by shiny objects.

About the Contributor

Martin E. Psinas is a recognized security expert and valued member of the open-source community. He has been contracted as a technical editor, code auditor, and is a published author with Pearson Education as well as the #1 PHP magazine, *PHP | Architect*. In his free time, he maintains his personal web site and is a volunteer administrator/contributor at codewalkers.com — a resource for PHP & MySQL developers. Martin interacts frequently with the leaders of the PHP project as well as PHP User's Groups.

Credits

Acquisitions Editor
Jenny Watson

Development Editor
Ed Connor

Technical Editor
Graham Christensen

Production Editor
Daniel Scribner

Copy Editor
Michael Koch

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Project Coordinator, Cover
Lynsey Stanford

Proofreader
Corina Copp

Indexer
Ron Strauss

Contents

Introduction xi

Chapter 1: User Registration 1

Plan the Directory Layout 1
Planning the Database 2
Writing Shared Code 3
User Class 5
CAPTCHA 9
Templates 11
Registering a New User 12
E-mailing a Validation Link 17
Logging In and Out 21
Changing Information 25
Forgotten Passwords 28
Summary 30

Chapter 2: Community Forum 31

Design of the Forum 31
Designing the Database 32
Working with Permissions and Bitwise Operators 33
Updating the User Class 35
Code and Code Explanation 40
Adding Forums 41
Adding Posts 43
Displaying Forums and Posts 47
Pagination 55
Avatars 56
BBCode 59
Summary 62

Chapter 3: Mailing List 63

Design of the Mailing List 63
Choosing POP3 64

Designing the Database **65**
Code and Code Explanation **66**
The POP3 Client 66
The Configuration File 73
Account Management 73
Processing Messages 79
Processing the Digest 83
Setting Up the Mailing List **83**
Summary **86**

Chapter 4: Search Engine **87**

Designing the Search Engine **87**
Problems with Full-Text Search **88**
Designing the Database **89**
Code and Code Explanation **91**
Administrative Interface 91
Crawler/Indexer 98
Front End 104
Summary **110**

Chapter 5: Personal Calendar **113**

Designing the Application **113**
Designing the Database **114**
Code and Code Explanation **115**
Creating a Month-View Calendar 115
Creating a Day-View Calendar 120
Adding and Showing Events 121
Sending Reminders 129
Exporting the Calendar 130
Summary **135**

Chapter 6: Ajax File Manager **137**

Design of the Ajax File Manager **137**
JavaScript and Ajax **138**
The XMLHttpRequest Object 139
Code and Code Explanation **142**
Main Interface 143
Client-Side Functionality 147
Server-Side Functionality 160
Summary **175**

Chapter 7: Online Photo Album **177**

Design of the Online Photo Album **177**
Code and Code Explanation **178**
Views 178
Helper Files 188
QuickTime Thumbnails **190**
Thumbnail Caching **192**
Summary **193**

Chapter 8: Shopping Cart **195**

Designing the Shopping Cart **195**
Designing the Database **196**
Code and Code Explanation **197**
The ShoppingCart Class 197
Working with the Shopping Cart 201
Building the Storefront 209
Adding Inventory 217
Summary **238**

Chapter 9: Web Site Statistics **239**

Determining What to Collect **239**
Designing the Database **241**
Obtaining Data **242**
Code and Code Explanation **244**
Pie Chart 244
Bar Chart 248
The Report 253
Summary **264**

Chapter 10: News/Blog System **265**

Tables **265**
Adding Posts **267**
Generating the RSS **278**
Displaying Posts **282**
Adding Comments **285**
Summary **289**

Chapter 11: Shell Scripts **291**

Designing the Script **292**
General Shell Scripting Advice **293**
Code and Code Explanation **294**
The CommandLine Class 294
startproject 303
The Skeleton **313**
Summary **314**

Chapter 12: Security and Logging **315**

Cross-Site Scripting **315**
Path Traversal **318**
Injection **320**
SQL Injection 320
Command Injection 324
Weak Authentication **325**
Logging **327**
Preventing Accidental Deletes **330**
Summary **332**

Index **333**

Introduction

I'm especially amazed at how the Internet has grown and evolved over the past decade or so. It has grown from a collection of static text documents connected by a few hyperlinks to a platform for delivering rich, distributed applications. And when it comes time to develop these web-based applications, many programmers are choosing PHP and MySQL.

In this book, I present basic code for 12 PHP-powered projects that you can use and extend however you wish. I have tried to write them so the code can be easily reused in future applications, but in some instances the entire application can be reused as well!

I've enjoyed the opportunity to write and share with you this information and I hope you have just as much fun reading it and learning from it. More importantly, I hope you find good, practical uses for the projects found within this book.

Who This Book Is For

I present basic yet functional projects for you to implement and extend in any way you see fit. That very fact assumes you know the fundamentals of programming in PHP and general web development. This book is not a textbook. Still, you do not need to be an advanced PHP programmer to gain much by reading it. New programmers should find this book helpful as it will give them guidance in how to program different applications. The 12 projects may even serve to ignite their curiosity and spur them to write 12 more projects of their own. Intermediate and more experienced programmers will find this book helpful because they are able to take the projects I present, modify them and apply them to their real-world needs.

Some projects build upon previous projects, so while you don't have to read the book from cover to cover, I do suggest reading all relevant chapters (or at least the pertinent sections) regardless of your skill level. For example, in Chapter 7, I present an online photo album, but pictures are uploaded using the AJAX file manager presented in Chapter 6. Both projects are laid out in the manner presented in Chapter 1.

What This Book Covers

The code in this book was written for MySQL 5.0 Community Server and PHP version 5.2.5, so essentially I am covering those releases or greater. Additional modification may be necessary if you plan on using earlier releases.

How This Book Is Structured

Each chapter is organized so following projects can build upon earlier projects. Here's a brief rundown of what you can look forward to in the following chapters:

Chapter 1: User Registration
Create a basic user registration system
Reusable components: configuration/include files, 401.php, User class

Chapter 2: Community Forum
Expand on user registration system to create a community forum with user privileges and threaded posts
Reusable components: `JpegThumbnail` class, `BBCode` class

Chapter 3: Mailing List
Create a mailing list with control address and digest mailings
Reusable components: `POP3Client` class

Chapter 4: Search Engine
Build a custom search engine for your own site
Reusable components: entire application

Chapter 5: Personal Calendar
Write a personal calendar utility to keep yourself organized
Reusable components: entire application

Chapter 6: AJAX File Manager
Create an AJAX-ified file upload and directory viewer
Reusable components: entire application (this project introduces AJAX which will be used in subsequent projects)

Chapter 7: Online Photo Album
Create a file-based image gallery with automatically generated thumbnails that supports JPEG and QuickTime formats.
Reusable components: `MovThumbnail` class

Chapter 8: Shopping Cart
Write a categorized shopping cart
Reusable components: `ShoppingCart` class

Chapter 9: Web Site Statistics
Log site traffic and collect information about site visitors to make better business decisions
Reusable components: `PieChart` class, `BarChart` class

Chapter 10: News/Blog system
Build a news or blog system with comments and RSS feed
Reusable components: entire application (project also introduces reusable components such as YUI calendar and TinyMCE rich text control)

Chapter 11: Shell Scripts
Write and run management scripts
Reusable components: `CommandLine` class, `recurs_copy()` function

Chapter 12: Security and Logging
Learn about SQL injection, path traversal, weak authentication, and XSS and how to avoid them
Reusable components: `write_log()` function, `view_log.php`, record delete script

What You Need to Use This Book

Since you'll be writing PHP code, you'll need an editor to do so. Whichever you choose to use is a matter of preference. Additionally, you will also need a server running PHP and MySQL to host your applications and a web browser to access them. What you use is a matter of choice. I've provided instructions for setting up applications on both Unix and Windows platforms when necessary, for example the Mailing List application in Chapter 3, which runs as a scheduled job.

Personally, I used vi to write code, hosted the projects on a server running Slackware Linux and accessed them from a Windows XP computer using Firefox.

Some of the projects make use of special extensions to PHP, although I have tried to keep this to a minimum. For example, the Search Engine application presented in Chapter 4 uses the pspell extension. If additional functionality was needed which could only be provided by an extension, I avoided third-party extensions so that if you want to install a particular extension you only need to look as far as the official documentation at `www.php.net`. The relevant extensions are mentioned in the appropriate chapters.

Conventions

To help you get the most from the text and keep track of what's happening, I've used a number of conventions throughout the book.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

```
We use a monofont type with no highlighting for most code examples.
We use gray highlighting to emphasize code that's particularly important
in the present context.
```

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `www.wrox.com`. When at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

> *Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-19242-9.*

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata you may save another reader hours of frustration and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At p2p.wrox.com you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

User Registration

Offering account registration and user logins is a great way of giving users a sense of individuality and serving tailored content. Such authentication is often at the very heart of many community-oriented and e-commerce web sites. Because this functionality is so useful, the first application I present is a user registration system.

From a functional perspective, the system will allow users to create accounts. Members must provide an e-mail address that they can use to validate their registration. Users should also be able to update their passwords and e-mail addresses and reset forgotten passwords. This is pretty standard functionality and what the web users of today have come to expect.

From an architectural standpoint, the directory holding your code should be logically organized. For example, support and include files should be kept outside of a publically accessible directory. Also, user records should be stored in a database. Since there are a large number of tools designed to view and work with data stored in relational databases such as MySQL, this affords transparency and flexibility.

Plan the Directory Layout

The first step is to plan the directory structure for the application. I'm going to recommend you create three main folders: One named `public_files` from which all publicly accessible files will be served, another named `lib` to store include files to be shared by any number of other files, and finally a `templates` folder to store presentation files. Although PHP will be able to reference files from anywhere in your setup, the web server should only serve files from the `public_files` folder. Keeping support files outside of the publicly accessible directory increases security.

Inside the `public_files` I also create `css` to store any style sheets, `js` for JavaScript source files and `img` for graphic files. You may want to create other folders to keep yourself organized. One named `sql` to store MySQL files would be a good idea, `doc` for documentation and development notes and `tests` to store smoke test or unit testing files.

Planning the Database

In addition to planning the directory layout, thought needs to be given to the database layout as well. The information you choose to collect from your users will depend on what type of service your site offers. In turn, this affects how your database tables will look. At the very least a unique user ID, username, password hash, and e-mail address should be stored. You will also need a mechanism to track which accounts have been verified or are pending verification.

```
CREATE TABLE WROX_USER (
    USER_ID    INTEGER UNSIGNED  NOT NULL  AUTO_INCREMENT,
    USERNAME   VARCHAR(20)       NOT NULL,
    PASSWORD   CHAR(40)          NOT NULL,
    EMAIL_ADDR VARCHAR(100)      NOT NULL,
    IS_ACTIVE  TINYINT(1)        DEFAULT 0,

    PRIMARY KEY (USER_ID)
)
ENGINE=MyISAM DEFAULT CHARACTER SET latin1
    COLLATE latin1_general_cs AUTO_INCREMENT=0;

CREATE TABLE WROX_PENDING (
    USER_ID       INTEGER UNSIGNED  NOT NULL,
    TOKEN         CHAR(10)          NOT NULL,
    CREATED_DATE  TIMESTAMP         DEFAULT  CURRENT_TIMESTAMP,

    FOREIGN KEY (USER_ID)
        REFERENCES WROX_USER(USER_ID)
)
ENGINE=MyISAM DEFAULT CHARACTER SET latin1
    COLLATE latin1_general_cs;
```

I have allocated 40 characters of storage for the password hash in `WROX_USER` as I will use the `sha1()` function which returns a 40-character hexadecimal string. You should never store the original password in the database — a good security precaution. The idea here is that a hash is generated when the user provides his or her password for the first time. The password given subsequently is hashed using the same function and the result is compared with what's stored to see if they match.

I set the maximum storage length for an e-mail address at 100 characters. Technically the standards set the maximum length for an e-mail address at 320 (64 characters are allowed for the username, one for the `@` symbol and then 255 for the hostname). I don't know anyone that has such a long e-mail address though and I have seen plenty of database schemas that use 100 and work fine.

Some other information you may want to store are first and last name, address, city, state/province, postal code, phone numbers, and the list goes on.

The `WROX_PENDING` table has an automatically initializing timestamp column, which lets you go back to the database and delete pending accounts that haven't been activated after a certain amount of time. The table's columns could be merged with `WROX_USER`, but I chose to separate them since the pending token is only used once. User data is considered more permanent and the `WROX_USER` table isn't cluttered with temporary data.

Writing Shared Code

Code that is shared by multiple files should be set aside in its own file and included using `include` or `require` so it's not duplicated, which makes maintaining the application easier. Where possible, code that might be useful in future applications should be collected separately as functions or classes to be reused. It's a good idea to write code with reusability in mind. `common.php` contains shared code to be included in other scripts in the application to establish a sane baseline environment at runtime. Since it should never be called directly by a user, it should be saved in the `lib` directory.

```
<?php
// set true if production environment else false for development
define ('IS_ENV_PRODUCTION', true);

// configure error reporting options
error_reporting(E_ALL | E_STRICT);
ini_set('display_errors', !IS_ENV_PRODUCTION);
ini_set('error_log', 'log/phperror.txt');

// set time zone to use date/time functions without warnings
date_default_timezone_set('America/New_York');

// compensate for magic quotes if necessary
if (get_magic_quotes_gpc())
{
    function _stripslashes_rcurs($variable, $top = true)
    {
        $clean_data = array();
        foreach ($variable as $key => $value)
        {
            $key = ($top) ? $key : stripslashes($key);
            $clean_data[$key] = (is_array($value)) ?
                stripslashes_rcurs($value, false) : stripslashes($value);
        }
        return $clean_data;
    }
    $_GET = _stripslashes_rcurs($_GET);
    $_POST = _stripslashes_rcurs($_POST);
    // $_REQUEST = _stripslashes_rcurs($_REQUEST);
    // $_COOKIE = _stripslashes_rcurs($_COOKIE);
}
?>
```

You may not always have control over the configuration of your server so it is wise to specify some common directives to make your applications more portable. Setting error reporting options, for example, lets you display errors while in development or redirect them in a production environment so they don't show to the user.

Magic quotes is a configuration option where PHP can automatically escape single quotes, double quotes, and backslashes in incoming data. Although this might seem useful, assuming whether this directive is on or not can lead to problems. It's better to normalize the data first and then escape it with `addslashes()` or `mysql_real_escape_string()` (preferably the latter if it's going to be stored in the

database) when necessary. Compensating for magic quotes ensures data is properly escaped *how* you want and *when* you want despite how PHP is configured, making development easier and less error-prone.

Establishing a connection to a MySQL database is a common activity which makes sense to move out to its own file. `db.php` holds configuration constants and code to establish the connection. Again, as it is meant to be included in other files and not called directly, it should be saved in `lib`.

```
<?php
// database connection and schema constants
define('DB_HOST', 'localhost');
define('DB_USER', 'username');
define('DB_PASSWORD', 'password');
define('DB_SCHEMA', 'WROX_DATABASE');
define('DB_TBL_PREFIX', 'WROX_');

// establish a connection to the database server
if (!$GLOBALS['DB'] = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD))
{
    die('Error: Unable to connect to database server.');
}
if (!mysql_select_db(DB_SCHEMA, $GLOBALS['DB']))
{
    mysql_close($GLOBALS['DB']);
    die('Error: Unable to select database schema.');
}
?>
```

The `DB_HOST`, `DB_USER`, `DB_PASSWORD` and `DB_SCHEMA` constants represent the values needed to establish a successful connection to the database. If the code is put into production in an environment where the database server is not running on the same host as PHP and the web server, you might also want to provide a `DB_PORT` value and adjust the call to `mysql_connect()` appropriately.

The connection handle for the database is then stored in the `$GLOBALS` super global array so it is available in any scope of any file that includes `db.php` (or that is included in the file that has referenced `db.php`).

Prefixing table names helps prevent clashes with other programs' tables that might be stored in the same schema and providing the prefix as a constant makes the code easier to update later if it should change, since the value appears just in one place.

Common functions can also be placed in their own files. I plan to use this `random_text()` function, for example, to generate a CAPTCHA string and validation token so it can be saved in a file named `functions.php`.

```
<?php
// return a string of random text of a desired length
function random_text($count, $rm_similar = false)
{
    // create list of characters
    $chars = array_flip(array_merge(range(0, 9), range('A', 'Z')));
```

```
        // remove similar looking characters that might cause confusion
        if ($rm_similar)
        {
            unset($chars[0], $chars[1], $chars[2], $chars[5], $chars[8],
                $chars['B'], $chars['I'], $chars['O'], $chars['Q'],
                $chars['S'], $chars['U'], $chars['V'], $chars['Z']);
        }

        // generate the string of random text
        for ($i = 0, $text = ''; $i < $count; $i++)
        {
            $text .= array_rand($chars);
        }

        return $text;
    }
    ?>
```

An important rule when programming no matter what language you're using is to never trust user input. People can (and will) provide all sorts of crazy and unexpected input. Sometimes this is accidental, at other times it's malicious. PHP's `filter_input()` and `filter_var()` functions can be used to scrub incoming data, though some people still prefer to write their own routines, as the filter extension may not be available in versions prior to 5.2.0. If you're one of those people, then they can be placed in `functions.php` as well.

User Class

The majority of the code written maintaining a user's account can be encapsulated into one data structure, making it easy to extend or reuse in future applications. This includes the database interaction logic, which will make storing and retrieving information easier. Here's `User.php`:

```
    <?php
    class User
    {
        private $uid;     // user id
        private $fields;  // other record fields

        // initialize a User object
        public function __construct()
        {
            $this->uid = null;
            $this->fields = array('username' => '',
                                  'password' => '',
                                  'emailAddr' => '',
                                  'isActive' => false);
        }

        // override magic method to retrieve properties
        public function __get($field)
```

(continued)