Understanding the Zend Framework, Part 2: Model-View-Controller and adding a database

Building the perfect reader

Skill Level: Intermediate

Nicholas Chase (ibmquestions@nicholaschase.com)
Consultant
Backstop Media

Tracy Peterson (tracy@tracypeterson.com)
Freelance Writer
Freelance Developer

11 Jul 2006

Updated 24 Jan 2011

This "Understanding the Zend Framework" series chronicles the building of an online feed reader, Chomp, while explaining the major aspects of using the open source PHP Zend Framework. Part 1 discusses the goals behind the Zend Framework, including easy-to-use components and an architecture based on the Model-View-Controller (MVC) pattern. In Part 2, you will see how to use the Zend Framework to create the beginnings of the online feed reader, Chomp, creating a form and adding information to a database while getting to know the MVC pattern.

Section 1. Before you start

This tutorial is for developers who want to better understand the MVC pattern, and for those who wish to use and learn more about the Zend Framework. You'll see how to use the Zend_Controller, Zend_View, Zend_Form, and Zend_Db components as the building blocks of your web application.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

About this series

This "Understanding the Zend Framework" series chronicles the building of an online feed reader, Chomp, while explaining the major aspects of using the open source PHP Zend Framework.

In Part 1 I talked about the overall concepts of the Zend Framework, including a list of relevant classes and a general discussion of the MVC pattern. Part 2 expands on that and shows how MVC can be implemented in a Zend Framework application. You will also create the user registration and login process, adding user information to the database and pulling it back out again.

Parts 3 and 4 deal with the actual RSS and Atom feeds. In Part 3, you enable users to subscribe to individual feeds and to display the items listed in those feeds. You also look at some of the Zend Framework's form-handling capabilities, validating data, and sanitizing feed items. Part 4 explains how to create a proxy to pull data from a site that has no feed.

The rest of the series involves adding value to the Chomp application. In Part 5, you look at using the PDF format as a type of backup for saved entries. In Part 6, you use the `Zend_Mail` module to alert users to new posts. In Part 7, you look at searching saved content and returning ranked results. In Part 8, you create your own mashup, leveraging web services from Amazon, Flickr, Twitter and Yahoo! And in Part 9, you add Ajax interactions to the site using JavaScript object notation.

About this tutorial

You'll soon begin developing with the Zend Framework. The goal is to make Chomp the perfect online feed reader. In this tutorial, you will create the basic application, including the registration and login pages, using the Zend Framework implementation of the MVC pattern and its basic database classes. You will learn:

- How to set up Apache and the `mod_rewrite` module
- How to integrate PHP V5 into Apache
- Configuration changes for the Zend Framework
- How to use the MVC pattern
- How to create custom controllers
- How to create custom actions
- How to insert, extract, and update data in a database

© Copyright IBM Corporation 2006, 2011. All rights reserved.

- How to use Zend's general database capabilities

At the end of this tutorial, you will have the basic framework of the application, and you'll add feeds in Part 3.

Prerequisites

This tutorial assumes that you are familiar with PHP. If you're not, check out the Resources to find the "Learning PHP" tutorial series. You should also have basic familiarity with how databases work, but you don't need to be an expert in the use of SQL.

System requirements

To follow along, you will need to have several pieces of software installed. This tutorial will cover installation and configuration, but make sure to download the following:

XAMPP
: XAMPP is an easy-to-install version of Apache, MySQL, and PHP rolled into one. The version used for this tutorial (V1.7.3) contains Apache V2.2.14, PHP V5.3.1, and MySQL V5.1.41.

Zend Framework
: This set of PHP classes is where all the work will be done. This tutorial was tested with V1.10.6.

Section 2. Genie in the magic XAMPP

For your Apache, MySQL, and PHP installations, you're going to use an easy-to-install version of each, rolled into one package called XAMPP. Based on the popularity of the Linux®-Apache-MySQL-PHP (LAMP) platform, XAMPP was developed to make LAMP more accessible to Windows users by building an installer around the key components of the platform. With XAMPP, you can stop, start, restart, and load modules into Apache, MySQL, and PHP using a simple context menu from the WAMP system tray icon. Installation is a snap if you follow the installation wizard, and the configuration files are directly accessible from the context menu after installation.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

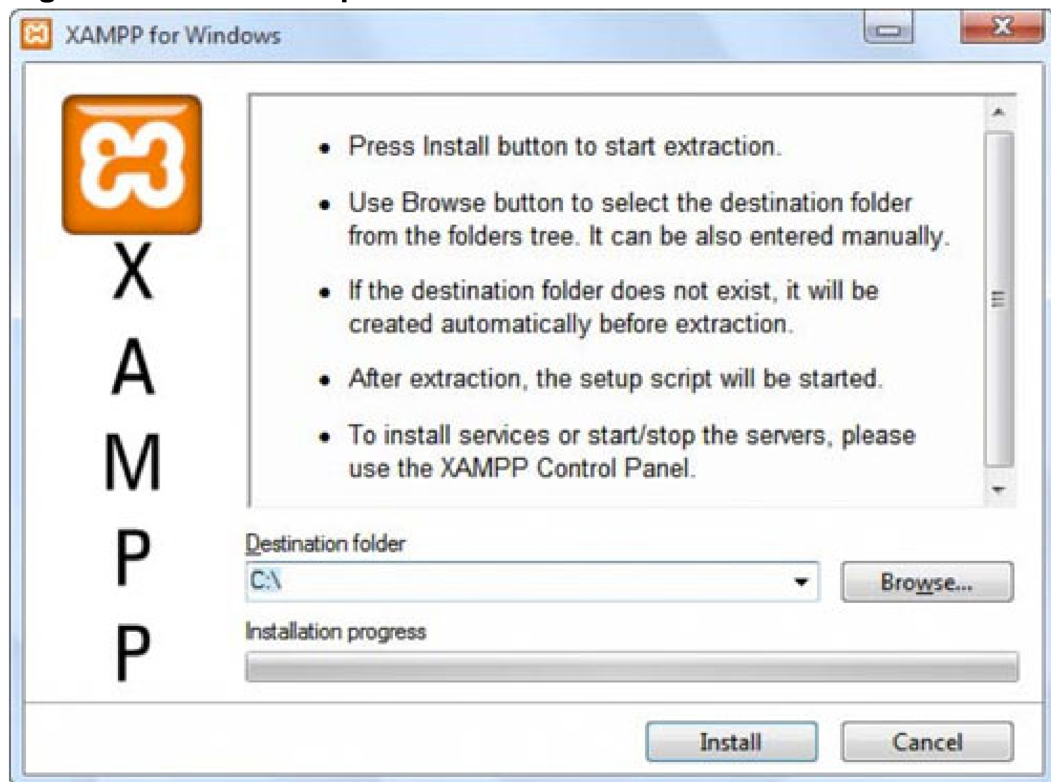
Downloading and installing XAMPP

See the Prerequisites section to download XAMPP. Here's how to install XAMPP:

1. Download the latest .exe file and save it to your drive. Locate the saved file and execute it.

2. The initial splash screen will pop up and ask you for the directory to which you wish to install XAMPP (see Figure 1). After specifying the installation location, click **Next**.

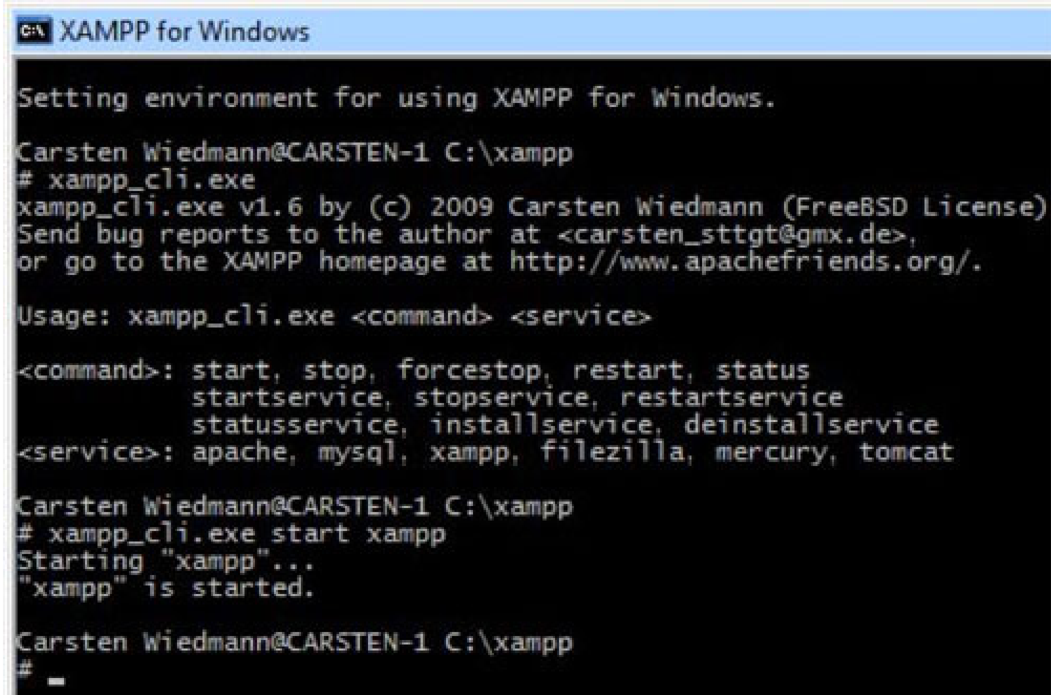
Figure 1. XAMPP setup wizard



3. The installation will then continue, after which the setup_xampp.bat script will run automatically (see Figure 2).

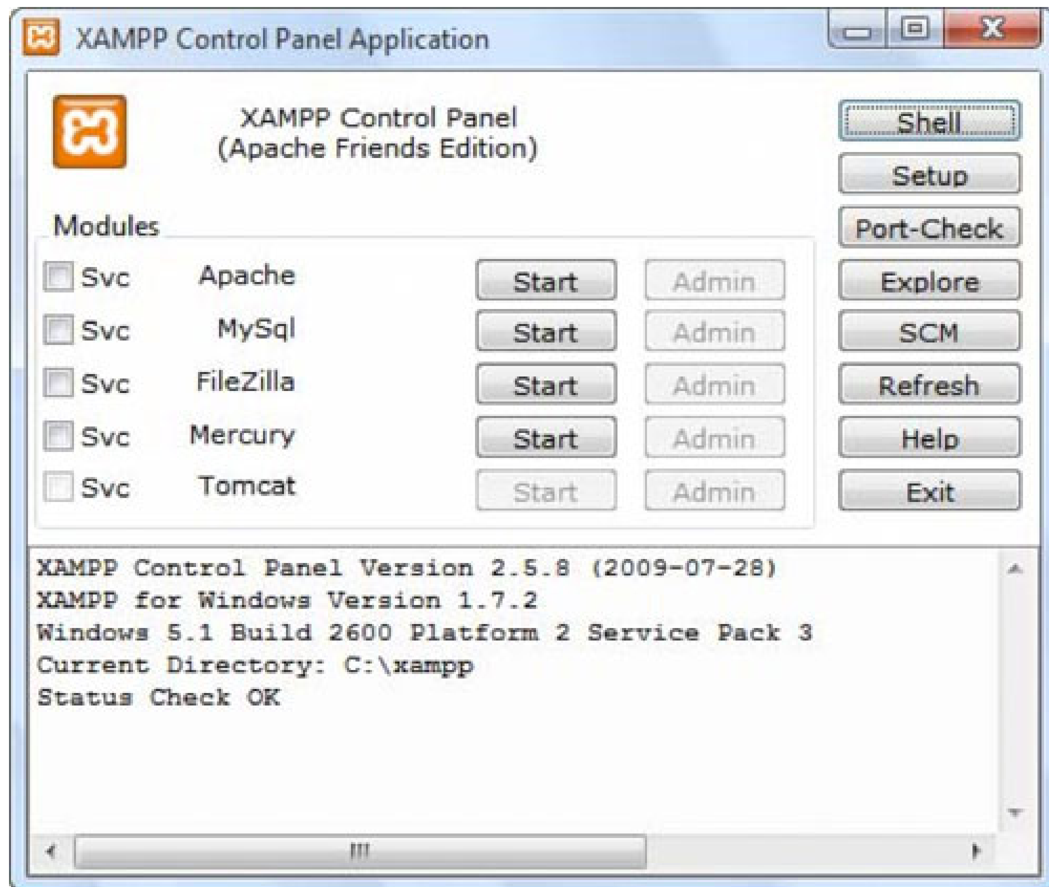
Figure 2. Setup_xampp.bat script execution

© Copyright IBM Corporation 2006, 2011. All rights reserved.



4. You can now start the XAMPP Control Panel (see Figure 3), which allows control of the Apache and MySQL services.
Figure 3. XAMPP Control Panel

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Installing the Zend Framework

The Zend Framework is available at the Zend website (see Prerequisites). Download the ZIP file and unzip it with the directory structure intact to a directory of your choice and make sure the HTTPD.CONF directives that reference it all have the correct path. You'll also add a .htaccess file in the webroot of your application's instance to properly handle all incoming requests (see Listing 1).

Listing 1. .htaccess

```
htaccess

RewriteEngine On
RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
RewriteRule ^.*$ index.php [NC,L]

php_value include_path ".;c:\ZendFramework\library"
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

The `php_value` and `include_path` directives need to accurately point to the right drive. And you will need to change the value of c: to whichever is correct for your installation.

You are now ready to roll.

Section 3. Models, and views, and controllers, oh my!

Now that you've got everything installed and working, it's time to start building the actual site. You'll start with the MVC pattern.

What is the Model-View-Controller pattern?

MVC is one of the more hotly contested patterns, with various groups of individuals having their own views on the subject. But design patterns exist to solve problems — not to cause them — so for the purpose of this tutorial, you'll just discuss the MVC pattern as it relates to the Zend Framework.

The MVC pattern separates application responsibilities into three tiers:

The model
: This is the conceptual representation of the entities involved in an application and would be analogous to a class or object in the object-oriented world. In some of implementations, the model contains only the structure of the entity. In others, it also contains the business logic associated with the object. In our case, the model contains only the structure of the entity.

The view
: This is how your page will look. It does not contain — or at least should not contain — any business logic. The view is a template — in this case, HTML — into which information can be placed.

The controller
: This is where the business logic resides. The controller goes in both directions, controlling the objects and their visual representations, the latter by manipulating the view.

The model

Although the final application will wind up dealing with many more entities, you have

© Copyright IBM Corporation 2006, 2011. All rights reserved.

just three to think about for now:

The user
: This entity represents the website user, and includes data like the user's name and e-mail address. The user will need to be able to log in, log out, and register. In future parts, he will also need to perform actions such as creating a subscription or reading a feed.

The feed
: This entity represents a RSS or Atom feed and, ultimately, will contain other models of feed items.

The index
: This entity actually represents the site itself. After all, the site is an object that will need to display views and perform certain actions. By tradition, it's called the index the same way the file displayed for a URL that does not specify a filename is called index.html (or index.php).

The first view

The view defines the way information is displayed, but not necessarily the information itself. To start, you can add the following content shown in Listing 2 to the index.php file.

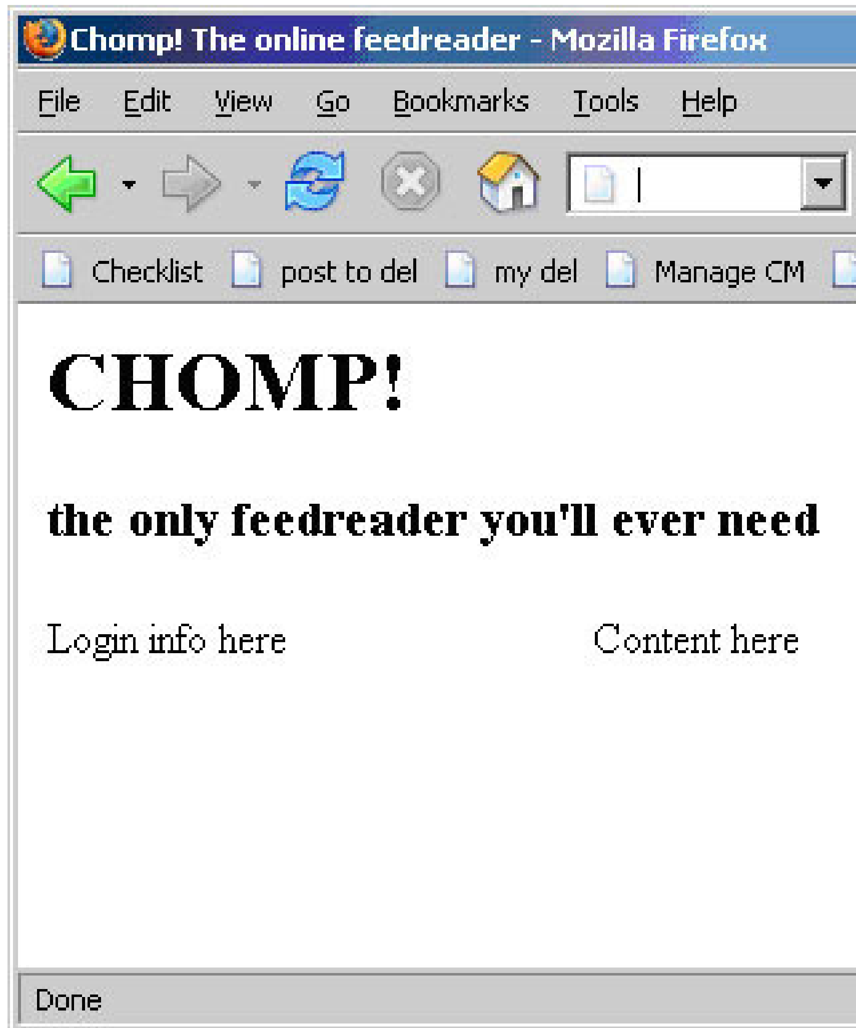
Listing 2. Creating a basic view in the index.php file

```
<html>
<head>
   <title>Chomp! The online feedreader</title>
</head>
<body>

<table>
<tr><td colspan="2">
   <h1>CHOMP!</h1>
   <h3>the only feedreader you'll ever need</h3>
</td></tr>
<tr><td style="width: 200px;">Login info here</td><td>Content here</td></tr>
</table>

</body>
</html>
```

Save this file as index.php in the document root of your application. For example, yours was saved in e:\sw\public_html. To see the view in action, point your browser to http://localhost. You should see something similar to Figure 4.

Figure 4. The view in action

© Copyright IBM Corporation 2006, 2011. All rights reserved.



The first controller

The controller contains the business logic for entities, broken down into actions. For example, in this tutorial, you're only going to ask the site to do three things: display the home page, let a user log in, and register a user. This means you can create a controller for the site, as shown in Listing 3.

Listing 3. The basic controller

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        echo "This is the indexAction().";
    }
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
    public function loginAction()
    {
        echo "This is the loginAction().";
    }

    public function registerAction()
    {
        echo "This is the registerAction().";
    }

}
?>
```

Remember: You're calling the entity that represents the site itself, the index, so its controller is the `IndexController`. Like all controllers, it's a class that extends `Zend_Controller_Action`, so you'll load that first. Once you've created the actual class, you create a function for each action, with the naming convention `<action_name>Action`.

Create a new directory called controllers in the document root for the website and save this file as IndexController.php in that directory.

Calling the controller

The `IndexController` will never be called directly. Remember that when you configured PHP and Apache, you told Apache to forward all requests to the index.php file. This file handles dispatching the request to the controller. Save the HTML text currently in the index.php file in a separate file and add the code in Listing 4 to index.php.

Listing 4. Dispatching the request to the controller

```
<?php

include 'Zend.php';

Zend_Loader::loadClass('Zend_Controller_Front');

Zend_Controller_Front::run('controllers');

?>
```

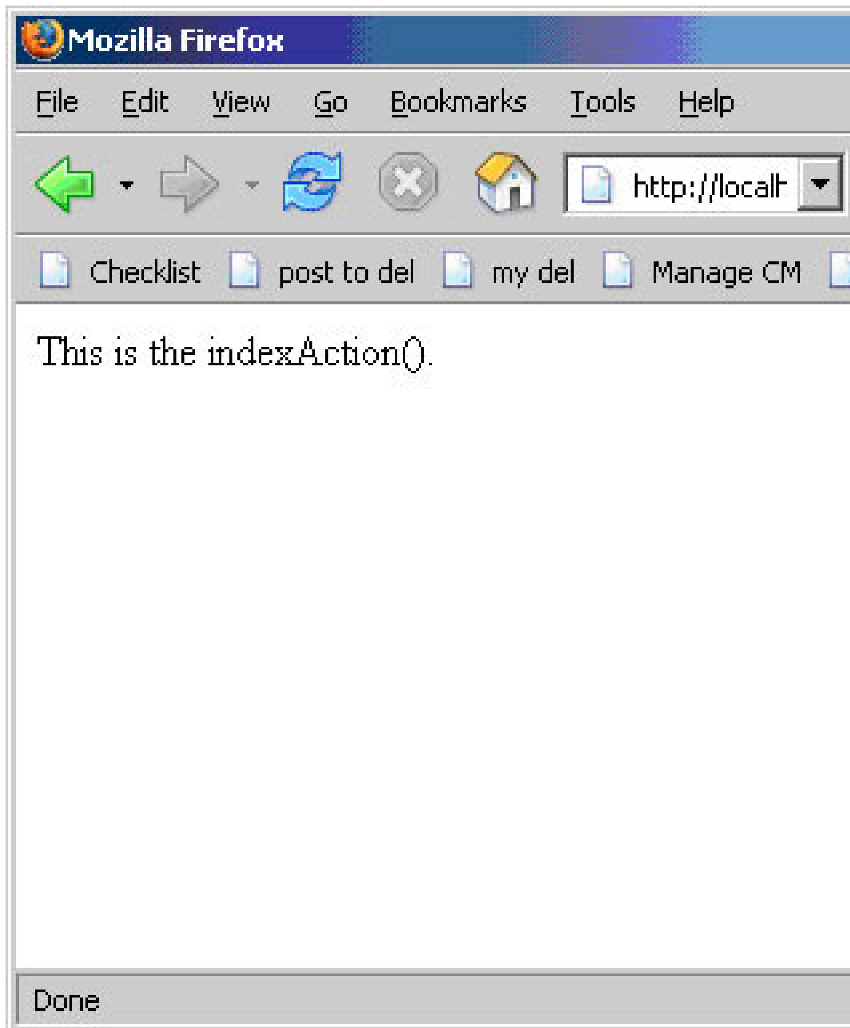
When Apache receives the request, it calls a new instance of the `Zend_Controller_Front::run()` static method, and enables you to set the location of the controller's directory. This also enables you to place controllers in a location outside the document root, where they are less susceptible to mischief. Once you set the directory, the controller can dispatch the request appropriately based on the URL.

Point your browser to `http://localhost`, and you should see results like those

© Copyright IBM Corporation 2006, 2011. All rights reserved.

shown in Figure 5.

Figure 5. Viewing the index results



Because the URL did not specify a particular controller, the Zend Framework chose the `IndexController`, just as Apache will normally look for a file called index.html. Similarly, because there was no action specified, it calls the `indexAction`. You'll see more about specifying controllers and actions later, but for now, let's move on to displaying the view.

Displaying the view

In order to display the view, you have to add it to one of the controller's actions. Start by creating a new directory in the document root (and, if necessary, in <ZEND_HOME>\library) called views. Take the HTML text you saved earlier and save it in this directory, in a file called indexIndexView.php. The actual name of the file is completely arbitrary, but you're going to call this file from the

© Copyright IBM Corporation 2006, 2011. All rights reserved.

IndexController's index action — hence, the name.

To actually call the view, add the code shown in Listing 5 to the IndexController.php file.

Listing 5. Showing the view

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');
        echo $view->render('indexIndexView.php');
    }

    public function loginAction()
    {
        echo "This is the loginAction().";
    }

    public function registerAction()
    {
        echo "This is the registerAction().";
    }

}
?>
```

Here, you have a new class, `Zend_View`, with which you can set a path to any PHP files you want to include. To actually include the file, generate the appropriate HTML using the `render()` function and echo it to the page. The result should be a page that once again looks like Figure 4.

Dynamic view information

If this were all views could do, they would still be useful, but you can make them even handier by including dynamic information. For example, you can add dynamic sections to the view of the home page.

Listing 6. Adding dynamic information

```
<html>
<head>
   <title><?php echo $this->title ?></title>
</head>
<body>

<table>
<tr><td colspan="2">
   <h1>CHOMP!</h1>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
   <h3><?php echo $this->slogan ?></h3>
</td></tr>
<tr><td style="width: 200px;">Login info
 here</td><td>Content here</td></tr>
</table>

</body>
</html>
```

You can put any commands in this file. It's just a straight PHP file. However, the whole point is to restrict this file to simply displaying information. In this case, the `$this` object represents the view itself. Let's look at setting these attributes.

Setting dynamic view information

To set dynamic information on the view, simply set the properties before rendering it, as shown in Listing 7.

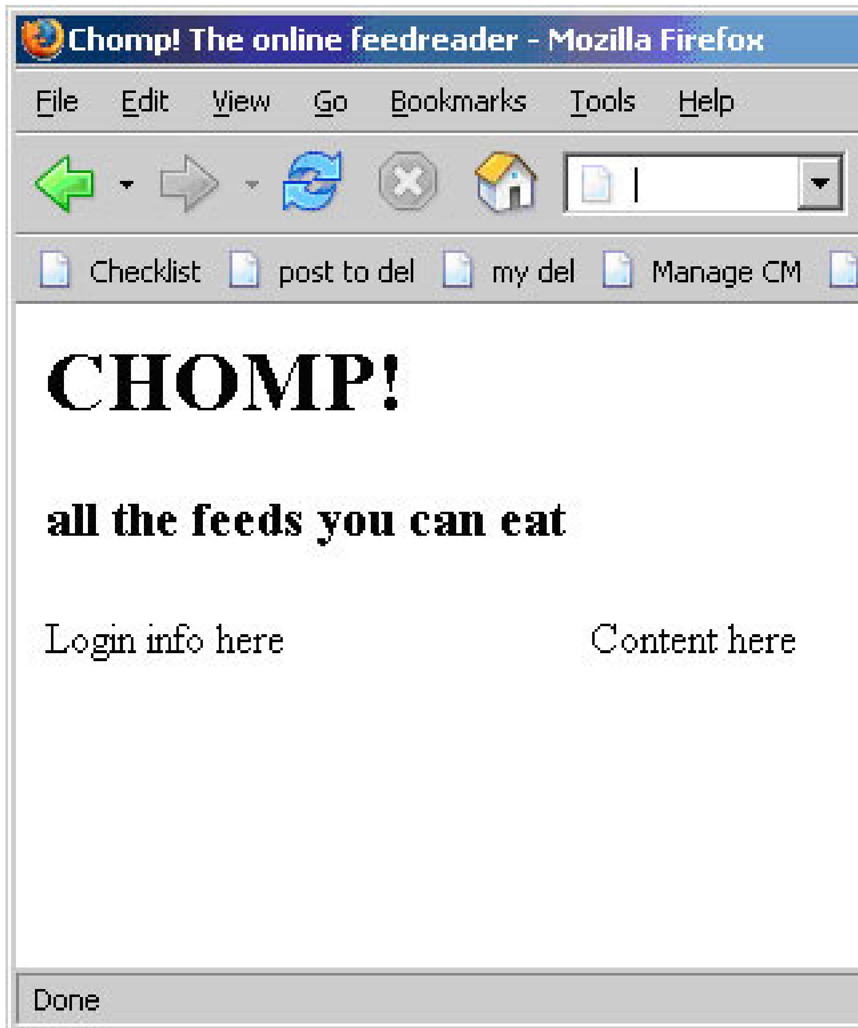
Listing 7. Adding dynamic information

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');

        $view->title = 'Chomp! The online feedreader';
        $view->slogan = 'all the feeds you can eat';

        echo $view->render('indexIndexView.php');
    }

    public function loginAction()
    {
        echo "This is the loginAction().";
    }

    public function registerAction()
    {
        echo "This is the registerAction().";
    }

}
?>
```

The results should be similar to that shown in Figure 6.

Figure 6. A new slogan

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Notice that you have a new slogan. You could, for example, generate random slogans when the user pulls up the page.

Now let's delve a little bit deeper into the world of actions.

Section 4. Working with actions

In a Zend Framework application, everything comes down to a matter of actions. Let's look at how that works.

The UserController

© Copyright IBM Corporation 2006, 2011. All rights reserved.

You'll start with a second controller: the `UserController` (see Listing 8). It contains functions for the various actions the user might take.

Listing 8. The basic UserController

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');

class UserController extends Zend_Controller_Action
{
    function indexAction()
    {
        echo 'This is the indexAction.';
    }

    function loginAction()
    {
        echo 'This is the loginAction.';
    }

    function authenticateAction()
    {
        echo 'This is the authenticateAction.';
    }

    function logoutAction()
    {
        echo 'This is the logoutAction.';
    }

    function registerAction()
    {
        echo 'This is the registerAction.';
    }

    function createAction()
    {
        echo 'This is the createAction.';
    }

    function displayProfileAction()
    {
        echo 'This is the displayProfileAction.';
    }

    function updateAction()
    {
        echo 'This is the updateAction.';
    }

}
?>
```

Here you have just a basic controller, with `echo` statements so you can see what is actually happening. Let's look at the effect of different URLs on the action the Zend Framework tries to take.

Controllers, actions, and the URL

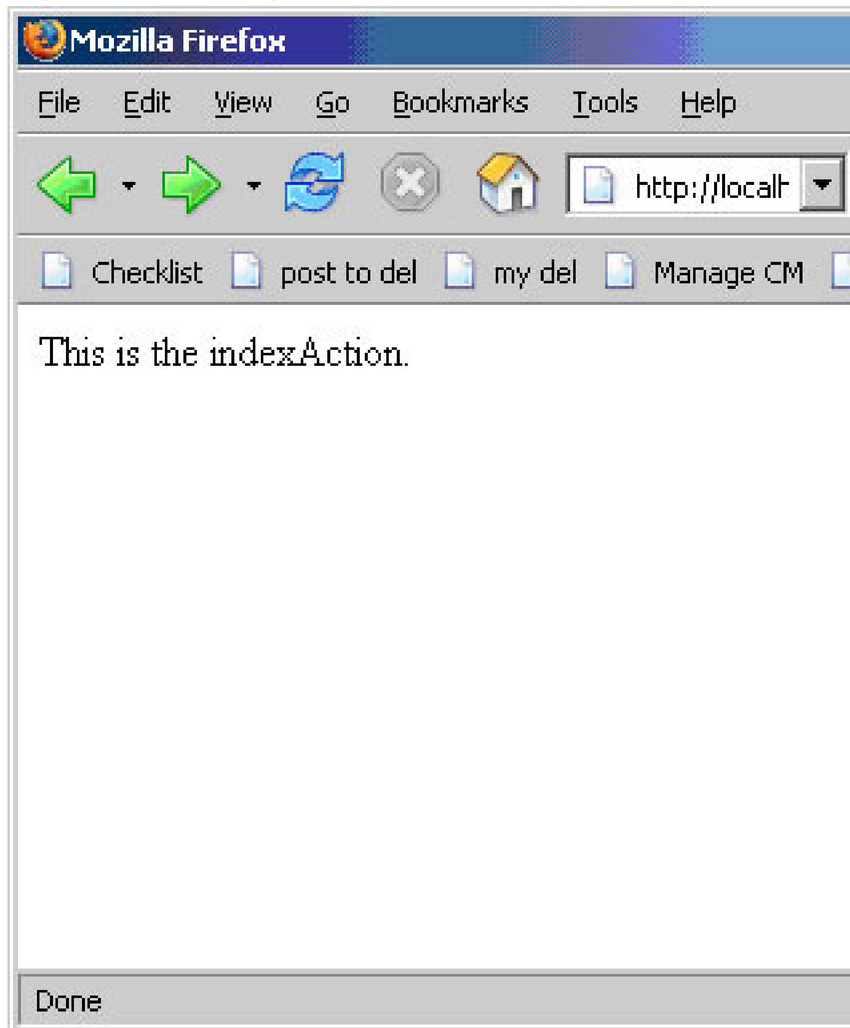
The first thing to understand about the way the Zend Framework routes requests is

© Copyright IBM Corporation 2006, 2011. All rights reserved.

that it breaks the URL into pieces. Those pieces are laid out as follows: http://hostname/controller/action/parameters. In this tutorial, you're only concerned with the controller and the action; you'll handle parameters in later parts of this series.

For example, if you were to call a URL of http://localhost/user/, the Zend Framework sends the request to the `UserController`, but because the URL doesn't specify an action, it uses the `indexAction`, as you can see in Figure 7.

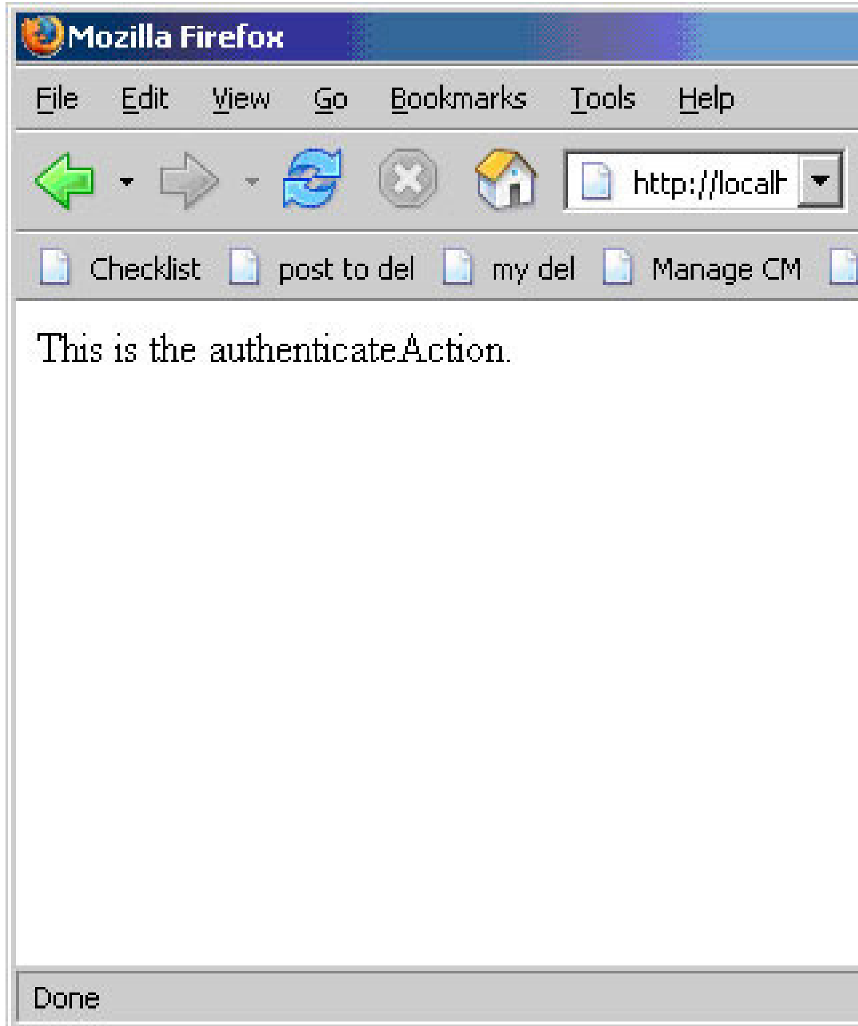
Figure 7. Showing the indexAction



On the other hand, if you use a URL of http://localhost/user/authenticate, the Zend Framework breaks this down into the `UserController` and the `authenticateAction`, so you get results like that shown in Figure 8.

Figure 8. authenticateAction

© Copyright IBM Corporation 2006, 2011. All rights reserved.



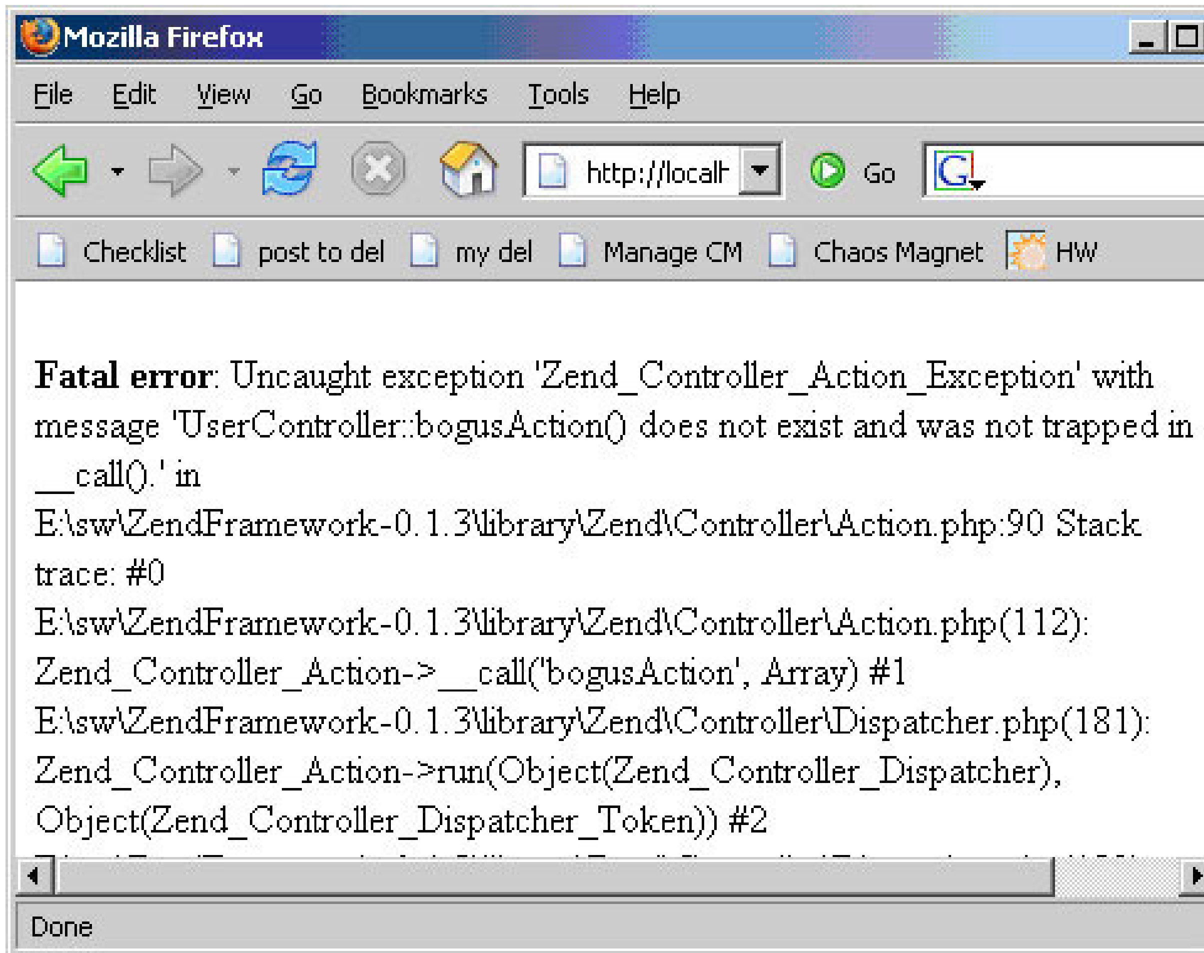
But what happens if the user enters an unknown action?

Dealing with unknown actions

If the URL doesn't specify an action, the Zend Framework uses the `indexAction`. However, what happens if the URL specifies a nonexistent action? For example: http://localhost/user/bogus. As it stands, this URL causes an error because you're calling a nonexistent action, as you can see in Figure 9.

Figure 9. Error

© Copyright IBM Corporation 2006, 2011. All rights reserved.



To fix that, you can create an `ErrorController` (see Listing 9).

Listing 9. Calling unknown actions

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
{

class ErrorController extends Zend_Controller_Action
{
    public function errorAction()
    {
        highlight_string(print_r($this->_getParam('error_handler'),true));
    }
}
```

The `errorAction()` function is specially named by the Zend Framework to be called whenever the URL refers to a nonexistent action.

Let's look at actually working with the database.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Section 5. Inserting data

Now that you have the basic framework of the application, let's start looking at the database.

Setting the action for a form

In this section, you'll build the user registration form. In doing that, you need to understand how to set the action for the HTML form in the browser. When the user submits the form, you want the data to go to the create action of the `UserController`, so you want a URL of `http://localhost/user/create`. To make that happen, you need to create the form, as shown in Listing 10.

Listing 10. The registration form

```
<html>
<head>
</head>
<body>
<p><strong>Register a User Account</strong></p>
<form name="form1" method="post" action="/user/create">
  <p>First Name:
    <input name="fName" type="text" id="fName">
  </p>
  <p>
    Last Name:
    <input name="lName" type="text" id="lName">
  </p>
  <p>Email Address:
    <input name="email" type="text" id="email">
  </p>
  <p>Username:
    <input name="user" type="text" id="user">
  </p>
  <p>Password:
    <input name="pass" type="password" id="pass">
  </p>
  <p>Confirm Password:
    <input name="passConfirm" type=\
    "password" id="passConfirm">
  </p>
  <p>
    <input name="Register" type="submit" id="Register"
 value="Register">
  </p>
</form>
</body>
</html>
```

Save this file as register.php in the views directory.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Display the registration form

To display the registration form, you need to alter the register action for the `UserController` (see Listing 11).

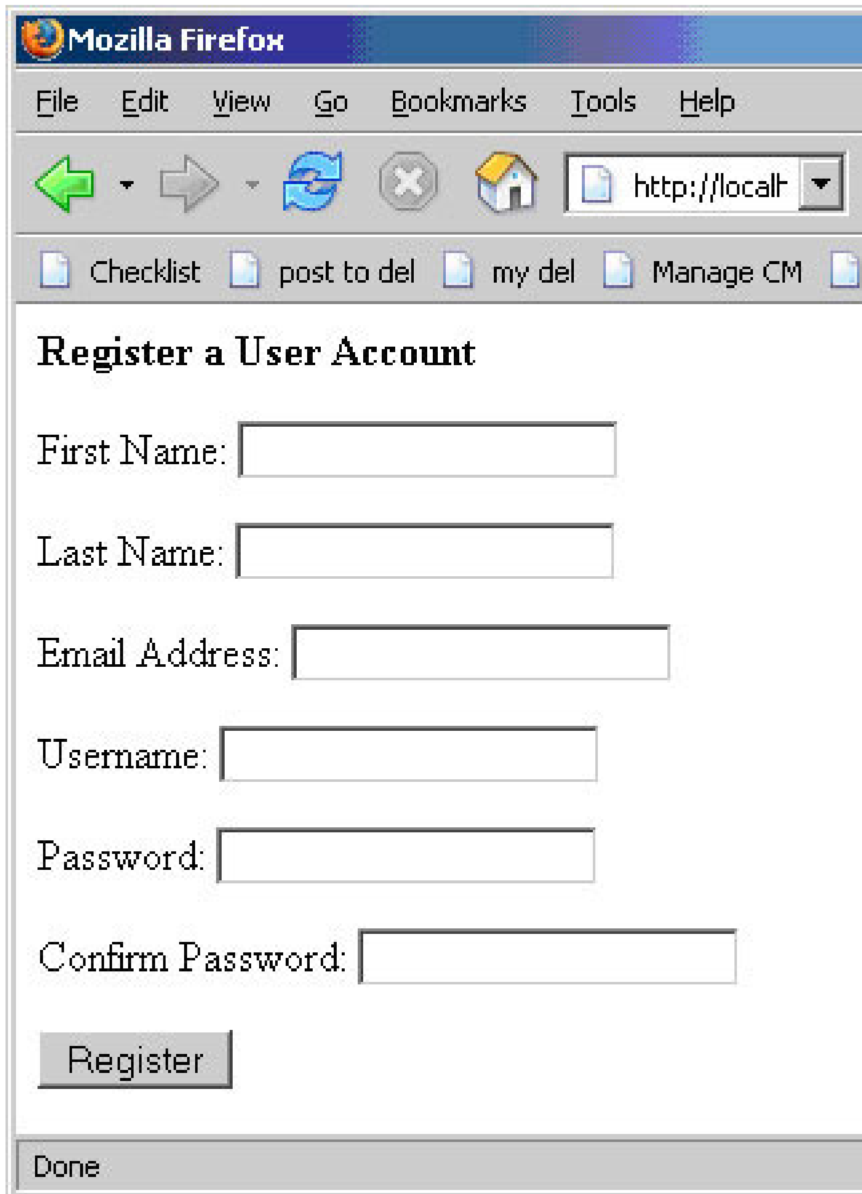
Listing 11. Displaying the registration form

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class UserController extends Zend_Controller_Action
{
...
    function logoutAction()
    {
        echo 'This is the logoutAction.';
    }

    function registerAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');
        echo $view->render('register.php');
    }

    function createAction()
    {
        echo 'This is the createAction.';
    }

...

    }

}
?>
```

The view is simple, so displaying it is straightforward. To see the form, point your browser to http://localhost/user/register. You should see a form similar to that shown in Figure 10.

Figure 10. Register a user account

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Let's look at what happens when the user submits the form.

Connecting to the database

Ultimately, you want to put the data the user entered in the registration form into the database, so the first step is to create a connection to that database (see Listing 12).

Listing 12. Connecting to the database

```
<?php
require_once 'Zend/Db.php';
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class UserController extends Zend_Controller_Action
{

...
    function registerAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');
        echo $view->render('register.php');
    }

    function createAction()
    {
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('PdoMysql', $params);

    }

    function displayProfileAction()
    {
        echo 'This is the displayProfileAction.';
    }

...

}
?>
```

After importing the required file, create an array with the appropriate parameters for the database. In this case, you're using MySQL. If you're using a different database, you may have different parameters. Once you have the array, you can create its connection to the database using the Zend_DB class. The factory can create one of several types of database connections. In this case, you're using the PDO implementation of MySQL.

Executing an insert statement

The simplest, most familiar way to insert the data into the database is to use an `Zend_DB::insert` method call (see Listing 13).

Listing 13. Executing an insert statement

```
...
    function createAction()
    {
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('PdoMysql', $params);

        $data = array(
             'firstName' => $_POST['fName'],
'lastName' => $_POST['lName'],
'emailAddress' => $_POST['email'],
'username' => $_POST['user'],
'password' => new Zend_Db_Expr($db->quoteInto('MD5(?)',$_POST['pass'])),
          );

        $DB->insert('users',$data);

    }

...
```

Here you build an associative array of the fields you plan to insert into the table, then call the `insert()` function to insert the data into the table. This has the added advantage over building the query manually in that this helps prevent SQL injection attacks In a production application, you would check to make sure the two instances of the password match before inserting the data, but for now, let's concentrate on the actual data insertion itself.

Being more general: Creating a row

Because you are executing a SQL statement directly, you'll run into problems if you decide to use a different type of database. One of the points of the Zend_DB classes is that you should be able to write your code once and run it against any database supported by the Zend Framework. To do that, you could use some of the `Zend_Db` functions. For example, you can take the data from the submitted form and turn it into a row object, as shown in Listing 14.

Listing 14. Creating a row

```
...
    function createAction()
    {
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);

        $row = array(
            'firstname' => $_POST['fName'],
            'LastName' => $_POST['lName'],
            'EmailAddress' => $_POST['email'],
            'Username' => $_POST['user'],
            'Password' => $_POST['pass']
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
        );

    }
...
```

As you can see, the row is just an associative array. Now let's insert it into the database.

Inserting data into the database

It's actually very straightforward, as you can see in Listing 15.

Listing 15. Inserting the data

```
...
    function createAction()
    {
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);

        $row = array(
            'FirstName' => $_POST['fName'],
            'LastName' => $_POST['lName'],
            'EmailAddress' => $_POST['email'],
            'Username' => $_POST['user'],
            'Password' => $_POST['pass']
        );

        $table = 'users';
        $rowsAffected = $DB->insert($table, $row);

    }
...
```

You simply specify the name of the table, and feed the table name and the row to the `insert()` function.

Seeing the results

You can see the results of this operation by opening a new command window and typing `mysql -u root -h localhost` (using your own values, of course). You should see results something like what is shown in Listing 16.

Listing 16. The data

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
+-----------+----------+---------------------+----------+----------+
| firstName | lastName | emailAddress        | username | password |
+-----------+----------+---------------------+----------+----------+
| John      | Mertic   | jmertic@gmail.com   | jmertic  | mypasswd |
+-----------+----------+---------------------+----------+----------+
1 row in set (0.00 sec)
```

You know how to get the data into the database, so let's look at getting it out again.

Section 6. Retrieving and updating data

Putting data into the database is useful, but only if you can get it back out again. In this section, you'll look at the login and update profile functions.

The login action

The login action simply displays a form, which acts as the view, as in Listing 17.

Listing 17. The login view

```
<html>
<body>
<p><strong>Chomp Login</strong></p>
<form name="login" method=\
"post" action="/user/authenticate">
  <p>Username
    <input name="username" type="text" id="username">
</p>
  <p>Password
    <input name="password" type="password" id="password">
</p>
  <p>
    <input type="submit" name="Submit" value="Submit">
  </p>
</form>

<p>If you don't have an account yet, please <a
 href="/user/register">register here</a>. </p>

</body>
</html>
```

Place this data into a file and save it as login.php in the views directory. To display the file, call the view from the `loginAction` (see Listing 18).

Listing 18. Displaying the login form

```
<?php
```

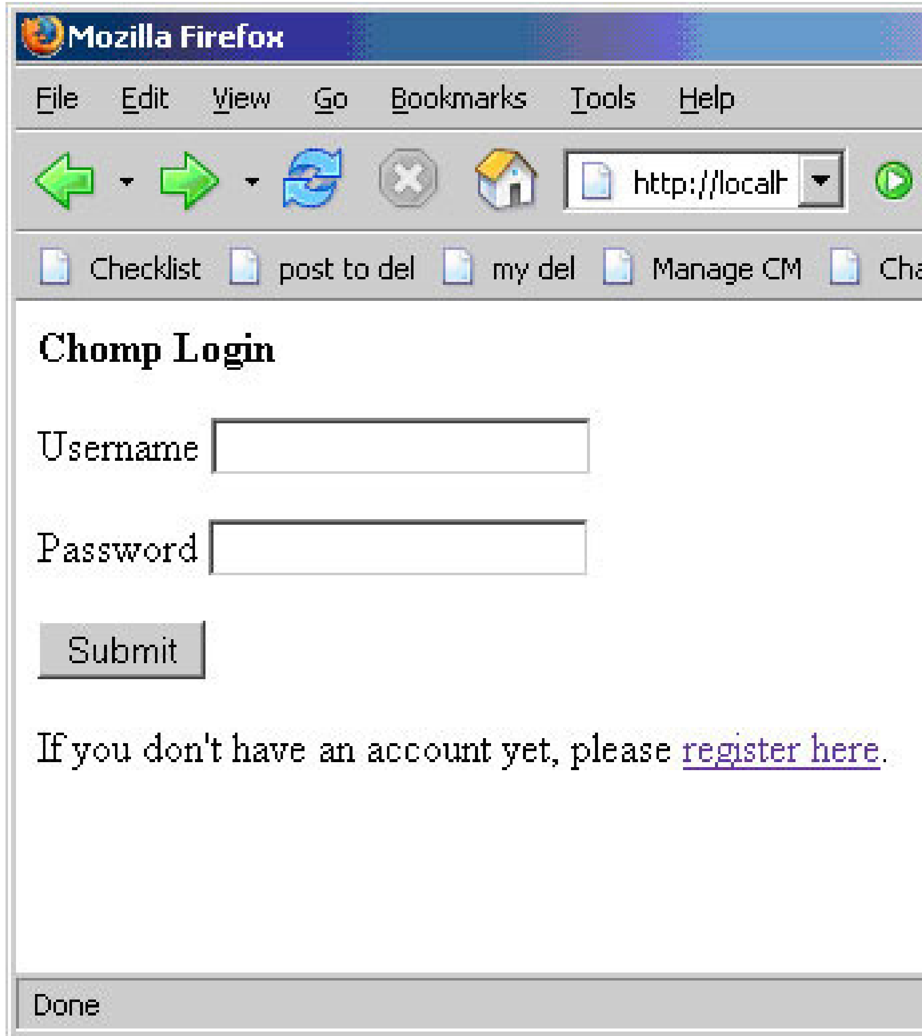

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
require_once 'Zend/Db.php';

Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class UserController extends Zend_Controller_Action
{
    function indexAction()
    {
        echo 'This is the loginAction.';
    }

    function loginAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');
        echo $view->render('login.php');
    }

    function authenticateAction()
    {
        echo 'This is the authenticateAction.';
    }
...
```

Now if you point your browser to http://localhost/user/login, you'll get the login form shown in Figure 11.

Figure 11. Chomp login

© Copyright IBM Corporation 2006, 2011. All rights reserved.



Processing the login

Processing the login involves extracting the appropriate information from the database and comparing it to that which the user entered (see Listing 19.

Listing 19. Processing the login

```
...
    function loginAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('views');
        echo $view->render('login.php');
    }

    function authenticateAction()
    {
        $params = array(
            'host' => 'localhost',
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);

        $select = $DB->select();
        $select->from('users', '*');
        $select->where('Username = ?', $_POST['username']);
        $select->where('Password = ?', $_POST['password']);
        $sql = $select->__toString();

        echo $sql;

    }

    function logoutAction()
...
```

In this case, you are connecting to the database as before, but rather than performing a straight SQL `select`, you're creating a `select` object starting with everything, and filtering down to the appropriate username and password. Just to make sure you're pulling the appropriate data, you can output the SQL statement by calling its `__toString()` method.

Once you're convinced the query is correct, you can execute it.

Executing the query

After building the query, you need to execute against the database, as shown in Listing 20.

Listing 20. Executing the query

```
...
    function authenticateAction()
    {
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);

        $select = $DB->select();
        $select->from('users', '*');
        $select->where('Username = ?', $_POST['username']);
        $select->where('Password = ?', $_POST['password']);
        $sql = $select->__toString();

        $rowsFound = $DB->fetchAll($select);
        if (isset($rowsFound[0]["username"]))
        {
           $this->_redirect('/');
        }
```

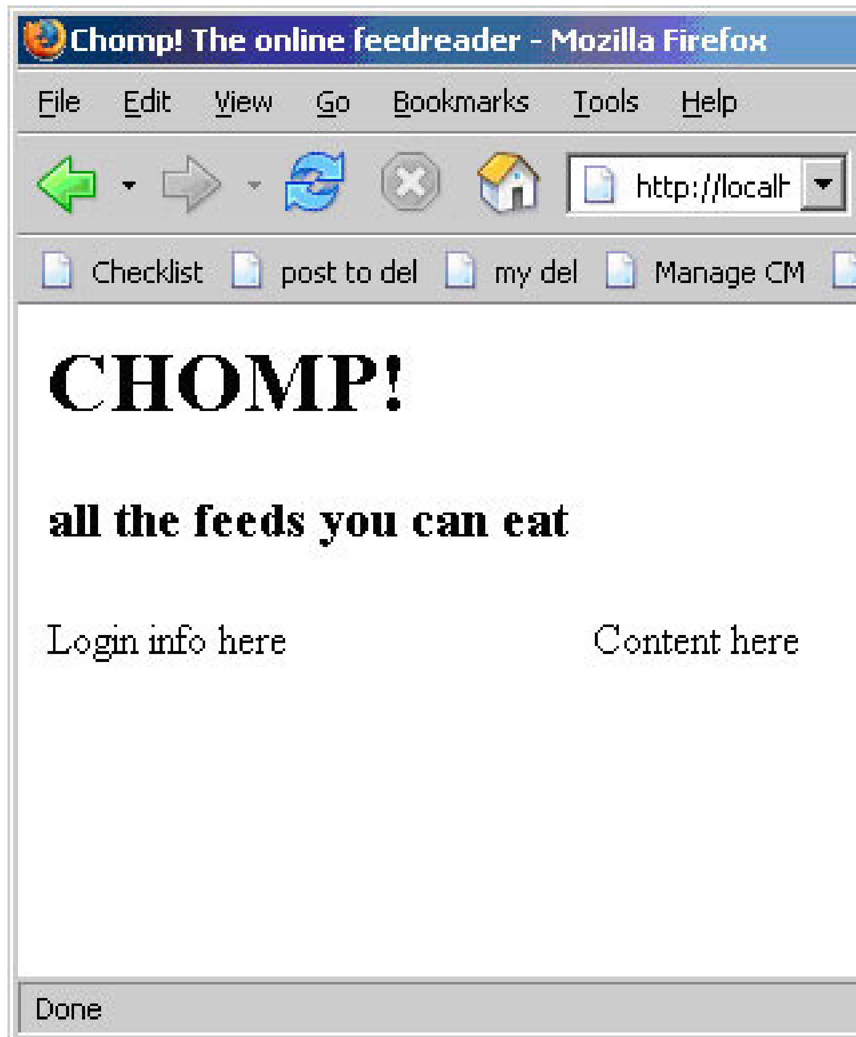

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
        else
        {
            echo "Login information incorrect. Please <a
href='/user/login'>try again</a>.";
        }

    }
...
```

The `fetchAll()` function retrieves all the records that satisfy the `select` statement into a 2-D array. The first dimension is the row, and the second is the column. So the expression `$rowsFound[0]['username']` refers to the username column of the first row found (see Figure 12).

Of course, there will only be a row found if the username and password match the information in the database. If so, redirect the user to the home page. (You'll do more than that in a moment.)

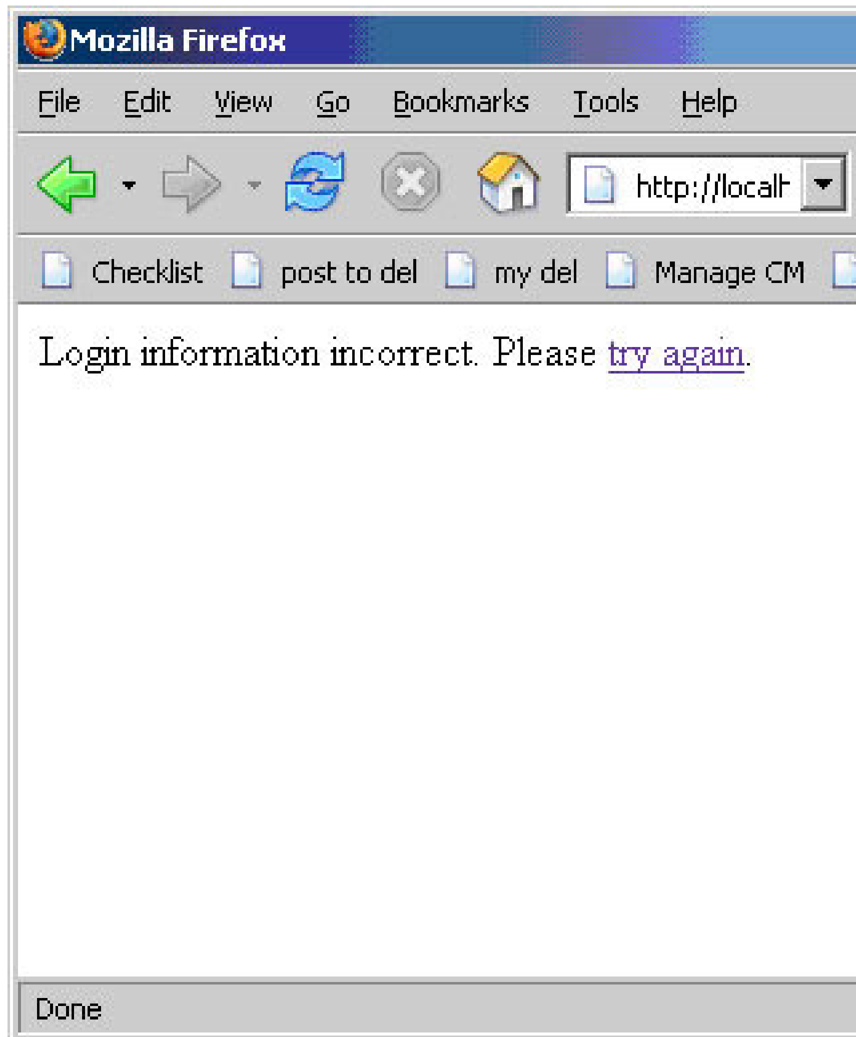
Figure 12. Retrieved records



© Copyright IBM Corporation 2006, 2011. All rights reserved.

If no record is found, give the user a chance to try again (see Figure 13).

Figure 13. Try again



Setting session information

If the `select` statement actually pulls a record, then you know that the username and password of at least one record matches what the user entered on the login form. Now you need to let the site know that the user is logged in. To do that, you will set session variables, as shown in Listing 21.

Listing 21. Setting session information

```
...
    function authenticateAction()
    {
        $params = array(
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);

        $select = $DB->select();
        $select->from('users', '*');
        $select->where('Username = ?', $_POST['username']);
        $select->where('Password = ?', $_POST['password']);
        $sql = $select->__toString();

        $rowsFound = $DB->fetchAll($select);
        if (isset($rowsFound[0]['username']))
        {
           session_start();
           $_SESSION['RSS_SESSION'] = session_id();
           $_SESSION['firstname'] = $rowsFound[0]['firstname'];
           $_SESSION['username'] = $rowsFound[0]['username'];

           echo "Login successful. Please <a
 href='/'>continue</a>.";
        }
        else
        {
           echo "Login information incorrect. Please <a
 href='/user/login'>try again</a>.";
        }

    }
...
```

In the `authenticateAction()` function, you first tell PHP to start the session or use an existing session, if possible. You can then set values on the `$_SESSION` variable. This variable data is an auto-global, meaning that it is available from anywhere in the application. It will also last as long as the user has his browser open.

You have three values. The first is the session ID, so you have a unique way to refer to the session. The second is the user's first name, so you can be polite and greet him from the home page. And the third is the user's username, so you can perform any user-related functions.

Retrieving session to display on the page

To display this new user data on the home page, you need to add it to the appropriate view object. Add the code from Listing 22 to the `indexController`.

Listing 22. Adding information to the home page

```
<?php
Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {

        session_start();

        $view = new Zend_View();
        $view->setScriptPath('../application/views/');

        $view->title = 'Chomp! The online feedreader';
        $view->slogan = 'all the feeds you can eat';

        if (isset($_SESSION['username']))
        {
            $view->loginSection = 'Welcome back, ' .
 $_SESSION['firstname'];
        } else {
            $view->loginSection = 'Login info here';
        }

        echo $view->render('indexIndexView.php');
    }

    public function loginAction()
    {
        echo "This is the loginAction().";
    }

    public function registerAction()
    {
        echo "This is the registerAction().";
    }

}
?>
```

If the user has logged in, the `$_SESSION['username']` variable will be set, so you can display the user's first name in the log and section. If not, you go back to the original login information. (Note that in later parts of this series, you'll see that you can actually call other code here, including actions from other controllers.)

Updating the view

Of course, to see this in action, you need to update the view itself. Open the indexIndexView.php file and add the information shown in Listing 23.

Listing 23. Adding the new information

```


<html>
<head>
   <title><?php echo $this->title ?></title>
</head>
<body>

<table>
<tr><td colspan="2">
   <h1>CHOMP!</h1>
   <h3><?php echo $this->slogan ?></h3>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
</td></tr>
<tr>
<td style="width: 200px;"><?php echo $this->loginSection
 ?></td>


<td>Content here</td></tr>
</table>


</body>
</html>
```

You can also use this view to provide a link for updating the user's profile.

The profile update form

An advantage of using views is that you can use the same file for multiple purposes. For example, the registration form can double as a profile update form. To do that, you need to alter the register.php file so that it can accept dynamic information (see Listing 24).

Listing 24. Adding dynamic information to the registration form

```


<html>
<head>
</head>
<body>
<p><strong><?php echo $this->userFunction; ?> a User Account</strong></p>
<form name="form1" method="post" action="<?php echo $this->action; ?>">
  <p>First Name:
    <input name="fName" type="text" id=\
    "fName" value="<?php echo $this->firstname; ?>">
</p>
  <p>
    Last Name:
    <input name="lName" type="text" id="lName" value=\
    "<?php echo $this->lastname; ?>">
</p>
  <p>Email Address:
    <input name="email" type="text" id="email" value=\
    "<?php echo $this->email; ?>">
  </p>
  <p>Username:
    <input name="user" type="text" id="user" value=\
    "<?php echo $this->username; ?>">
</p>
  <p>Password:
    <input name="pass" type="password" id="pass" >
  </p>
  <p>Confirm Password:
    <input name="passConfirm" type="password" id="passConfirm">
  </p>
  <p>
    <input name="<?php echo $this->button; ?>" type="submit"
       id="<?php echo $this->button; ?>"
       value="<?php echo $this->button; ?>">
  </p>
</form>
</body>
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
</html>
```

With these additions in place, you can use this view for the registration and the profile update form.

Displaying the profile form

Displaying the form involves making sure the view object has the appropriate attributes (see Listing 25).

Listing 25. Setting the appropriate attributes

```
<?php
require_once 'Zend/Db.php';

Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class UserController extends Zend_Controller_Action
{
...

    function registerAction()
    {
        $view = new Zend_View();
        $view->setScriptPath('../application/views/');
        $view->userFunction = 'Create';
        $view->action = '/user/create';
        $view->button = 'Register';
        echo $view->render('register.php');
    }

    function createAction()
{
        $params = array(
            'host' => 'localhost',
            'username' => 'chompuser',
            'password' => 'chomppassword',
            'dbname' => 'chomp'
        );

        $DB = Zend_Db::factory('pdoMysql', $params);


        $row = array(
            'FirstName' => $_POST['fName'],
            'LastName' => $_POST['lName'],
            'EmailAddress' => $_POST['email'],
            'Username' => $_POST['user'],
            'Password' => $_POST['pass']
        );

        $table = 'users';
        $rowsAffected = $DB->insert($table, $row);

        $this->_redirect('/');

    }

    function displayProfileAction()
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
    {
        session_start();
        if (isset($_SESSION['username']))
        {
            $params = array(
                'host' => 'localhost',
                'username' => 'chompuser',
                'password' => 'chomppassword',
                'dbname' => 'chomp'
            );

            $DB = Zend_Db::factory('pdoMysql', $params);
            $select = $DB->select();
            $select->from('users', '*');
            $select->where('Username = ?', $_SESSION['username']);

            $rowsFound = $DB->fetchAll($select);

            $firstname = $rowsFound[0]["firstname"];
            $lastname = $rowsFound[0]["lastname"];
            $email = $rowsFound[0]["emailaddress"];
            $username = $rowsFound[0]["username"];
            $password = $rowsFound[0]["password"];

            $view = new Zend_View();
            $view->setScriptPath('views');
            $view->userFunction = 'Update';
            $view->action = '/user/update';
            $view->button = 'Update';
            $view->firstname = $firstname;
            $view->lastname = $lastname;
            $view->email = $email;
            $view->username = $username;

            echo $view->render('register.php');
       }
       else
       {
           $this->_redirect('/');
       }

    }

    function updateAction()
    {
        echo 'This is the updateAction.';
    }
}
?>
```

When you're using the view for the registration form, you need to set only the title, action, and button name. The rest of the information, such as the first name, email address, etc., remains blank, which is what you want.

The `displayProfileAction`, on the other hand, involves a bit more work. First, you create a `$select` object, specifying the `source` and the `where` clause. For the `where` clause, specify the clause itself, using a `?` as a placeholder and providing the data to fill that placeholder. You can then use that `select` statement to pull the data from the database and assign the retrieved data to variables. You then assign the variables to the attributes of the view and render the view. This way, if you point your browser to http://localhost/user/displayProfile, you should see something similar to Figure 14.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Figure 14. Update a user account

Updating data

When the user submits the form, the browser sends the information to the `updateAction` (see Listing 26).

Listing 26. Updating user information

© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
...
            echo $view->render('register.php');
        }
        else
        {
            $this->_redirect('/');
        }

    }

    function updateAction()
    {
        session_start();
        if (isset($_POST))
        {
            $params = array(
                'host' => 'localhost',
                'username' => 'chompuser',
                'password' => 'chomppassword',
                'dbname' => 'chomp'
            );

            $DB = Zend_Db::factory('pdoMysql', $params);
            $table = 'users';

            $record = array (
                        'firstname' => $_POST['fName'],
                        'lastname' => $_POST['lName'],
                        'emailaddress' => $_POST['email'],
                        'username' => $_POST['user'],
                        'password' => $_POST['pass']
                      );

            $row = $DB->quoteInto('Username =?',
                                  $_SESSION['username']);

            $rowsAffected = $DB->update($table, $record, $row);

            echo 'User record updated.';
        }
        else
        {
            $this->_redirect('/');
        }

    }
}
?>
```

First, activate the session so you can be sure you're only changing the information for the logged-in user. Next, you check to make sure the form has actually been submitted, and if it has, you connect to the database.

Your goal here is to update a specific record in the database with new information. To do that, you need to provide three pieces of information: the name of the table; the new record, represented as an array; and the `where` clause that identifies the record or records you want to update.

This last one is the least familiar: using the `quoteInto()` function. Essentially, you are simply building the statement by replacing the question mark with the session variable, but the `quoteInto()` function adds any quotes as appropriate, even

© Copyright IBM Corporation 2006, 2011. All rights reserved.

escaping quotes in the middle of the text, if necessary. Finally, you feed all three objects to the `update()` function, which returns the number of rows updated.

In the event that this action was called from anything other than the submission of a form, the user is simply redirected to the home page.

Section 7. Using the registry

So far, you've covered how to make all of this work, but there's one more feature you need to look at that will make your life easier: the Zend_Registry.

What is the Zend_Registry?

The Zend_Registry is a place to store all of those objects you're going to use repeatedly in your application. For example, if you have a view that you use in multiple places, storing it here will enable you to have a single instance you can change without having to find all of the instances throughout replication.

The registry can only store objects and stores them with a string value name by which you can later retrieve them. You may not remove an object from the registry, but understand that it exists only in the context of your current request, anyway. It is not a means for achieving persistent storage of objects, but, rather, a way to avoid storing them in session variables.

In your case, you can see it in action by storing the database connection in the registry, avoiding the necessity of recreating it every time you want to access the database.

Registering an object

Because all requests flow through the index.php file, that's the most likely place for us to create the database connection and register (see Listing 27).

Listing 27. Creating the database connection

```
<?php

require_once 'Zend/Loader.php';
Zend_Loader::loadClass('Zend_Db');
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Registry');
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
$params = array(
             'host' => 'localhost',
             username' => 'chompuser',
             'password' => 'chomppassword',
             dbname' => 'chomp'
             );
$db = Zend_Db::factory('Pdo_Mysql', $params);
Zend_Registry::set('db', $db);
$baseURL = str_replace('/index.php','/',$_SERVER['PHP_SELF']);
Zend_Registry::set('baseURL', $baseURL);

$front = Zend_Controller_Front::getInstance();
$front->setBaseUrl($baseURL);
$front->setControllerDirectory('../application/controllers');
$front->setParam('noViewRenderer', true);
$front->dispatch();

?>
```

Here, you simply create the database connection as usual, adding it to the registry using the `register()` function and assigning it a name of `DB`. Now, if you need to change a username, password, or other information, simply change it here, and you're done. You will probably want to put this information in a less-vulnerable file and simply include it.

Accessing the registered object

Once the database connection has been registered, you can access it easily. At the moment, the only file that uses the database is the `UserController` (see Listing 28).

Listing 28. Accessing the registry

```
<?php
require_once 'Zend/Db.php';

Zend_Loader::loadClass('Zend_Controller_Action');
Zend_Loader::loadClass('Zend_View');

class UserController extends Zend_Controller_Action
{

...

    function authenticateAction()
    {
        $DB = Zend_Registry::get('db');

        $select = $DB->select();
        $select->from('users', '*');
        $select->where('Username = ?', $_POST['username']);
        $select->where('Password = ?', $_POST['password']);
        $sql = $select->__toString();

        $rowsFound = $DB->fetchAll($select);
        if (isset($rowsFound[0]['username']))
        {
```


© Copyright IBM Corporation 2006, 2011. All rights reserved.

```
            session_start();
            $_SESSION['RSS_SESSION'] = session_id();
            $_SESSION['firstname'] = $rowsFound[0]['firstname'];
            $_SESSION['username'] = $rowsFound[0]['username'];

            echo "Login successful. Please <a href='/'>continue</a>.";
         }
         else
         {
            echo "Login information incorrect. \
            Please <a href='/user/login'>try again</a>.";
         }

     }

     function logoutAction()
     {
         echo 'This is the logoutAction.';
     }

     function registerAction()
     {
         $view = new Zend_View();
         $view->setScriptPath('../application/views/');
         echo $view->render('register.php');
     }

     function createAction()
     {
         $DB = Zend_Registry::get('db');

         $row = array(
             'FirstName' => $_POST['fName'],
             'LastName' => $_POST['lName'],
             'EmailAddress' => $_POST['email'],
             'Username' => $_POST['user'],
...
```

Notice that all you have done here is replace the database connection information with a call to the registry asking for the object reregistered under the string `DB`. Once you have the object, you use it just as you used it before.

Section 8. Summary

The Zend Framework was designed to provide an easy way to perform common actions in developing PHP applications. Its reliance on the MVC pattern takes a little bit of getting used to, but once you do, it opens up a lot of possibilities and organizes your code in such a way that programming becomes much less complex by separating code into controllers — organized by entity — and actions. Actions can call dynamically generated views.

The Zend Framework also provides components to make your programming life easier, including a database management class that simplifies the process of adding

© Copyright IBM Corporation 2006, 2011. All rights reserved.

information to and retrieving information from the database.

In this tutorial, you built the basic framework of the Chomp online feed reader, putting into place the MVC pattern and creating a user registration and login system. In Part 3, you add the actual feeds to the application.

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Downloads

Description	Name	Size	Download method
Part 2 source code	os-php-zend2.zip	3.3KB	HTTP

Information about download methods

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Resources

Learn

- The Zend Framework website maintains the latest documentation.
- To better understand the goals of the feed-reader project, read "Introduction to Syndication, (RSS) Really Simple Syndication," by Vincent Luria.
- The PHP Function Reference is another great resource.
- Thought Storms ModelViewController explains the MVC and the controversy and confusion surrounding it.
- The phpPatterns Web site explains MVC from the PHP point of view.
- Learn more about Zend Core for IBM, a seamless, easy-to-install, and supported PHP development and production environment that features integration with DB2 and Informix databases.
- Implement SOAP services with the Zend Framework (Vikram Vaswani, developerWorks, May 2010): Check out this article on using SOAP Web services with the Zend Framework.
- The developerWorks tutorial series "Learning PHP" takes you from the most basic PHP script to working with databases and streaming from the file system.
- PHP.net is the central resource for PHP developers.
- Check out the "Recommended PHP reading list."
- Browse all the PHP content on developerWorks.
- Follow developerWorks on Twitter.
- Expand your PHP skills by checking out IBM developerWorks' PHP project resources.
- To listen to interesting interviews and discussions for software developers, check out developerWorks podcasts.
- Stay current with developerWorks' Technical events and webcasts.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products, as well as our most popular articles and tutorials.
- Watch and learn about IBM and open source technologies and product

© Copyright IBM Corporation 2006, 2011. All rights reserved.

functions with the no-cost developerWorks On demand demos.

Get products and technologies

- Download the Zend Framework from Zend Technologies.
- Download PHP V5.x from PHP.net.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.
- Download IBM product evaluation versions or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.
- Participate in developerWorks blogs and get involved in the developerWorks community.
- Participate in the developerWorks PHP Forum: Developing PHP applications with IBM Information Management products (DB2, IDS).

About the authors

Nicholas Chase

Nicholas Chase has been involved in Web-site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. He has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, an Oracle instructor, and the chief technology officer of an interactive communications company. He is the author of several books, including *XML Primer Plus* (Sams).

Tracy Peterson

Tracy Peterson has worked as an IT project manager and Web developer since 1997 and most recently worked as an operations program manager on MSN Search at Microsoft. He is based in San

© Copyright IBM Corporation 2006, 2011. All rights reserved.

Francisco.

© Copyright IBM Corporation 2006, 2011. All rights reserved.