Cook up Web sites fast with CakePHP, Part 4: Use CakePHP's Session and Request Handler components

Streamline PHP applications

Skill Level: Intermediate

Sean Kelly (skelly@idsociety.com)
Web Application Developer
ID Society

02 Jan 2007

Updated 22 Jan 2008

CakePHP is a stable production-ready, rapid-development aid for building Web sites in PHP. This "Cook up Web sites fast with CakePHP" series shows you how to build an online product catalog using CakePHP.

Section 1. Before you start

Editor's note: This series was originally published in 2006 and 2007. Since its publication, CakePHP developers made significant changes to CakePHP, which made this series obsolete. In response to these changes and the popularity of this series, the authors revised each of its five parts to make it compliant with the version of CakePHP available in January 2008.

This "Cook up Web sites fast with CakePHP" series is designed for PHP application developers who want to start using CakePHP to make their lives easier. In the end, you will have learned how to install and configure CakePHP, the basics of Model-View-Controller (MVC) design, how to validate user data in CakePHP, how to use CakePHP helpers, and how to get an application up and running quickly using CakePHP. It might sound like a lot to learn, but don't worry — CakePHP does most of it for you.

About this series

- Part 1 focuses on getting CakePHP up and running, and the basics of how to put together a simple application allowing users to register for an account and log in to the application.
- Part 2 demonstrates how to use scaffolding and Bake to get a jump-start on your application, and using CakePHP's access control lists (ACLs).
- Part 3 shows how to use Sanitize, a handy CakePHP class, which helps secure an application by cleaning up user-submitted data. It also covers the CakePHP security component, handling invalid requests, and other advanced request authentication.
- Part 4 focuses primarily on the Session component of CakePHP, demonstrating three ways to save session data, as well as the Request Handler component to help you manage multiple types of requests (mobile browsers, requests containing XML or HTML, etc.).
- Part 5 deals with caching, specifically view and layout caching, which can help reduce server resource consumption and speed up your application.

About this tutorial

There are multiple ways of saving session data using CakePHP's Session component, and each method has its advantages. In this tutorial, you'll learn how to use the Session component by incorporating all three ways into your application, so you'll be able to pick the best one that works for you. On top of that, you'll learn how to use the Request Handler component to aid in your handling of various HTTP requests, including requests from mobile browsers, or requests containing XML or HTML content.

This tutorial is divided into two main topics:

- The different types of session handling covered by CakePHP — You will learn the advantages and disadvantages of each, and how to implement them.
- How to use the Request Handler in your controllers — We will use it for two purposes: to add an RSS feed of your products and to implement Ajax functionality.

Prerequisites

This tutorial assumes you have already completed Part 1, Part 2, and Part 3, and that you still have the working environment you set up for those tutorials. If you do not have CakePHP installed, you should run through Parts 1 and 2 before continuing.

It is assumed that you are familiar with the PHP programming language, have a fundamental grasp of database design, and are comfortable getting your hands dirty.

For the section on Ajax, it is also assumed that you have a basic understanding of Ajax. See Resources for links to help you get started with Ajax.

System requirements

Before you begin, you need to have an environment in which you can work. CakePHP has reasonably minimal server requirements:

1. An HTTP server that supports sessions (and preferably `mod_rewrite`). This tutorial was written using Apache V2.2.4 with `mod_rewrite` enabled.
2. PHP V4.3.2 or later (including PHP V5). This tutorial was written using PHP V5.2.3
3. A supported database engine. this tutorial was written using MySQL V5.0.4

You'll also need a database ready for your application to use. The tutorial will provide syntax for creating any necessary tables in MySQL.

The simplest way to download CakePHP is to visit CakeForge.org and download the latest stable version. This tutorial was written using V1.2.0. Nightly builds and copies straight from Subversion are also available. Details are in the CakePHP Manual (see Resources).

Section 2. Tor, so far

Up to now, you have used CakePHP to create a simple application for managing products and dealers. In Part 1, you gained an understanding of the MVC paradigm. In Part 2, you used the powerful component of scaffolding to easily develop a framework for your application. You finished Part 3 with a number of projects to improve Tor. The first was to sanitize your data.

What to sanitize

When you sanitized your data, you probably noticed that most of the user input so far is fairly simple. Most of the data input can be filtered using the `paranoid` method, since you should not get anything too complex from the user. The login action of the users controller is shown below.

Listing 1. Sanitizing the user name input

```
function login()
{
   if ($this->data)
      {
        $results = $this->User->findByUsername(Sanitize::paranoid($this->data
                                        ['User']['username']));
        if ($results && $results['User']['password'] == md5($this->data
                                        ['User']['password']))
        {
        $this->Session->write('user', $this->data['User']['username']);
        $this->redirect(array('action' => 'index'), null, true);
        } else {
        $this->set('error', true);
        }
    }
 }
```

Similarly for registration, you would expect that a user's name should only contain letters, spaces, hyphens, and apostrophes. However, apostrophes and hyphens can be bad news for a SQL database. Using the `sql` method ensures that these reach the database safely.

Listing 2. You should sanitize different inputs based on their expected values

```
function register()
{
   if (!empty($this->data))
      {
        $this->data['User']['username'] = Sanitize::paranoid($this->data
                        ['User']['username']);
        $this->data['User']['email'] = Sanitize::paranoid($this->data
                        ['User']['email'], array('@', '.', '-', '+'));
        $this->data['User']['first_name'] = Sanitize::sql($this->data
                        ['User']['first_name']);
        $this->data['User']['last_name'] = Sanitize::sql($this->data
                        ['User']['last_name']);
        $this->data['User']['password'] = md5($this->data
                        ['User']['password']);
```

These are just a couple of examples of how you could have sanitized your data.

Securing the application

You next task was to secure Tor using the Security component. If you consider what forms need to have the most security, a natural guess would be any forms which change the database. This is a good rule of thumb: If a change is made to the database, the form should be submitted via the `POST` method.

One action that fits this description is the product `delete` action. Since deleting a product removes a row from the database, it should be a `POST`-only request. The code to require this is shown below.

Listing 3. ProductsController requiring POST for delete action

```
function beforeFilter()
{
  $this->Security->requireAuth('delete');
  $this->Security->requirePost('delete');
}
```

If you try deleting a product now, you will notice that you get a 400 error. This is good because it means that any delete has to occur because of a particularly formatted request. However, we need to get the delete functionality back, so you'll need to make an appropriate change on your views that point to the `delete` action. See Listing 4 for how the new `showDelete` section could look.

Listing 4. The delete action calls should happen inside forms

```
<?php if ($showDelete) { ?>

        <?php echo $form->create('Product', array('action' => 'delete/' .
               $product['Product']['id']));?>

        <li><?php echo $form->end('Delete product');?></li>

<?php } ?>
```

You should also require that the registration and login forms be `POST` requests. The code will be similar to what we have implemented here.

Giving feedback for invalid requests

Your final task was to use the `blackHoleCallback` to provide the user with an appropriate response from an invalid request. Its purpose is to provide some helpful and friendly feedback, instead of a nasty server error.

One possible implementation for this code is provided below.

Listing 5. Friendlier information for a bad request function beforeFilter()

```
{
  $this->Security->requireAuth('delete');
  $this->Security->requirePost('delete');
  $this->Security->blackHoleCallback='invalid';
}

function invalid() {
  header('HTTP/x 400 Bad Request');
  echo('<h1>Tor</h1>');
  echo('<p>We\'re sorry - there has been a problem processing your request.
Please try submitting the form again.</p>');
  die;
}
```

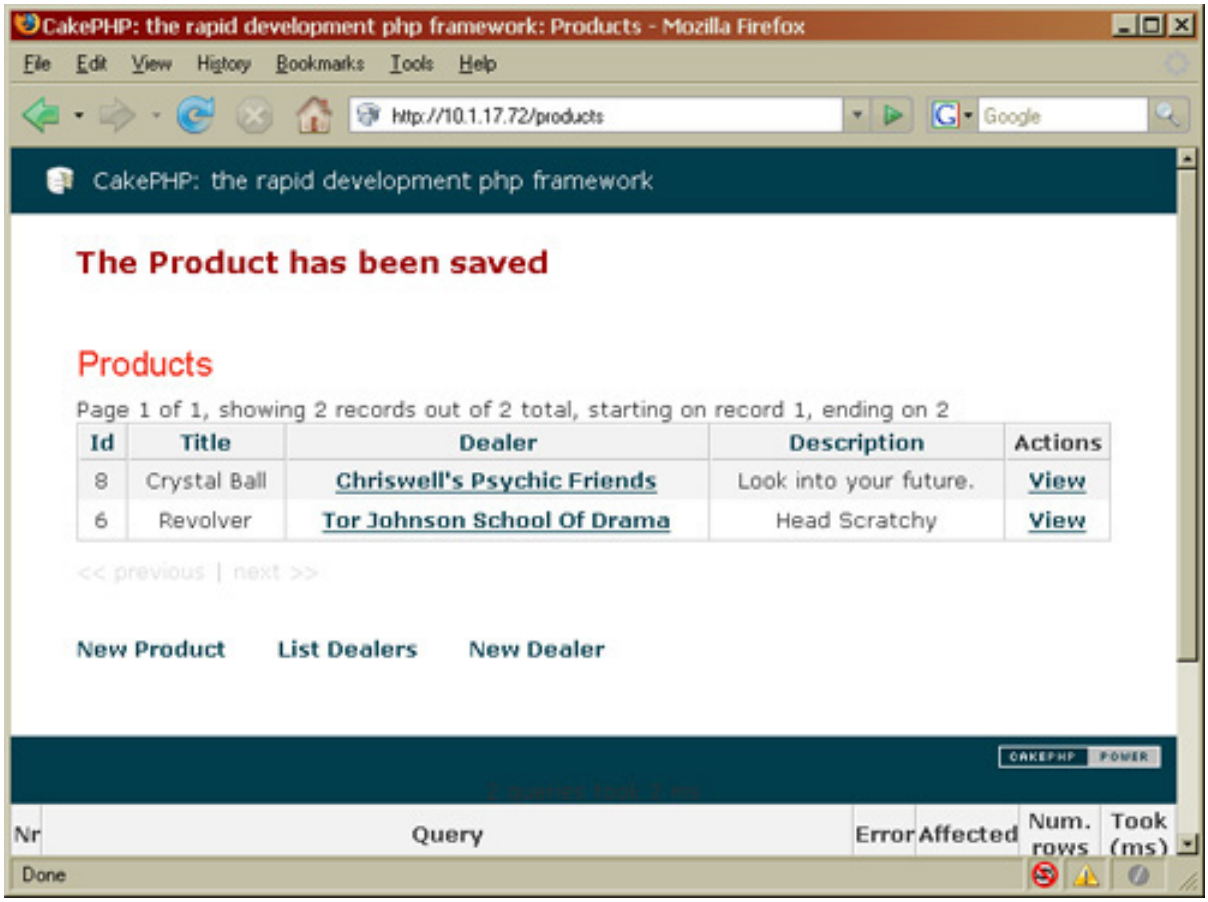
Changing the default layout

You will be adding a favorite products feature to Tor. The visitors to your site should be able to save products for later, or use an RSS feed to keep track of new products

as they become available. To add this functionality, you will be dealing with the session-handling function and Request Handler component. However, before you can begin, you need to expand Tor a bit to give it a place for its new features. To this end, you will change the default layout to customize the look and feel of Tor.

When you created the products controller for Tor, you used the Bake script to create the application framework. The end result was a generic page with a single table on it that looked like Figure 1.

Figure 1. Plain scaffolding



While this scaffolding is good for solidifying your application design, you are now at the point where users are going to need more functionality than simply viewing products. Since the default layout provided a good start, all you have to do is update the existing views a bit to organize the application.

Layouts

Layouts are a kind of view that do not correspond directly with a controller or model. That is, a layout is simply a template. Changing the default layout affects the area surrounding all of the other views and would commonly be used for things like headers, footers, and menu bars.

We have been utilizing the default CakePHP layout, which lives in cake/libs/view/templates/layouts/default.ctp (don't change it!). To override the layout

with our own custom template, copy that default template to app/views/layouts/default.ctp and modify until it looks like Listing 6 (nothing after the menu div changes from the default template).

Listing 6. Overriding the layout with our own custom template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>
                Tor : <?php echo $title_for_layout;?>
        </title>

        <?php echo $html->charset();?>
        <link rel="icon" href="<?php echo $this->webroot;?>favicon.ico"
                              type="image/x-icon" />
        <link rel="shortcut icon" href="<?php echo $this->webroot;?>
                              favicon.ico" type="image/x-icon" />
        <?php echo $html->css('cake.generic');?>
        <?php echo $scripts_for_layout;?>
</head>

<body>
        <div id="container">
                <div id="header">
                     <h1><?php echo $html->link('Tor', '/'); ?>
                     : Welcome <?php echo $session->read('user') ?></h1>
                </div>
                <div id="content">
                <div id="menu">
                <?php echo $html->link('Products',
                                       array('controller' => 'products')); ?> | <?php echo
                                       $html->link('Favorites', array('controller' =>
'users',
                                       'action' => 'favorites')); ?> |

                <?php echo $html->link('Login', array('controller' =>
                                 'users', 'action' => 'login')); ?> |
                <?php echo $html->link('Logout', array('controller' =>
                                  'users', 'action' => 'logout')); ?>
                </div>
...
(message flash, content for layour, cake link, debug, etc
...
</body>
</html>
```

This is code is based on the default layout for CakePHP, and simply adds a menu bar and adds some Tor branding. Since it still uses the default CSS, the site will not change very much visually.

Notice that the layout is XHTML Transitional. Even though all of our views are in XHTML, it is entirely possible that they be plugged into a layout in another format, such as Atom, WML, or XHTML-Mobile. We will return to the idea of layouts when we add this functionality to the site.

Section 3. Session handling

When a request is made for a Web site, it is entirely independent of every request before it. To implement user functionality, a particular Web client needs repeated access to a unique, possibly secret, piece of information. This is called *session handling* in Web applications, and the subtleties of properly handling sessions is a headache to many developers.

Session-handling basics

The basic idea is the same across any programming language. Upon first visiting the Web site, the client is given a unique ID, generally stored as a cookie. On the server side, an array of variables is stored, in some way, which corresponds to that unique ID. For any subsequent request by the same client, the unique ID is sent along with the request, and the server loads the array into memory. These variables are called the *session variables* for the client. In PHP, they are stored in the `$_SESSION` global variable.

You could simply use the `$_SESSION` global variable to access session information when using CakePHP. However, CakePHP comes with its own session-handling object, which, as we will see later, has a number of benefits.

Session handling within Tor

You have already implemented sessions without much effort. When the user logs in, the `UsersController` object stores the user name and associates it automatically with the client during subsequent requests.

Every `Controller` object (anything inheriting from `AppController`) automatically has access to the Session component. Recall that a `Component` is a class associated with a controller that allows extra functionality, much like a helper provides extra functionality to a view. You used the `Acl` component in Part 2 when you added ACLs. Recall that you needed to add the following line to the beginning of the `ProductsController` and `UsersController` classes: `var $components = array('Acl');`.

Though session handling is a component in CakePHP, it is a standard component included with every controller, so there is no need to add it to this list.

To use the Session component, simply access the session instance variable in the controller. The two most important functions of the session handler are setting and getting variables, which are achieved by calling the `read` and `write` methods. For user login, you used the code in Listing 7 to write the current client's user name to the session. This code is located in the `UsersController` class.

Listing 7. Using the write method to store the user name

```
function login()
{
```

```
  ...
  $this->Session->write('user', $this->data['User']['username']);
  ...
{
```

The `write` method takes two parameters. The first is the key being assigned, and the second is the value to assign it. In this case, the key is `user`, and the value is the user name entered (see Listing 8). The user name is later accessed to test whether a product can be viewed or edited based on the ACL.

Listing 8. Using the read method to access the user name

```
function view($id) {
  ...
  if ($this->Acl->check($this->Session->read('user'),
$id.'-'.$product['Product']['title'], 'read'))
  {
    ...
  }
}
```

The `read` method takes a single parameter — the key you wish to access — and returns the value that was stored.

Storing sessions

All this should be no news to Web developers, but session handling is generally only a simple matter for small applications. If an application requires sensitive user information to be stored in session, such as passwords or credit card information, where and how the data is stored is vitally important.

Session storing is controlled by a single line in CakePHP: `Configure::write('Session.save', 'php');`. This line, located in the /app/config/core.php file, tells Cake how sessions should be stored. There are three valid values: `php` (use whatever the php.ini file says), `cake` (save session files in Cake's tmp directory), and `database` (use Cake's database sessions).

Storing sessions: cake

When this value is set, sessions are stored as files within your application folder. CakePHP provides a folder at /app/tmp/session with files that look something like sess_50bfa744a2ab2c98df808f70c893704c. Within these files, individual session variables are stored as plain text without encryption.

The Apache Web server has a setting called safe mode that restricts the folders Apache has access to. Generally, Apache can only read anything within the docroot of the site, with some exceptions regarding library files. In this series, you have installed CakePHP by dropping the entire directory into Apache's DocumentRoot

directory (in this case, the /webroot directory). That's certainly an easy way to install the framework, but it's not the most secure, and this is one reason why. By setting session handling to `cake`, you are storing session variables in a directory that could be accessed directly by the browser. If your site is compromised in such a way that an attacker can cause Apache to return arbitrary files, every user's session data will be vulnerable (though that might be the least of your worries). Cake does a pretty good job of protecting these files even if everything is in the DocumentRoot, but ideally, you should install more securely by making the app/webroot directory your DocumentRoot directory. This would keep the Cake session files offline.

It is important to note that you cannot control the permissions on these session files. They will have the exact same user and group permissions PHP was granted. On some systems, this may be a security vulnerability, as well.

Advantages:

- Session variables are stored within Cake, and, thus, the entire application stays in one place.
- Session files can be read with a text browser, possibly for debugging (unlikely useful).
- Starting and accessing a session do not require a database connection (unlikely helpful, since accessing models in CakePHP initiates a PHP session).

Disadvantages:

- Any files stored in DocumentRoot can be compromised if the Web server is compromised.
- Session files have the same permissions PHP is given.
- Load-balanced Web servers not sharing a file system cannot share access to session files, causing sessions to be mysteriously dropped unless using sticky sessions.

Storing sessions: database

If the session information you are storing needs a higher level of security or a greater control over permissions, database sessions are better. By setting `Session_save` to `database`, you're telling CakePHP to store all serialized variable information on a table in the database with the rest of your application.

Before you can use database sessions with CakePHP, you must create the table. Feel free to give it a try with *Tor*. By default, the name of the table is cake_sessions, although this can be changed in the file app/config/core.php. You will need to uncomment the `Configure::write` lines for Session.table and Session.database in order to use database session storage. The schema for this table is stored in app/config/sql/sessions.sql. The schema included with CakePHP V1.2.0.x is shown

below.

Listing 9. Create table cake_sessions

```
CREATE TABLE cake_sessions (
  id varchar(255) NOT NULL default '',
  data text,
  expires int(11) default NULL,
  PRIMARY KEY  (id)
);
```

After you've created the table and set the `Session.save` to `database`, try logging into Tor at /users/login. After you log in, check the database to see if your session appeared. There should be a single row that looks something like this:

```
id                                  data                                expires
50bfa744a2ab2c98df808f70c893704c    Config|a:3:{s:4:"rand";i:94427...   1164661678
```

Storing sessions in the database directly addresses all three disadvantages listed. First, Apache generally has no direct access to the database, so compromising the Web server doesn't expose the session files. Second, permissions on the database can be explicitly set for your application, and you can even restrict access to a particular host. Third, load-balanced servers have shared access to the session table with no extra work on your part.

For most large applications, database sessions are essential because they allow the greatest amount of security with the least amount of effort.

Advantages:

- Simple to set up — only requires one extra table in your database.
- A loss of security in the Web server will probably not result in sessions being compromised.
- Sessions can be more easily shared across load-balanced servers.

Disadvantages:

- Using the database to store sessions adds some database overhead. That can add up.
- Sessions are still stored in plain text on the database; database backups may cause sensitive data to be stored for prolonged periods of time.
- Depending on how your database is set up, communications between your application and the database may not be secure. If your database isn't on localhost, or isn't over a secure channel, such as a VPN, it is possible that the communications can become compromised.

Storing sessions: php

The final method for storing sessions is to use whatever session handling PHP is set up to use. By default, PHP will write its sessions as files similar to the `cake` setting for `Session.save`. One main difference is that instead of saving session variables within the Cake application, they are generally stored in a temporary directory elsewhere on the file system. This may or may not be preferable to having sessions stored in the same directory as your site.

By setting sessions to be stored via PHP, you are giving more control over sessions to PHP, rather than to Cake. PHP allows you to override a number of session-handler functions, essentially overriding the way sessions are stored in a method of your choosing. That could be storing sessions in a separate database, over a custom secure channel, or whatever crazy method you can come up with.

If you need to change the way session handling is done in PHP, there is a single function, `session_set_save_handler`, that controls the session-callback function. Passing this function the names of function that open, close, read, and write to the session is demonstrated below. An in-depth discussion of the `session_set_save_handler` function is outside the scope of this article (see Resources).

Listing 10. Redefining how PHP stores sessions

```
function open($save_path, $session_name)
{
  global $sess_save_path;

  $sess_save_path = $save_path;
  return(true);
}

function read($id)
{
  global $sess_save_path;

  $sess_file = "$sess_save_path/sess_$id";
  return (string) @file_get_contents($sess_file);
}

...

session_set_save_handler("open", "close", "read", "write", "destroy", "gc");
```

The session-handling functions are redefined by the PHP functions: `session_start()`. Using this method has some advantages.

Advantages:

- It's flexible. Any storage method supported by PHP.
- If you don't override the session-handling functions, sessions will be stored the same way as all the other applications on your server.

Disadvantages:

- Since PHP is set up to store sessions in the tmp directory by default, you could face some of the same disadvantages described above in "Storing sessions: `'cake'`."

Deciding what method to use for handling session storage isn't always cut and dry. Most of the time, letting PHP handle your sessions will be fine. But ultimately, you will need to make that decision based on the particulars of your application.

In the next section, we discuss the Request Handler component and how you can use it to add Ajax functionality to your application.

Section 4. Utilizing the Request Handler

CakePHP comes with a Request Handler that allows you to present your application in different ways, depending on the type of request being made. In the previous sections, you saw how layouts controlled the wrapper of your application, which, by default, in CakePHP is XHTML Transitional. Here, we'll examine how you can use the Request Handler to return data from your application in a different form from XHTML, depending on what is requesting the page.

Anatomy of an HTTP request

Every element on a Web page is obtained via an HTTP request, a text header that contains the resource being requested, and information on the requesting agent. A sample HTTP request is given below.

Listing 11. Sample HTTP request

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.8.1)
Gecko/20061010 Firefox/2.0
Accept: text/xml,application/xhtml+xml,text/html
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive: 300
Connection: keep-alive
Cache-Control: max-age=0
```

In Web sites of yore, it wasn't uncommon to try and parse the `User-Agent` string, providing completely different content to different Web browsers. For example, a company that designed its site for Internet Explorer® might find it does not look correct in other browsers, and suggest these visitors use IE.

Luckily, those days are coming to an end, and good manners on the Web mean every Web browser should get the exact same content for the exact same resources. Note that the same content does not necessarily mean it has to be presented the same way.

Instead of using the `User-Agent` to determine how to format content, a better method is to use the `Accept` header. In the example above, Firefox accepts three types of content identified by their MIME types: XML, XHTML+XML, and HTML.

Creating an RSS feed

Now you will create an RSS feed for Tor, which is an updated list of all new products. This has changed dramatically since CakePHP V1.1.8 — now it's even easier. You need to include the Request Handler component in the Products handler, create a view that uses the RSS Helper, and tell the Router to parse RSS extensions.

Giving the product list this behavior is easy when you use the Request Handler component. Since your feed will be located at /products, open the file containing the class `ProductsController`. To begin, include the Request Handler component.

Listing 12. Including the Request Handler component

```
<?php
class ProductsController extends AppController
{

  var $name = 'Products';
  var $helpers = array('Html', 'Form' );
  var $components = array('Acl', 'Security', 'RequestHandler');
  ...

}
```

There is a lot more to the Request Handler than what we're going to cover here. You can use it to determine what kind of requests are accepted, or preferred, and much more. But for now, simply including the handler will suffice.

Next, you need to create a view just for the RSS feed. This view will be automagically rendered when required and makes use of the RSS Helper to get things done. Create the file app/views/products/rss/index.ctp (you may need to create that RSS directory). The view should look like what is shown below.

Listing 13. The products index RSS view

```
<?php
     echo $rss->items($products, 'transformRSS');
     function transformRSS($products) {
           return array(
                'title'  => $products['Product']['title'],
                 'link'    => array('url'=>'/products/view/'
                                    .$products['Product']['id']),
                        'description'   => $products['Product']['description'],
```

```
                );
        }
?>
```

Finally, add the following line to the app/config/routes.php file: `Router::parseExtensions('rss');`.

That's it! You can view the new RSS feed at http://localhost/products/index.rss. You may need to view the source to read the XML. And if your debug level is set to 2 or 3, the debug info will certainly invalidate the XML format. But this should get you well on the road to playing with RSS in Cake.

Adding Ajax

Asynchronous JavaScript and XML (Ajax) is a popular method for creating interactive Web applications without sacrificing browser compatibility. In this section, we describe how to use CakePHP's Ajax helper and Request Handler components to streamline the updating of products. It is assumed that you have a general familiarity with Ajax concepts. If you need an introduction, try reading "Mastering Ajax" and "Considering Ajax" (see Resources).

Recall that Ajax is a series of asynchronous requests from the client to the server, generally initiated by user interaction. Since CakePHP separates the back-end and front-end plug-ins into components and helpers, respectively, it means that all of your JavaScript functions are going to be implemented by the Ajax helper, while all of the back-end functionality will be supported by the Request Handler component.

The Request Handler component effectively allows you to reuse the non-Ajax code you created. It has an `isajax` method, which returns true if the request was made by an `XMLHttpRequest` call. Within your controller, you can use this method to check the nature of the request and tweak the output slightly, depending on whether Ajax is involved.

script.aculo.us

CakePHP's Ajax helper requires Prototype for Ajax and uses script.aculo.us for effects. Both are released under the MIT license. (The MIT license is the same as CakePHP and basically says you can use the software for any means with few restrictions). script.aculo.us is distributed with a copy of Prototype, so there is no need to download both. You should begin by downloading the latest copy of script.aculo.us (see Resources). You don't really need to know much about script.aculo.us, except that you need to include its JavaScript files on your site.

After you've downloaded and extracted script.aculo.us, you should see a folder called src with a number of JavaScript files in it. You won't be using all of them, but you might as well copy all of them into the app/webroot/js folder of your site. Also copy the file lib/prototype.js to app/webroot/js.

Now that you have script.aculo.us, the JavaScript needs to be included on every

page. Open the file app/views/layouts/default.ctp and add the JavaScript related lines shown below.

Listing 14. Changing the header

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<?php
if ( isset($javascript) ) {
  echo $javascript->link('prototype.js');
  echo $javascript->link('scriptaculous.js?load=effects');
  echo $javascript->link('controls.js');
}
?>
...
```

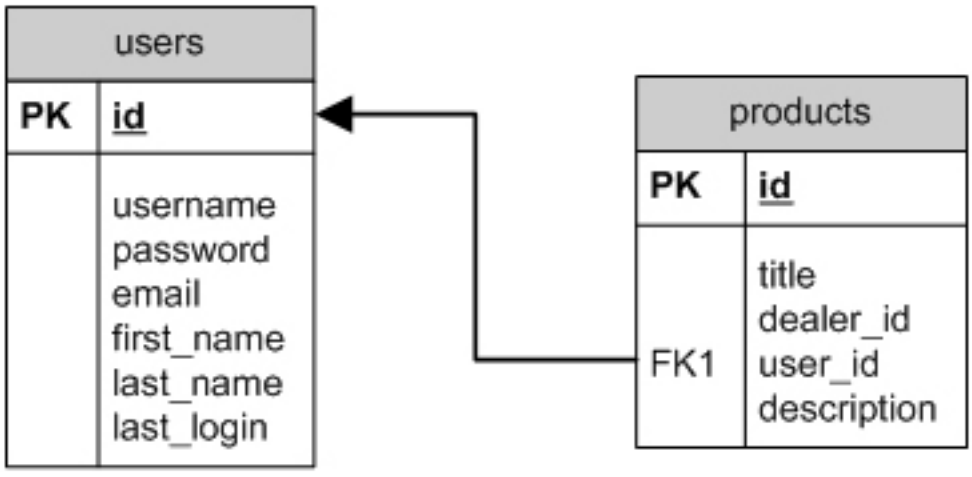
That's pretty much all you're going to have to do with the script.aculo.us library. Everything from here on out only involves objects in CakePHP.

Adding a list of favorites

Recall from Part 2 that you were able to link up two models with the `hasMany` and `belongsTo` relationships. In particular, you linked up Dealers with Product. There is another relationship, `hasAndBelongsToMany`, which is particularly useful when you are linking many models.

To illustrate, imagine you created a user's favorite items by using the `hasMany` relationship. Then each product must "belongTo" a particular user, meaning that no other user could mark it as a favorite. That would not be a good thing. The resulting database structure is shown in Figure 2.

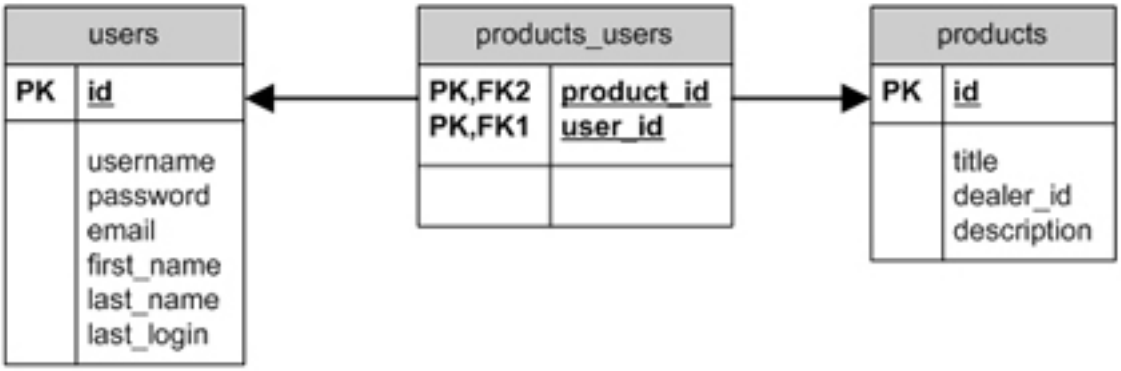
Figure 2. Database schema for hasMany and belongsTo relationship



Notice that a product may belong to only one user. Instead, the `hasAndBelongsToMany` relationship uses a "join table," which is a way to ensure that a user "hasMany" products, without the products "belongingTo" a single user. A

join table simply keeps track of a list of pairs — a user and a product — to signify that the two are linked. The resulting database structure is shown below.

Figure 3. Database schema for hasAndBelongsToMany relationship



Any number of connections can be made between the two tables. If you set up a join table properly, CakePHP will automatically create the proper links for you. To ensure that CakePHP can find your join table, there are a number of name conventions you should follow:

- The name of the table should be the names of the two models, pluralized, in alphabetical order. For example, if you were linking a user and group object, the name of the table should be groups_users.
- The table should have two foreign keys, each named the same as the "belongsTo" relationship. For example, a users/groups join table would have the fields user_id and group_id.
- For optimization, the two keys should form a primary key on the join table.

In our case, we are linking products to users, so the SQL query will create a table that meets these standards.

Listing 15. Create a table to link products to users

```
CREATE TABLE products_users (
  product_id int(5) NOT NULL,
  user_id int(5) NOT NULL,
  PRIMARY KEY (product_id,user_id)
) ENGINE=MyISAM;
```

To link up the two models, we simply add the `hasAndBelongsToMany` relationship to the product and user models.

Listing 16. Add hasAndBelongsToMany relationship to product and user models

```
//In app/models/user.php:
<?php
  class User extends AppModel
  {
    ...

    var $hasAndBelongsToMany = array( 'Product' =>
        array(
            'className' => 'Product'
        )
    );
}
?>

//In app/models/product.php:
<?php
  class Product extends AppModel
  {
    ...

    var $hasAndBelongsToMany = array( 'User' =>
        array(
            'className' => 'User'
        )
    );
}
?>
```

Now, any time a product or user is read from the database, it will check the products_users table to see if there are any links that should be made. If it finds applicable rows, it will load up the corresponding objects. Create a new action in the users_controller.php called favorites.

Listing 17. Retrieving a list of favorites

```
   function favorites () {
       $username = $this->Session->read('user');
       $favorites = array();

       if ($username)
       {
         $this->User->recursive = 2;
         $results = $this->User->findByUsername($username);

         foreach($results['Product'] as $product) {
           $favorites[] = array('Product' => $product, 'Dealer' =>
$product['Dealer']);
         }

         $this->set('products', $favorites);
       }
   }
```

The addToFavorites action

From an interface standpoint, we want the user to add a product to their favorites. Therefore, we need an `addToFavorites` action in the `ProductsController`.

Open the file app/controllers/products_controller.php and add the `addToFavorites` action, shown below.

Listing 18. Adding addToFavorites action

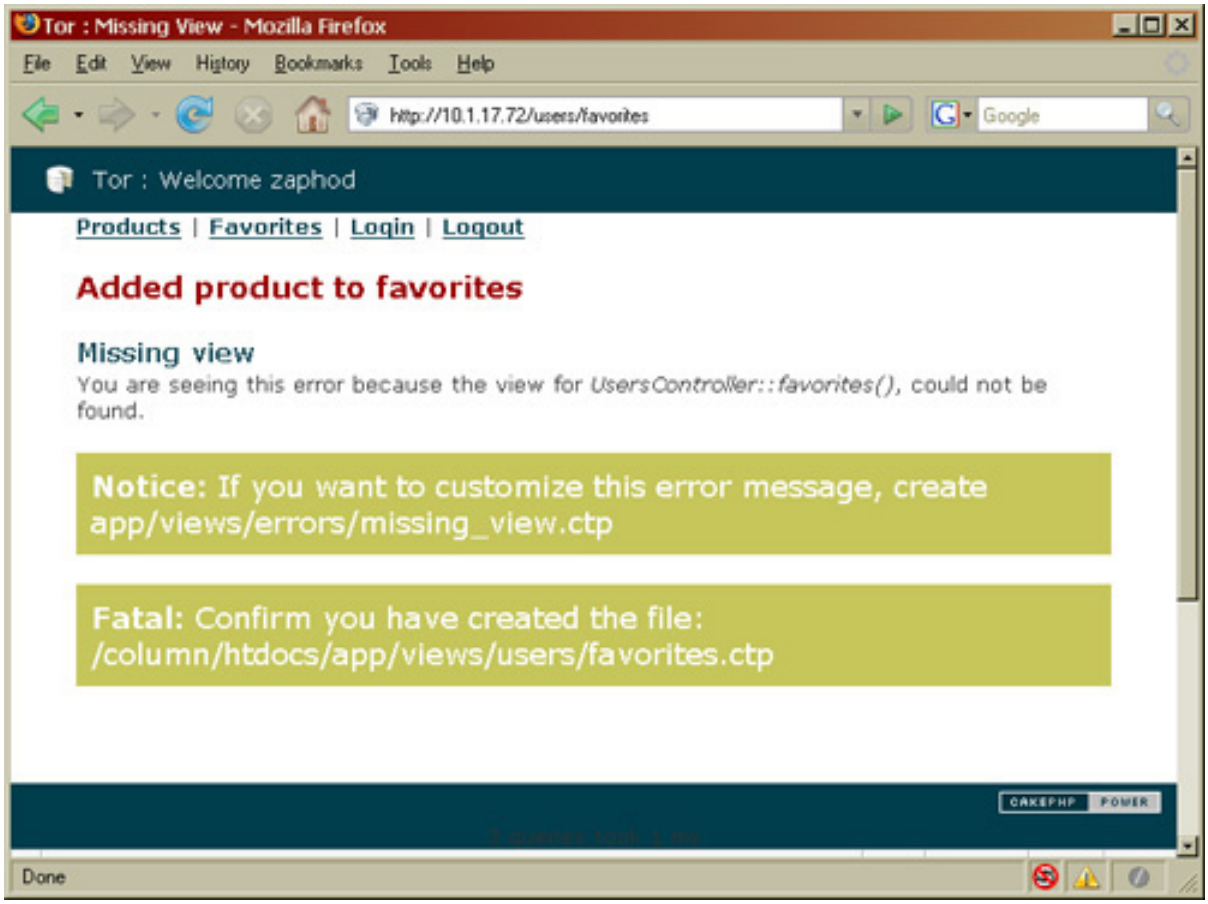
```
function addToFavorites($id) {
   $product = $this->Product->read(null, $id);
   $username = $this->Session->read('user');
   $success = false;
        if ($this->Acl->check($username, $id.'-'.$product['Product']
                                          ['title'], 'read')) {
             $result = $this->Product->User->findByUsername($username);
              product['User'] = array( 'User' =>
                  array($result['User']['id'])
              );
              $this->Product->save($product);
              $success = true;
        }
        if ( $this->RequestHandler->isAjax() ) {
              $this->layout = 'ajax';
              if ( $success ) {
                    echo 'Added product to favorites';
              } else {
                    echo 'Access denied';
                     }
        } else {
               if ( $success ) {
                    $this->Session->setFlash('Added product to favorites');
                    $this->redirect(array('controller' => 'users', 'action'
                                                    => 'favorites'), null, true);
                } else {
                    $this->Session->setFlash('Access denied');
                    $this->redirect(array('action' => 'index'), null, true);
                }
        }
  }
```

Let's go over what this action does. First, it does a quick ACL check to see if the user is actually able to read the product. Next, it loads up the user object based on value stored in the `user` session variable. Then it formats the data for saving and saves the item in the favorites list.

Finally, Request Handler is called to see if the action was an Ajax request. If it was, an informational message is returned. Otherwise, the user is redirected to the appropriate page.

You can now go to /products to try out the action, giving the result shown in Figure 4. View the list of products and pick one by ID, such as product 8. Then go to http://localhost/products/addToFavorites/8 and see the results. Even though you have no evidence of it other than the output message, the link has been made. Connect to your database and look at the products_users table, and you should see a single row linking your user_id with the selected product_id.

Figure 4. Added product successful



Linking to the addToFavorites action

Finally, you should provide a link to the `addToFavorites` action on the view products page. You'll utilize the Ajax helper's link method to create the link. The method, `$ajax->link`, acts much like the `$html->link` method. It takes the following parameters:

- `$title` — A string that will be used for the title of the link — in this case, "Delete"
- `$href` — The action the link will perform
- `$options` — An array of options, the most important of which is update, the ID of the element that the response text should feed into — in this case, we are updating the entire table
- `$confirm` — A JavaScript alert that will pop up confirming the action

This method outputs all the JavaScript needed to make a full Ajax request. On the back end, we only need to update the products controller to account for a new type of request. Utilizing the Request Handler component, we change the behavior if the action is part of an Ajax call, as shown in Figure 18. You'll also need to add a test in the products View view to see if Ajax is enabled and, if so, display an Ajax link. This link will be looking to update a DOM element called `favMessage`. You will want to

create that somewhere in the view, like at the end of the product div.

Listing 19. Adding to favorites links in the products view

```
<?php
if ( isset($ajax) ) {
      echo $ajax->link('Add to Favorites', array('action' => 'addToFavorites/' .
         $product['Product']['id']), array('update' => 'flashMessage'));
} else {
      echo $html->link('Add to Favorites', array('action' => 'addToFavorites/' .
         $product['Product']['id']));
}
?>

...

<span id='favMessage'> </span>
...
```

For this to work, you need to add the JavaScript and Ajax helpers to the Products controller like so: `var $helpers = array('Html', 'Form', 'Javascript', 'Ajax' );`.

By using CakePHP's Ajax helper, we've added Ajax functionality without having to write a single line of JavaScript.

Filling in the gaps

Rather than give you code to display the favorites table, you should try doing it on your own. You need to add the `favorites` method to the users controller and a users/favorites view.

When you are done with that, try adding a "Remove from favorites" link to the products table. Set it up so that the user is shown the "Add/Remove" link based on whether the product is already in favorites.

Section 5. Summary

Any Web application will run into the problem of storing sessions eventually. With CakePHP, you are given a great deal of flexibility on how you want to address the issue. Database sessions can easily be set up by changing a parameter and creating a table, which is a big time-saver.

You also learned about how the Request Handler is a multipurpose tool for changing the way your application is output. You saw how it can be used to handle Ajax requests and output RSS feeds. When you are playing around with CakePHP, try experimenting with other request types, such as detecting mobile devices.

Downloads

Description	Name	Size	Download method
Part 4 source code	os-php-cake4.source.zip	13KB	HTTP

Information about download methods

Resources

Learn

- Visit CakePHP.org to learn more.
- The CakePHP API has been thoroughly documented. This is the place to get the most up-to-date documentation.
- There's a ton of information available at The Bakery, the CakePHP user community.
- CakePHP Data Validation uses PHP Perl-compatible regular expressions.
- Read a tutorial titled "How to use regular expressions in PHP."
- Want to learn more about design patterns? Check out *Design Patterns: Elements of Reusable Object-Oriented Software* , also known as the "Gang Of Four" book.
- Check out some Source material for creating users.
- Check out Wikipedia's Model-View-Controller.
- Here is more useful background on the Model-View-Controller.
- Check out a list of software design patterns.
- Read more about Design Patterns.
- PHP.net is the central resource for PHP developers.
- Check out the "Recommended PHP reading list."
- Browse all the PHP content on developerWorks.
- 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.
- Using a database with PHP? Check out the Zend Core for IBM, a seamless, out-of-the-box, easy-to-install PHP development and production environment that supports IBM DB2 V9.
- 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.
- Watch and learn about IBM and open source technologies and product functions with the no-cost developerWorks On demand demos.

Get products and technologies

- Innovate your next open source development project with IBM trial software, available for download or on DVD.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- 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

Sean Kelly
Sean Kelly graduated with a degree in mathematics from Reed College. He is currently a Web application developer for ID Society, a full-service Internet marketing agency in New York City. He is a supporter of open source content management systems, and contributes to Joomla! and the Wikipedia project.