

---

# Understanding the Zend Framework, Part 3: The feeds

## Building the perfect reader

Skill Level: Intermediate

[Tyler Anderson](#)  
Freelance Writer  
Backstop Media

18 Jul 2006

Updated 18 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 recently introduced open source PHP Zend Framework. Parts 1 and 2 discuss the goals behind the Zend Framework and show you how to use it to create the beginnings of your online feed reader, creating a form and adding information to a database while getting to know the MVC pattern. In this tutorial, you will see how to use the Zend Framework to implement the online feed-reader portion of the application.

## Section 1. Before you start

This tutorial is for developers who want to understand the Model-View-Controller (MVC) pattern, pushing beyond what is covered in [Part 2](#), as well as how to build a feed reader using the `Zend_Feed` and `Zend_Filter_Input` modules of the Zend Framework.

## 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 recently introduced open source PHP Zend Framework.

[Part 1](#) looked at the overall concepts of the Zend Framework, including a list of relevant classes and a general discussion of the MVC pattern. [Part 2](#) expanded on that to show how MVC can be implemented in a Zend Framework application. You also created 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. Here in Part 3, you enable users to subscribe to individual feeds and to display the items listed in those feeds. You will 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. [Part 7](#) looks at searching saved content and returning ranked results. In [Part 8](#), you create your own mashup, adding information from Amazon, Flickr, Twitter and Yahoo! And in [Part 9](#), you add Ajax interactions to the site using JavaScript object notation.

## About this tutorial

In this tutorial, you add feeds into the equation, enabling the user to create an account, subscribe to particular feeds, and display those feeds. This tutorial uses the `Zend_Feed` and `Zend_Input_Filter` modules, the latter used to verify email addresses and strip HTML tags from feed entries. You will learn:

- How to use the `Zend_Input_Filter` module to:
  - Filter incoming GET or POST data
- About RSS feeds
- How to use the `Zend_Feed` module to read data from RSS feeds
- How to save, view, and delete feed entries to or from the database for later access

At the end of this tutorial, you will have the basic framework of the feed-reader application and be ready to start saving entire articles from non-RSS supporting websites to the database in Part 4.

## Prerequisites

This tutorial assumes that you are familiar with PHP. (If you're not, see [Resources](#) for 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 (you'll be especially fine if you have completed [Part 2](#) of this series). And you should also be familiar with RSS feeds.

## System requirements

To follow along, you will need to have some pieces of software installed. Look back at [Part 2](#) for details on installation and configuration of 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. Overview

In this section, you'll learn about RSS feeds and the data they provide. Next, you'll learn about the new database schema this tutorial requires, then you'll create and initialize new tables. Last, you'll go over some miscellaneous functionality, including auto-loading classes, registering the view object, and logging out.

## RSS feeds

The most popular feed-reading format is done via RSS. It stands for several things: RDF Site Summary, Rich Site Summary, or Really Simple Syndication. It's an XML format for distributing headline data and other information over the Internet. No special parameters are necessarily sent to the site; you just have to know the URL for the RSS feed you want information from and point your browser there.

You can take a look at what RSS data looks like by pointing your browser to [Google](#)

[News](#). Now you can see all the information returned in RSS XML format. There is a lot of information here, and for this tutorial, you'll only focus on the headline data: the title and the URL to get to the full article.

## New database schema

You should already have the user table from Part 2, so you're going to add a few more tables here:

### **feeds**

This is used to store the available feeds. It has two fields: `feedname` and `link`.

### **subscribedfeeds**

This holds all the feeds to which a user is subscribed. Subscribing to a feed adds a row to this table with the user and the name of the feed being subscribed to. There are two fields: `username` and `feedname`.

### **savedentries**

This holds all the RSS feed entries a user has saved. This way, a user can revisit them later. There are four fields: `username`, `feedname`, `channelname` (article title), and `link`.

## Creating new tables

You create the new tables for the database next. Open a MySQL window and add the code in [Listing 1](#).

### **Listing 1. Creating the new tables**

```
use chomp;

create table feeds
  (feedname varchar(256), link varchar(512));

create table subscribedfeeds
  (username varchar(20), feedname varchar(256));

create table savedentries
  (username varchar(20), feedname varchar(256),
   channelname varchar(256), link varchar(512));
```

The new database tables now exist in your database.

## Initializing the new tables

The feeds table is initialized with the titles and URLs of all the RSS feeds you wish to have available on your feed reader. Add the code in Listing 2 the same MySQL window:

### Listing 2. Initializing database tables

```
insert into feeds values
('Fox Sports', 'http://feeds.feedburner.com/foxsports/rss/headlines'),
('Google News', 'http://news.google.com/?output=rss'),
('Yahoo News', 'http://rss.news.yahoo.com/rss/topstories');

insert into subscribedfeeds values ('tsa', 'Google News');
```

You now have three available feeds in your application: Fox Sports, Google News, and Yahoo News. You have also added a subscribed feed to begin with to the subscribedfeeds table.

## Class auto-loading

The Zend Framework requires that you add `Zend_Loader::loadClass`, which can add to the length of your code, or you may not remember all the classes needed. Listing 3 removes the need to load classes manually.

### Listing 3. Auto-loading classes

```
<?php
session_start();

include 'Zend.php';

function __autoload($class)
{
    require_once 'Zend/Loader.php';
    Zend_Loader::loadClass($class);
}
...
```

The Zend Framework detects this method and loads the necessary classes for you. Also, `session_start()` is added here (one place), so remove other instances of it from the Part 2 code.

## Registering the view

You register the view, which can turn two lines of code into one by adding the view object into the Zend registry. Add to the registry, as shown in Listing 4.

### Listing 4. Adding the Zend view object to the registry

```
...
    Zend_Loader::loadClass($class);
}
}
$baseUrl = str_replace('/index.php', '/', $_SERVER['PHP_SELF']);
Zend_Registry::set('baseUrl', $baseUrl);

$view = new Zend_View;
$view->setScriptPath('../application/views');
$view->baseUrl = $baseUrl;
Zend_Registry::set('view', $view);
...
```

In the register and login actions, retrieve the view object from the registry, as shown in Listing 5.

### Listing 5. Retrieving the view object from the Zend registry

```
public function registerAction()
{
    $view = Zend_Registry::get('view');
    $view->userFunction = 'Create';
    $view->action = '/user/create';
    $view->button = 'Register';
    echo $view->render('register.php');
}

public function loginAction()
{
    $view = Zend_Registry::get('view');
    $view->title = 'Login';
    echo $view->render('login.php');
}
```

---

## Section 3. The class

The functionality the `Zend_Filter_Input` class provides is incredible, and it's something every Zend Framework user should use. This section goes over the specific functionality it provides, then you incorporate its use into the code as it stood at the end of Part 2.

### Functionality provided

With the `Zend_Filter_Input` class, you can load arrays like `$_GET`, `$_POST`, and `$_SESSION`, then access them in much the same way, except you can filter and validate the data.

To do this, you need to define the validation and filter logic for each field you will be processing from the client. The rules are defined in associative arrays, so for example you can specify whether the field only accepts numeric values or alphanumeric, and how long the inputted value should be. Listing 6 shows an example of using `Zend_Filter_Input` to process the month portion of a date.

### Listing 6. Process the month portion of a date using `Zend_Filter_Input`

```
$filters = array(
    'month' => 'Digits',
);

$validators = array(
    'month' => array(
        'Digits',
        new Zend_Validate_Int(),
        array('Between', 1, 12),
    );
);

$input = new Zend_Filter_Input($filters, $validators, $_REQUEST);

$month
if ( $input->isValid('month') ) {
    $month = $input->month;
}
```

You define the validators and filters for the field `$month`, and pass them into `Zend_Filter_Input` along with the input data that comes from the `$_REQUEST` global array. You then can access the properly validated and filtered `$month` value through the `$month` member of the object.

You incorporate the usage of the `Zend_Filter_Input` class into the current code next.

## Revisiting the `IndexController`

You use the input filter in this class to retrieve the username of the current user to verify that a user is currently logged in and either direct that user to the login page or to the `viewFeeds` page. Modify the `IndexController.php` file, as shown in Listing 7.

### Listing 7. Modifying the `IndexController` class

```
<?php

class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $input = new Zend_Filter_Input(
            array('username'=>'StringTrim'),
            array('username'=>'Alpha'),
            $_SESSION);
        $username = $input->getUnescaped('username');
```

```

        if(!isset($username))
            $this->_redirect('user/login');
        else{
            $db = Zend_Registry::get('db');
            $select = $db->select()
                ->from('subscribedfeeds')
                ->joinUsing('feeds', 'feedname')
                ->where('subscribedfeeds.Username = ?', $username);
            $results = $db->fetchAll($select);

            $view = Zend_Registry::get('view');
            $view->username = $username;
            $view->feeds = $results;
            echo $view->render('viewFeeds.php');
        }
    }

    public function noRouteAction()
    {
        $this->_redirect('/');
    }
}
?>

```

You retrieve the `filterSession` object from the registry and retrieve the username from it, as shown in Listing 7. If a user is logged in, you retrieve the feeds from the database he is currently subscribed to. These are then displayed in the `viewFeeds` view, which is covered later in the tutorial.

## Revisiting the createAction

You will now revisit the `createAction`, where you validate the information provided via POST from the register form. Modify the `createAction` method, as shown in Listing 8.

### Listing 8. Modifying the createAction method

```

function createAction()
{
    $input = new Zend_Filter_Input(
        null,
        array(
            'username'=>'Alnum',
            'password'=>'Alnum',
            'password2'=>'Alnum',
            'lName'=>'Alpha',
            'fName'=>'Alpha',
            'email'=>new
Zend_Validate_Regex( "[a-z0-9]+[a-z0-9_-]*(\.[a-z0-9_-]+)*@[a-z0-9_-]+(\.[a-z0-9_-]+)*\.\.[a-z]{2,}$^"
            ),
        $_POST);

    $username = $input->username;
    $password = $input->password;
    $password2 = $input->password2;
    $email = $input->email;
    $lName = $input->lName;
}

```

```

    $fName = $input->fName;

    if($username != '' &&
        $email != '' &&
        $lName != '' &&
        $fName != '' &&
        $password != '' &&
        $password2 != '' &&
        $password == $password2){

        $db = Zend_Registry::get('db');
        $row = array(
            'FirstName' => $fName,
            'LastName' => $lName,
            'EmailAddress' => $email,
            'Username' => $username,
            'Password' => $password
        );

        ...
    }

```

The above code validates the email, username, last name, and first name inputs using the `Zend_Filter_Input` class, rather than accessing directly.

## Revisiting the displayProfileAction

In the `displayProfileAction` method, you'll use the `Zend_Filter_Input` class to retrieve the username. Modify it as shown in Listing 9.

### Listing 9. Modifying the displayProfileAction class

```

function displayProfileAction()
{
    $input = new Zend_Filter_Input(
        array('username'=>'StringTrim'),
        array('username'=>'Alpha'),
        $_SESSION);
    $username = $input->getUnescaped('username');
    if ($username != ''){
        $db = Zend_Registry::get('db');

        ...

        $view->email = $email;
        $view->username = $username;

        ...
    }
}

```

---

## Section 4. Subscribing to feeds

If this was a brokerage website providing feeds for critical financial information, such as the Dow Jones news wire, it's likely you'd charge people before they could

subscribe. In this tutorial, it is assumed that all feeds are free.

This section covers subscribing to and unsubscribing from feeds, as well as viewing all headlines of a given feed.

## The viewFeeds view

This is the main view the `IndexController` sends logged-in users to. It displays available actions and links to view the headlines of available feeds. Define this file and place it in the views directory, as shown in Listing 10.

### Listing 10. The main viewFeeds view

```
<html>
<head>
  <title>You're Logged In!</title>
</head>
<body>
  [<a href="user/logout">Logout</a>]<br>
  [<a href="user/displayProfile">View/Update
  profile</a>]<br>
  [<a href="feed">Sub/Unsubscribe to/from
  Feeds</a>]<br>
  [<a href="feed/viewSavedEntries">View
  Saved
  Entries</a>]<br>
  <h1>CHOMP! The Feed Reader</h1>
  <h3>Hello <?php echo
  $this->escape($this->username); ?>
  you're logged in!</h3>
  ...
</table>
</body>
</html>
```

There are four available actions: Logout, View/Update profile, Subscribe or unsubscribe, and view saved entries (see [Figure 1](#)).

### Figure 1. Viewing the viewFeeds view



## The FeedController

The `FeedController` holds several actions related to feeds. When you click on the "Sub/Unsubscribe to/from Feeds" link, the `indexAction` method of the `FeedController` is called. Define the `FeedController.php` file and place it in the controllers directory, as shown in Listing 11.

### Listing 11. The FeedController

```
<?php
class FeedController extends Zend_Controller_Action
{
    public function indexAction()
    {
```

```

$input = new Zend_Filter_Input(
    array('username'=>'StringTrim'),
    array('username'=>'Alpha'),
    $_SESSION);
$username = $input->getUnescaped('username');

        $db = Zend_Registry::get('db');
$select = $db->select();
$select->from('feeds', '*');
$select->where("feedname NOT IN " .
    "(select feedname from subscribedfeeds ".
    "     where username=?)",
    $username);
$sql = $select->__toString();
$subFeeds = $db->fetchAll($select);

$select = $db->select();
$select->from('subscribedfeeds', '*');
$select->where("username=?", $username);
$sql = $select->__toString();
$unsubFeeds = $db->fetchAll($sql);

$view = Zend_Registry::get('view');
$view->title = 'View Available Feeds';
$view->subFeeds = $subFeeds;
$view->unsubFeeds = $unsubFeeds;
echo $view->render('feeds.php');
}

public function noRouteAction()
{
    $this->_redirect('/');
}
}
?>

```

The code first retrieves the `username` from the `fSession` object stored in the registry, then it retrieves two lists of feeds: those that are subscribed to by this user and those that aren't. The first query to the database retrieves the subscribed-to feeds, which get stored in `subFeeds`, and the next retrieves the feeds that aren't currently subscribed to and stores them in `unsubFeeds`. The view is then retrieved from the registry, initialized, rendered, and displayed.

## The feeds view

The rendered view from Listing 12 is `feeds.php`. This view displays the subscribed-to feeds with links to unsubscribe, and it shows a list of feeds that are unsubscribed to with links to subscribe. Define the `feeds.php` file, as shown below.

### Listing 12. Defining the feeds view

```

<html>
<head>
    <title><?php echo $this->escape($this->title);
    ?></title>
</head>
<body>

```





You define the `subscribeAction` method next; then you'll define the `unsubscribeAction` method.

## Subscribing to feeds

When a user wants to subscribe to a feed, the `subscribeAction` is called. The action needed is to add the feed link to the database, along with the username that subscribed to it. Define the `subscribeAction` in the `FeedController` class, as shown in Listing 13.

### Listing 13. Defining the `subscribeAction` method

```
public function subscribeAction()
{
    $input = new Zend_Filter_Input(
        array('username'=>'StringTrim'),
        array('username'=>'Alpha'),
        $_SESSION);
    $username = $input->getUnescaped('username');

    $input = new Zend_Filter_Input(
        array('*'=>'StringTrim'),
        null,
        $_GET);
    $feedTitle = $input->getUnescaped('title');

    $db = Zend_Registry::get('db');

    $row = array(
        'Username' => $username,
        'feedname' => $feedTitle
    );

    $table = 'subscribedfeeds';
    $rowsAffected = $db->insert($table, $row);

    $this->_redirect('/feed');
}
```

This retrieves the `username` and the `feedTitle` from the `fSession` and `fGet` objects in the Zend registry, respectively. A SQL statement is then prepared that inserts a row into the `subscribedfeeds` table, subscribing the user to his desired feed. See [Figure 3](#) for the example browser after subscribing to Yahoo News.

### Figure 3. Subscribing to Yahoo News



## Unsubscribing from feeds

Unsubscribing from a feed is the reverse of subscribing to it. Define the `unsubscribeAction` method of the `FeedController`, as shown in [Listing 16](#).

### Listing 16. Defining the `unsubscribeAction` method

```
public function unsubscribeAction()
{
    $input = new Zend_Filter_Input(
        array('username'=>'StringTrim'),
        array('username'=>'Alpha'),
        $_SESSION);
    $username = $input->getUnescaped('username');
```

```

    $input = new Zend_Filter_Input(
        array('*'=>'StringTrim'),
        null,
        $_GET);
    $feedTitle = $input->getUnescaped('title');

    $db = Zend_Registry::get('db');
    $table = 'subscribedfeeds';
    $where = "username='$username' and feedname='$feedTitle'";
    $rowsAffected = $db->delete($table, $where);

    $this->_redirect('/feed');
}

```

Here, the `username` and `feedTitle` are retrieved from the `fSession` and `fGet` objects in the Zend registry, respectively, just as for subscribing to feeds. Then a SQL statement is prepared that deletes the record from the `subscribedfeeds` table that matches the given `username` and `feedTitle`. In the next section, you'll cover viewing available feeds. Unsubscribing from Yahoo News will bring you back to Figure 3.

---

## Section 5. Viewing available feeds

Now that users can subscribe to feeds, they should be able to view the feeds they've subscribed to, as well as the headlines in each. This section completes the `viewFeeds` view, viewing headlines in a feed, and the `viewChannel` view.

### Completing the `viewFeeds` view

In Listing 12, you started the `viewFeeds` view, and now you'll complete it, allowing you to view the feeds currently available to you. Finish defining the `viewFeeds.php` file, as shown in Listing 17.

#### Listing 17. Finishing the `viewFeeds` view

```

<html>
<head>
  <title>You're Logged In!</title>
</head>
<body>
  ...
  <h3>Hello <?php echo $this->escape($this->username); ?>
    you're logged in!</h3>
  Subscribed Feeds:<br>
  <?php
    $ct = 0;
    $feeds = $this->feeds;
    foreach($feeds as $row){

```

```
        $feedTitle = $row['feedname'];  
        echo "<a href='feed/viewChannel?title=$feedTitle'>".  
            "$feedTitle</a><br>";  
    }  
    ?>  
</table>  
</body>  
</html>
```

Here, you iterate over the available feeds and display a title, and add a link to view the latest headlines within. Note that the name of the action is the `viewChannel` action in the `FeedController` class. Example browser output of the completed `viewFeeds` view is shown in Figure 4.

**Figure 4. The finished viewFeeds view**



## Viewing the latest feeds in a channel

When a user clicks the link to view a feed, he is brought to a new action, the `viewChannel` action. Define this action in the `FeedController` class, as shown in Listing 18.

### Listing 18. Defining the `viewChannelAction` method

```
public function viewChannelAction()
{
    $input = new Zend_Filter_Input(
        array('*'=>'StringTrim'),
        null,
        $_GET);
    $feedTitle = $input->getUnescaped('title');

    $db = Zend_Registry::get('db');
    $select = $db->select();
    $select->from('feeds', '*');
    $sql = $select->__toString();
    $results = $db->fetchAll($sql);

    $feedLink = $results[0]['link'];
    $rssFeed = Zend_Feed::import($feedLink);

    $view = Zend_Registry::get('view');
    $view->title = $feedTitle;
    $view->rssFeed = $rssFeed;
    echo $view->render('viewChannel.php');
}
```

This method retrieves the name of the feed the user has desired to view headlines for from the `fGet` object in the Zend registry. The link is retrieved from the database, and the RSS feed data is retrieved from the link, which is passed onto the view object, which is also where the `viewChannel` will retrieve its headline and link data from.

## The `viewChannel` view

Now that you've retrieved the RSS feed data, you will use it in the `viewChannel` view and display the links to the user. Define the `viewChannel` view, as shown in Listing 19.

### Listing 19. Defining the `viewChannel` view

```
<html>
<head>
    <title><?php echo $this->escape($this->title);
    ?></title>
</head>
<body>
    [<a href='/'>Back to Main Menu</a>]<br>
```



## Section 6. Saving, viewing, and deleting feed entries

With links available to save entries to the database, you need to define the appropriate action, as well as provide functionality to view and subsequently delete them when no longer needed.

### Saving entries

Saving entries allows a user to view them later. Recall that the link to save entries to the database was covered in Listing 19, and now you'll define the action to facilitate this functionality. Define the `saveEntryAction` method in the `FeedController` class, as shown in Listing 20.

#### Listing 20. Defining the `saveEntryAction` method

```
public function saveEntryAction()
{
    $input = new Zend_Filter_Input(
        array('username'=>'StringTrim'),
        array('username'=>'Alpha'),
        $_SESSION);
    $username = $input->getUnescaped('username');

    $input = new Zend_Filter_Input(
        array('*'=>'StringTrim'),
        null,
        $_GET);
    $feedTitle = $input->getUnescaped('feedTitle');
    $channelTitle = $input->getUnescaped('title');
    $channelLink = $input->getUnescaped('link');

    $db = Zend_Registry::get('db');

    $row = array(
        'Username' => $username,
        'feedname' => $feedTitle,
        'channelname' => $channelTitle,
        'link' => $channelLink
    );

    $table = 'savedentries';
    $rowsAffected = $db->insert($table, $row);

    $this->_redirect\
        ("/feed/viewChannel?title=$feedTitle");
}
```

The above code retrieves the `username` from the `fSession` object in the Zend registry, and the `feedTitle`, `channelTitle` (headline title) and `channelLink` (link to full article) from the `fGet` object in the Zend registry. These four fields are then saved in a row of the `savedentries` table in the database. After saving the

data to the database, the user is redirected to the `viewChannel` action with the same `feedTitle` set in the URL.

## Viewing saved entries

Now that entries can be saved in the database, you need to define the `viewSavedEntries` action. Recall that you provided this link in the `viewFeeds` view in Listing 12. Define the `viewSavedEntriesAction` method in the `FeedController` class, as shown in Listing 21.

### Listing 21. Defining the `viewSavedEntriesAction` method

```
public function viewSavedEntriesAction()
{
    $input = new Zend_Filter_Input(
        array('username'=>'StringTrim'),
        array('username'=>'Alpha'),
        $_SESSION);
    $username = $input->getUnescaped('username');

    $filterSession = Zend_Registry::get('fSession');
    $username = $filterSession->getRaw('username');

    $db = Zend::registry('db');
    $select = $db->select();
    $select->from('savedentries', '*');
    $select->where("username=?", $username);
    $sql = $select->__toString();
    $entries = $db->fetchAll($sql);

    $view = Zend_Registry::get('view');
    $view->title = 'View Saved Entries';
    $view->entries = $entries;
    echo $view->render('viewSavedEntries.php');
}
```

This action retrieves the `username` from the `fSession` object in the Zend registry, then sets up a query to retrieve all the feeds this user is subscribed to and tunnels this into the `viewSavedEntries` view.

## The `viewSavedEntries` view

This is the view rendered in Listing 21, which displays all the saved entries to the user. Define the `viewSavedEntriesAction` method in the `FeedController` class, as shown in Listing 22.

### Listing 22. Defining the `viewSavedEntries` view

```
<html>
<head>
  <title><?php echo $this->escape($this->title);
```



```
$input = new Zend_Filter_Input(
    array('username'=>'StringTrim'),
    array('username'=>'Alpha'),
    $_SESSION);
$username = $input->getUnescaped('username');

$input = new Zend_Filter_Input(
    array('*'=>'StringTrim'),
    null,
    $_GET);
$feedTitle = $input->getUnescaped('feedTitle');
$channelTitle = $input->getUnescaped('channelTitle');

$db = Zend_Registry::get('db');
$table = 'savedentries';
$where = "username='$username' and feedname='$feedTitle'".
        " and channelname='$channelTitle'";
$rowsAffected = $db->delete($table, $where);

$this->_redirect('/feed/viewSavedEntries/');
}
```

The above code retrieves the username from the `fSession` object in the Zend registry, and the `feedTitle`, `channelTitle` (headline title) and `channelLink` (link to full article) from the `fGet` object in the Zend registry, just as in Listing 20. Example browser output after deleting an entry is shown in Figure 7.

### Figure 7. Deleting an entry

[\[Back to Main Menu\]](#)

## View Saved Entries

[Feed > Title \(Click to View\)](#)    [Delete Channel Entry](#)

That completes the feed reader! In the next section, you define the last couple of management tasks.

---

## Section 7. Management actions

There are a couple of final management tasks that users can do to view or update their accounts and log out.

### Updating an account

When a user updates an account, you can see from Listing 10 that the username is retrieved from the `SESSION` array, so this should not be changeable. To make it unchangeable in the register view, define it as shown in Listing 24.

#### Listing 24. Modifying register.php

```
...
<p>Username:
  <?php
    if($this->button == 'Update')
      echo $this->username .
        '<input name=\
          "username" type="hidden" value="'.
          $this->username . '>';
    else
      echo '<input name="username" value="'.
        $this->username . '>';
  ?>
...
```

Thus, if the user is updating his account, the username is shown, the input tag is hidden, and its value is set to the logged-in user's username. Otherwise, an empty text box is shown. The modifications can be seen in Figure 8.

#### Figure 8. Viewing the register view in update/view profile mode

[\[Back to Main Menu\]](#)

# Update a User Account

First Name:

Last Name:

Email Address:

Username: jmertic

Password:

Confirm Password:

Another great example using the same view for two purposes with the Zend Framework.

## Logging out

Logging out destroys the current session. Define the `logoutAction` method in the `UserController`, as shown in Listing 25.

### Listing 25. Logging out

```
public function logoutAction()
{
    session_destroy();
    $this->_redirect('/');
}
```

This destroys the current session and sends the user back to the root.

Congratulations! You're all done.

---

## Section 8. Summary

Along with your newfound MVC skills and Zend database skills, you've become more comfortable with the MVC style with the added `FeedController` and associated views. You've also mastered the `Zend_Feed` class by retrieving RSS feeds from the Internet. Additionally, you've learned how to use helper functions of the `Zend_Filter_Input` class to help validate input data easily.

[Part 4](#) shows where you'll use the `Zend_HTTPClient` class to retrieve and store entire articles (feed entries) in the database, and add subscribable websites that don't support RSS feeds into the feed-reader interface, whose contents will also be optionally savable into the database using the `Zend_HTTPClient`.

## Downloads

Description	Name	Size	Download method
Part 3 source code	os-php-zend3.source.zip	11KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- The tutorial series "[Learning PHP](#)" takes you from the most basic PHP script to working with databases and streaming from the file system.
- The [PHP Function Reference](#) is another valuable resource.
- The [Zend Framework](#) website maintains the [latest documentation](#).
- 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.
- To better understand the goals of the feed-reader project, read "[Introduction to Syndication, \(RSS\) Really Simple Syndication](#)," by Vincent Luria.
- [Thought Storms ModelViewController](#) explains the MVC and the controversy and confusion surrounding it.
- The phpPatterns website explains [MVC from the PHP point of view](#).
- [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 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 author

### Tyler Anderson

Tyler Anderson received his bachelor's degree in computer science in 2004 and his master's degree in electrical and computer engineering in 2005 from Brigham Young University. He worked with Stexar Corp. as a design engineer, R&D, from May 2005 to August 2006. Since his discovery by Backstop Media LLC in early 2005, he has written and coded numerous articles and tutorials for IBM developerWorks and DevX.com.