Cook up Web sites fast with CakePHP, Part 1:
Getting started

Quick and easy PHP rapid-development aid

Skill Level: Intermediate

Duane O'Brien
PHP developer
Freelance

21 Nov 2006

Updated 08 Feb 2011

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. 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.

Section 1. Before you start

Editor's note: This series was originally published in 2006, and has undergone several revisions since in order to keep up with ongoing developments on CakePHP. This revision was written for CakePHP V1.3.4.

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 how to use 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. Part 3 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, and so on).

- 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

This tutorial shows how to get started using CakePHP. You'll go through the installation process, then get down and dirty by writing the user registration code. Through it all, you'll see how much time you could have saved had you been using CakePHP all along. This part of the tutorial builds the online product application, *Tor*, which includes a "request dealership username and password" page and a login page.

CakePHP topics include:

- MVC design

- Helpers

- CakePHP data validation

Prerequisites

It is assumed that you are familiar with PHP, have a fundamental grasp of database

design, and are comfortable getting your hands dirty. A full grasp of the MVC design pattern is not necessary, as the fundamentals will be covered during this tutorial. More than anything, you should be eager to learn, ready to jump in, and anxious to speed up your development time.

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 CakePHP.org and download the latest stable version from the Downloads section (see Resources). This tutorial was written using V1.3.4.

Section 2. Installation

CakePHP wants to make your life easier, regardless of your level of experience, by making your applications easier to maintain and quicker to write. CakePHP is full of cool and useful features. CakePHP wants to handle your Ajax, your data validation, your sessions. It will even slice your bread if you can write a plug-into tell it how. But you can't use CakePHP yet. You need to install it first.

Unpack and install

For the purpose of this tutorial, the entire CakePHP installation directory should be unpacked within the webroot of your Web server. In Listing 1, the webroot is

/webroot.

Listing 1. Unpacking the CakePHP installation directory

```
unzip cakephp-cakephp-1.3.4-0-g703344c.zip
cd cakephp-cakephp-df6ffd4
mv * /webroot
```

Type `ls -la /webroot` to list the contents of the webroot and verify that the files have been moved properly. The output should look something like Listing 2.

Listing 2. Output of the ls command

```
-rw-r--r--    1 YOURUSER   YOURGROUP       139 2010-09-12 00:57 .htaccess
-rw-r--r--    1 YOURUSER   YOURGROUP      1154 2010-09-12 00:57 README
drwxr-xr-x   13 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 app
drwxr-xr-x    6 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 cake
-rw-r--r--    1 YOURUSER   YOURGROUP      1848 2010-09-12 00:57 index.php
drwxr-xr-x    2 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 plugins
drwxr-xr-x    3 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 vendors
```

The directory app/tmp needs to be writable by your Web server. Confirm the permissions on this folder by typing `ls -l app`. The output will probably look similar to Listing 3.

Listing 3. Confirming the folder permissions

```
-rw-r--r--    1 YOURUSER   YOURGROUP       141 2010-09-12 00:57 .htaccess
drwxr-xr-x    3 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 config
drwxr-xr-x    3 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 controllers
-rw-r--r--    1 YOURUSER   YOURGROUP       685 2010-09-12 00:57 index.php
drwxr-xr-x    2 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 libs
drwxr-xr-x    3 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 locale
drwxr-xr-x    4 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 models
drwxr-xr-x    2 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 plugins
drwxr-xr-x    5 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 tests
drwxr-xr-x    6 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 tmp
drwxr-xr-x    3 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 vendors
drwxr-xr-x    8 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 views
drwxr-xr-x    6 YOURUSER   YOURGROUP      4096 2010-09-12 00:57 webroot
```

The simplest way to accomplish this is probably the most common and least secure: give write permissions to everyone (see Listing 4).

Listing 4. Giving write permissions to everyone

```
chmod -R 777 app/tmp
ls -l app
```

The permissions for the tmp folder should have been updated, as shown in Listing 5.

Listing 5. Updating the permissions for the tmp folder

```
drwxrwxrwx    6 YOURUSER   YOURGROUP      2010-09-12 00:57 tmp
```

Giving write permissions to everyone is **not** recommended for general use. Ideally, you should change the ownership of this folder to match the user that the Web server uses, or add the user that the Web server uses to a the group for the directory and add group write permissions. This tutorial is intended to demonstrate how to use CakePHP and is not designed to be a guide for building secure applications. While security should be at the head of any application development, a full discussion of secure PHP practices is outside the scope of this tutorial.

For a production installation, change the webroot of the Web server to app/webroot, which will minimize the amount of code accessible via the Web browser and help enhance the security of your installation.

Validation 1

In a browser, go to the URL that corresponds with the webroot for your Web server. For example, if you've installed CakePHP into the webroot of your localhost, go to http://localhost; you should see the CakePHP default home page (see Figure 1).

Figure 1. The CakePHP default home page as it should be seen

CakePHP: the rapid development php framework

Release Notes for CakePHP 1.3.4.

Read the changelog

Notice (1024): Please change the value of 'Security.salt' in app/config/core.php to a

Notice (1024): Please change the value of 'Security.cipherSeed' in app/config/core.php

Your tmp directory is writable.

The *FileEngine* is being used for caching. To change the config edit APP/config/core.php

Your database configuration file is NOT present.
Rename config/database.php.default to config/database.php

Editing this Page

To change the content of this page, create: APP/views/pages/home.ctp.
To change its layout, create: APP/views/layouts/default.ctp.
You can also add some CSS styles for your pages at: APP/webroot/css.

Note: If the default home page looks more like Figure 2, then `mod_rewrite` is not working the way CakePHP requires. This can sometimes be a problem for first-time users.

Figure 2. The incorrect-looking home page

CakePHP: the rapid development php framework

Release Notes for CakePHP 1.3.4.

Read the changelog

Notice (1024): Please change the value of 'Security.salt' in app/config/core.php to a salt

Notice (1024): Please change the value of 'Security.cipherSeed' in app/config/core.php to a

Your tmp directory is writable.

The *FileEngine* is being used for caching. To change the config edit APP/config/core.php

Your database configuration file is NOT present.
Rename config/database.php.default to config/database.php

Editing this Page

To change the content of this page, create: APP/views/pages/home.ctp.
To change its layout, create: APP/views/layouts/default.ctp.
You can also add some CSS styles for your pages at: APP/webroot/css.

Getting Started

New CakePHP 1.3 Docs

The 15 min Blog Tutorial

More about Cake

CakePHP is a rapid development framework for PHP which uses commonly known design patterns like Active

The following list are some items to verify.

Is your .htaccess file correct?

You should have gotten an .htaccess file in the CakePHP installation directory. On most *nix systems, this will be hidden from view by default. If you do not have the file, check the source you downloaded or get a fresh update from CakePHP.org. Confirm that the file exists and is valid by going to the installation directory and running `cat .htaccess`. This will display the file's contents, which should look like Listing 6.

Listing 6. Confirming that the .htaccess file exists

```
<IfModule mod_rewrite.c>
   RewriteEngine on
   RewriteRule    ^$ app/webroot/    [L]
   RewriteRule    (.*) app/webroot/$1 [L]
</IfModule>
```

Is mod_rewrite enabled for the server?

Make sure that `mod_rewrite` is enabled for your Web server. For Apache, there are two lines that should appear in the httpd.conf file. In the LoadModule list, you should see the following line (or something very close to it): `LoadModule rewrite_module libexec/mod_rewrite.so`. In the AddModule list, you should see this line (or something very close): `AddModule mod_rewrite.c`.

If you cannot find these lines in your httpd.conf file, `mod_rewrite` is not enabled. Consult your server documentation for details on how to do this.

Does the server allow .htaccess override?

Make sure your Web server is configured to allow .htaccess override. For Apache, each directory should be defined in the httpd.conf file. These definitions can look very different from installation to installation, but you should still see the line `AllowOverride All` in the definition. Your definition might look something like Listing 7.

Listing 7. Definitions in the httpd.conf file

```
<Directory "/webroot">
    Options Indexes MultiViews
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

Consult your server documentation for more details about .htaccess override.

Configuring a database connection

Now that you've got CakePHP installed and on speaking terms with your Web server, you need to introduce CakePHP to your database. This section will cover setting up the database configuration and verifying that CakePHP likes your database. Tor is going to need a place to store its user and product data. You'll be soon be creating a users table to be used to build the login and registration parts of Tor.

Editing the database configuration file

Setting up your database configuration is pretty darn easy, but before you start, make sure your database server is running, that you have created a database for your application, and that you have a username and password for a user with rights

to the database.

To start, make a copy of the app/config/database.php.default file and save it as app/config/database.php. Do this to preserve a copy of the original template. Open the file in your favorite text editor and look for the section shown in Listing 8 (it should be just near the bottom of the file).

Listing 8. The app/config/database.php.default file

```
var $default = array(
                'driver' => 'mysql',
                'persistent' => false,
                'host' => 'localhost',
                'login' => 'user',
                'password' => 'password',
                'database' => 'database_name',
                'prefix' => ''
        );
```

Modify this information to fit your installation:

driver
> This can be `mysql`, `mysqli`, `postgres`, `sqlite`, `mssql`, `db2`, `oracle`, `adodb` or `pear-drivername`. This tutorial assumes `mysql`.

persistent
> This field tells CakePHP whether it should use persistent database connections. This should be true or false. This tutorial assumes false.

host
> This is the hostname of your database server, such as `localhost` or `mysql.yourdomain`.

login
> This is the username for your database login, such as `dbuser`.

password
> This is the password for your database login, such as `secretsecret`.

database
> This is the name of the database you wish to use, such as `cakedev`.

prefix
> Prefix is a string, such as `cp_`, that is prepended to table names for any database call made by CakePHP. Using a prefix may be necessary if the database is shared among applications to keep tables from stepping on each other where two or more applications want a table with the same name, such as `users`.

Getting started
Trademarks
Page 9 of 37

Don't forget to save the file.

Any number of database configurations can be specified in database.php provided they have distinct names. You can specify which database configuration the application should use in the models.

Some notes about databases and CakePHP:

- The tables must have a primary key named id.

- If you include a *created* or *modified* column in the table, CakePHP will automatically populate the field when appropriate.

- Table names should be plural (users, products, eggs, sodas, winners, losers). Their corresponding models will have singular names (user, product, egg, soda, winner, loser).

- If tables are to be related, foreign keys should follow the format table_id with singular table names. For example, user_id, product_id, egg_id, soda_id, winner_id and loser_id would be foreign keys for the table's user, product, egg, soda, winner, and loser.

Validation 2

Go back to the URL you used to validate the initial installation. You should see that the CakePHP default home page has changed to indicate the status of your database configuration (see Figure 3).

Figure 3. CakePHP default home page has changed to indicate status of database configuration

CakePHP: the rapid development php framework

Release Notes for CakePHP 1.3.4.

Read the changelog

Notice (1024): Please change the value of 'Security.salt' in app/config/core.php to a

Notice (1024): Please change the value of 'Security.cipherSeed' in app/config/core.php

Your tmp directory is writable.

The *FileEngine* is being used for caching. To change the config edit APP/config/core.php

Your database configuration file is present.

Cake is able to connect to the database.

Editing this Page

To change the content of this page, create: APP/views/pages/home.ctp.

If the default home page says the database configuration file is not present, you may have put it in the wrong place or misnamed it. Make sure the database configuration file is app/config/database.php. If the default home page says CakePHP is unable to connect to the database, confirm that the connection information you entered is valid and retry.

Creating the application tables

So now CakePHP, your Web server, and your database are all on speaking terms. Time to roll up your sleeves and get to the real work. Tor needs a users table.

This table will contain the basic information necessary to identify and interact with a user. A simple username and password field would probably suffice, but other information can be useful, such as an e-mail address (for sending password reset requests), first and last name (for personalization), and last login date (to help track inactive accounts). You probably want your username and e-mail fields to be unique.

And don't forget the primary key ID field. The SQL to create your table might look
something like Listing 9.

Listing 9. SQL to create your table

```
CREATE TABLE 'users' (
'id' INT( 10 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
'username' VARCHAR( 40 ) NOT NULL ,
'password' VARCHAR( 40 ) NOT NULL ,
'email' VARCHAR( 255 ) NOT NULL ,
'first_name' VARCHAR( 40 ) NOT NULL ,
'last_name' VARCHAR( 40 ) NOT NULL ,
    UNIQUE (
        'username' ,,
            'email'
                )
) TYPE = MYISAM ;
```

Notice that the `username`, `password`, `first_name`, and `last_name` fields have a
maximum character length of 40. We will enforce this character length in the user
model. The 40-character maximum length, in this case, is entirely arbitrary.

Section 3. MVC design

It's common enough at this point to jump in and start cranking out code based on
what the application is intended to do. A login page, a database, registration —
BAM! Simple enough. And it usually is. Until you want to change how it looks, how it
interacts with the database, change validation rules, or change essentially anything
about the application at all. Then things start to get complicated.

CakePHP is designed to make writing the application as easy as possible, while
giving you something you can maintain long-term. The tutorial will show you how to
build Tor using the MVC design pattern. CakePHP will show you how it can be easy.
You'll see for yourself why it makes sense.

The MVC design pattern breaks an application into three distinct layers: Data
management, UI and Logic, respectively. MVC was first described in the book
Design Patterns: Elements of Reusable Object-Oriented Software, also known as
the "Gang Of Four" book. A full discussion of design patterns and MVC is outside
the scope of this tutorial, but it will be helpful to touch on what the three pieces mean
— specifically, in the CakePHP world.

Model

Users, products, prices, messages — when you boil it all down, it's all just data. You make sure the data is what you want, it goes into a database, then you have to get it back out. It will be useful for all the data-handling aspects of Tor to live in one place. That where model comes in.

Model is primarily concerned with handling data. Getting data from a database, inserting data into a database, validating data — all of this takes place within the model.

An individual model typically is used to access a specific table in the database. Generally, a model will associate to a database table if the table name is a plural form of the model name. For example, a Product model will, by default, associate to a Products table. All user-defined models will extend the CakePHP `AppModel` class.

The user model

Having created a users table for Tor, you will need a user model to interact with the table. The user model will save your user data, get it back out of the table for you, and verify that the user data is in an acceptable format. For now, this model can be a stub. It should be created as app/models/user.php and might look something like Listing 10.

Listing 10. The user model

```php
<?php
class User extends AppModel
{
        var $name = 'User';
}
?>
```

Notice the line `var $name = 'User';`. It is accepted as best practice to specify the name of the model in `$name`.

View

Being able to save, retrieve, and validate your data is pretty useless without some means of displaying the data. By putting all display and presentation code in one place, you can change the look and feel of your application without having to work around application logic and data-related code.

View is primarily concerned with formatting data for presentation to the user. The view represents any and all UI work, including all templates and HTML. CakePHP's view files are regular HTML files embedded with PHP code.

Ultimately, a view is just a page template. Usually, the view will be named after the action associated with it. For example, a `display()` action would normally have a display view.

Returning to the sample application, we have created a users table and a user model to interact with this table. Now we need some views. The user will need to register an account and log in to the application. This calls for a register and a login view.

The register view

Before a user can use the Tor application, he will need to register an account. This will require a register view, which will present the registration form to the user. You should already know what kind of user information Tor is looking for — specifically, you will need to collect a username, password, e-mail address, and first and last name.

The register view should be created as app/views/users/register.ctp and might look something like that shown in Listing 11.

Listing 11. The register view

```
<form action="/users/register" method="post">
<p>Please fill out the form below to register an account.</p>
<label>Username:</label><input name="username" size="40" />

<label>Password:</label><input type="password" name="password" size="40"
/>

<label>Email Address:</label><input name="email" size="40"
maxlength="255" />

<label>First Name:</label><input name="first_name" size="40" />

<label>Last Name:</label><input name="last_name" size="40" />

<input type="submit" value="register" />
</form>
```

It is important to note that the field names are the same as the column names in your database. This will come into play when we get to the users controller. Later, when you learn a little more about Cake and see how Cake Helpers work, you will rewrite this page completely, but for now this page is regular HTML, just like you have probably always written.

Controller

With data handling all contained within in the model, and the presentation layer all

contained within the view, the rest of the application is going to live in the controller. This is where the application 'does' things — logic, decision-making, workflow, etc. The model manages your data, the view shows it to you, the controller does everything else.

The controller manages server requests. It accepts user input (URL requests, form POSTs, GET requests, etc.), applies logic where necessary, invokes models where data handling is required, and sends output through the appropriate view.

Generally, a controller will manage logic for a single model. A controller contains any number of functions, referred to as *actions.* An action typically applies logic and displays a view. All user-defined controllers will need to extend the CakePHP AppController class.

In Tor, you have created a users table, a user model to interact with the table, and views to present registration and login forms to the user. Those forms will submit to the users controller, invoking the register and login actions, respectively. The users controller should be created as app/controllers/users_controller.php and should start out looking something like Listing 12.

Listing 12. The users controller

```
<?php
class UsersController extends AppController
{
}
?>
```

The register action

You now have a controller. But it's not doing anything yet. Before it will do anything for you, you'll have to give it some actions to perform.

Now you are getting down to the nuts and bolts. The user has filled out a form and submitted it to the application. We will cover data validation later, but for now, we will just push the registration data into the database. We do this by adding the register action to the users controller (see Listing 13).

Listing 13. The register action

```
<?php
class UsersController extends AppController
{
  function register()
  {
    if (!empty($this->params['form']))
    {
```

```
        if ($this->User->save($this->params['form']))
        {
          $this->flash('Your registration information was accepted.',
'/users/register');
        } else {
          $this->flash('There was a problem with your registration',
'/users/register');
        }
      }
    }
  }
}
?>
```

The `register` action starts by looking at the form (`$this->params['form']`) to see if it was submitted. If the form was not submitted, the action will not do anything. Once you know the form was submitted with data, the controller calls the `save` method on the user model, which is an extension of `AppModel`. The save method will, by default, look for a table that's a plural of the model itself. In this case, it is looking for a users table to go with the user model. If the table is found, the save method will turn the passed array (`$this->params['form']`) into an `INSERT` statement, using the array keys as column names and the array values as the `INSERT` values.

Once you know the form was submitted with data, the controller calls the `save` method on the user model, which is an extension of `AppModel`. The `save` method will, by default, look for a table that's a plural of the model itself. In this case, it is looking for a users table to go with the user model. If the table is found, the save method will turn the passed array (`$this->params['form']`) into an `INSERT` statement, using the array keys as column names and the array values as the `INSERT` values. In this case, `$this->params['form']` will look like Listing 14.

Listing 14. The $this->params['form'] array

```
Array
(
    [username] => zaphod
    [password] => secret
    [email] => beeblebrox@heartofgold.hhg
    [first_name] => zaphod
    [last_name] => beeblebrox
)
```

The `save` method will build an `INSERT` statement from the rest. It would look something like Listing 15.

Listing 15. INSERT statement

```
INSERT INTO
 users
(username, password, email, first_name, last_name)
VALUES
```

```
('zaphod', 'secret', 'beeblebrox@heartofgold.hhg', 'zaphod', 'beeblebrox')
```

It should now be apparent why you used the database column names in the register view for the register form field names. By doing so, you simplified the process of saving your data significantly.

To continue, if the data inserts successfully, the `register` action calls the Flash method. Flash presents a basic message to the user (in this case, a success or failure message) with a link away from the message (in this case, back to the register view, since nothing else has been defined.

Now that you have a user model to interface with your users table, and a register view to show a registration form, and a users controller with a `register` action, you can actually see your application in action.

Try it out

All the pieces have fallen into place. It's time to bring Tor to life. Fire up your browser and jump in. Load the register view by going to http://localhost/users/register. You should see something like Figure 4.

Figure 4. Loading the register view

Fill out and submit the form. You should see your success message (see Figure 5).

Figure 5. Success message

If you look in your database, you should see a record corresponding to your form submission, complete with a plain-text password (a terribly bad idea, which we will fix later). Try to fill out the form again, using the same information. You should see the failure message (see Figure 6).

Figure 6. Register failure


```
Warning (512): SQL Error: 1062: Duplicate entry 'zaphod-beeblebrox@heartofgold.hhg' for key 2

Query: INSERT INTO `users` (`username`, `password`, `email`, `first_name`, `last_name`

                    There was a problem with your registration
```

If you are using a database that supports the UNIQUE field restraint, the registration will fail. During the creation of the users table, the username and e-mail fields were defined as UNIQUE, meaning values for those fields cannot be duplicated. CakePHP recognizes the error when the database throws it and displays an error.

Section 4. Helpers

Helpers in CakePHP exist primarily to help speed up the development of your views. There are helpers for HTML, Ajax, JavaScript and more. Helpers make it easier to insert pieces of HTML code you find yourself writing multiple times.

Modifying the users_controller.php file

To start using Helpers, you will need to modify the users_controller.php file you created earlier. You should give the controller a name and tell it that you want to use some helpers — in this case, the HTML and Form helpers. Edit the file to add the $name and $helpers class variables (see Listing 16).

Listing 16. The users controller

```php
<?php
class UsersController extends AppController
{
    var $name = 'Users';

    var $helpers = array('Html', 'Form' );


    function register()

    {
```

```
...
```

Now that you have included your helpers, you can begin using them.

Making tables easier

Users of Tor should be able to see who else is registered to use the application. CakePHP has a number of helpers in place to assist with creating tables. These helpers include many useful bits of functionality, some of which you have probably written more than once. To demonstrate this, you will create a view to display registered users.

Start by creating a `knownusers` action in the users controller (see Listing 17).

Listing 17. Creating a knownusers action

```
function knownusers()
{
  $this->set('knownusers', $this->User->find(
    'all',
    array(
      'fields' => array('id','username', 'first_name', 'last_name'),
      'order' => 'id DESC'
    )
  ));
}
```

This calls the built in `find` function on the user model. The `find` function takes a number of parameters, such of matching conditions (in this case, you passed `all` instead of some matching conditions), an array of fields to be returned (we don't want all of the user information — just what you would want everyone to see), and a sort field and order (in this case, `id DESC` to sort the fields in descending order by ID). You can also specify a limit (maximum rows to return), page (if you are paging the data), and a recursive option, which can be specified to return models associated with the data (for example, if you were querying a groups table and several users belong to each group).

The output from `find` is put into the `knownusers` variable. The data can now be accessed from a view.

Creating the knownusers view

Create the file app/views/users/knownusers.ctp in a text editor. To see what the data returned by `findAll` looks like, output the `knownusers` variable using `var_dump` (see Listing 18).

Listing 18. Output the knownusers variable using var_dump

```
<pre>
<?php var_dump($knownusers) ?>
</pre>
```

Visit the view at `http://localhost/users/knownusers`. You should see an array of user data, shown in Figure 7.

Figure 7. The results

If you only have one user listed, go back to http://localhost/users/register and register a few more. The end result will be more impressive.

Got a nice big array of users? Good! Time to turn that into a table. Replace the contents of knownusers.ctp with the following from Listing 19.

Listing 19. Creating a table

```
<table>
<?php

echo $html->tableHeaders(array_keys($knownusers[0]['User']));

foreach ($knownusers as $thisuser)
{
        echo $html->tableCells($thisuser['User']);
}

?>
</table>
```

The first helper will create a set of table headers from an array of data — in this
case, the list of keys for our users. The second helper will create a set of table cells,
wrapped in table row tags, from an array of data — in this case, the values for each
user.

That's it! Save it, then visit http://localhost/users/knownusers to see the results (see
Figure 8).

Figure 8. The results

By using the `tableCells` helper, you have eliminated the need to write your own
code to iterate through the array of user data. This is just one example of how to use
helpers to make it easier to work with HTML in CakePHP.

Form generation

Building a Web application without using forms is like milking a chicken: It's

extremely complicated and rarely works. Well-built and maintainable forms are a foundation to any well-built application. Given how often you will need to build forms, it only seems natural to look for ways to make the process easier without cutting corners.

Using helpers in the registration form

Helpers are especially useful when generating forms. You can use them to generate the HTML for your input fields, as well as placeholders to hold validation error messages. Using helpers to generate the form fields and error message holders for Tor, the register view might look more like Listing 20.

Listing 20. Using helpers to generate the registration form

```
<p>Please fill out the form below to register an account.</p>
<?php echo $form->create('User', array('action' => 'register'));?>

<?php
    echo $form->input('first_name');
    echo $form->input('last_name');
    echo $form->input('username');
    echo $form->input('email');
    echo $form->input('password');
?>

<?php echo $form->end('Register');?>
```

As you can see, making use of CakePHP's helpers can save you quite a bit of code when writing basic form elements. Now let's look to make even more use of your helpers.

Making the most of your helpers

To start getting more payoff from your helpers, you'll need to do two things: update your users controller and introduce a little data validation. Open controllers/users_controller.php and change your register function to match the one in Listing 21.

Listing 21. Updating the users controller

```
function register()
{
  if (!empty($this->data))
  {
    if ($this->User->save($this->data))
    {
      $this->Session->setFlash('Your registration information was accepted.');
    }
  }
}
```

Note that the occurrences of `$this->params['form']` have changed to
`$this->data`.

Now open the user model and add a little data validation (covered in detail later). For
now, modify your user model to look like the one in Listing 22.

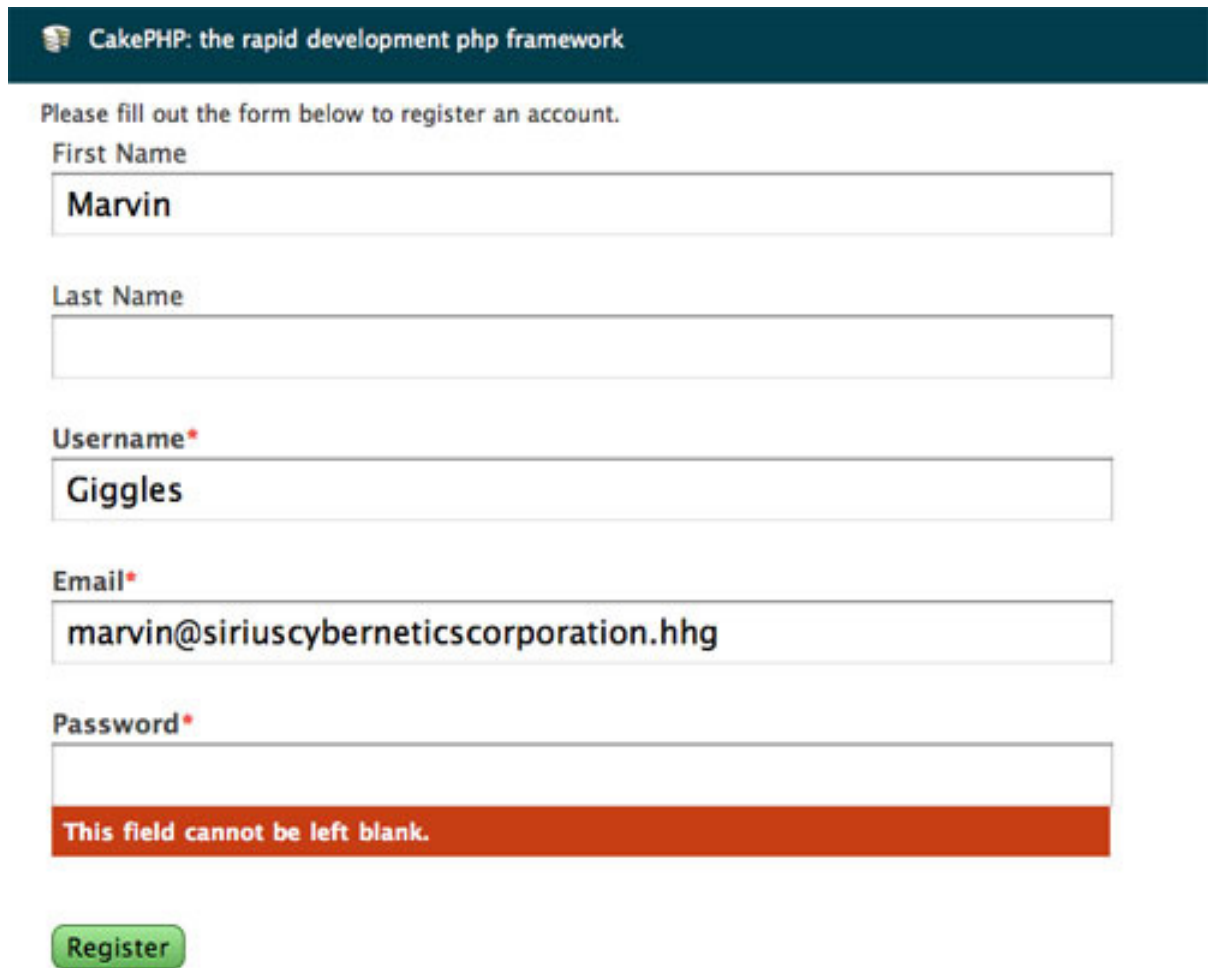**Listing 22. Modifying your user model**

```php
<?php
class User extends AppModel
{
  var $name = 'User';

  var $validate = array (
    'username' => array (
      'rule' => 'notEmpty',
      'message' => 'This field cannot be left blank.'
    ),
    'password' => array (
      'rule' => 'notEmpty',
      'message' => 'This field cannot be left blank.'
    ),
    'email' => array(
      'rule' => 'email',
      'message' => 'Please supply a valid email address.'
    )
  );
}
?>
```

To break this down briefly, the `$validate` array contains entries for validation,
consisting of a key (the form field name) and an array of parameters used to
evaluate the data — the rule to be invoked, and the message to be displayed when
the rule is violated. It is not necessary to validate all of the form fields. In Listing 20,
`last_name` and `first_name` were left optional. CakePHP comes with several
predefined regular expressions for data validation. The `notEmpty` rule is used just
to make sure the field is not empty, while the `email` rule is used to verify that a
string looks more or less like an e-mail address. There are many other predefined
validation rules available.

Now take it for a spin. Try submitting the form with no data, with one or two required
fields empty, with an invalid e-mail address (see Figure 9. What do you see?

Figure 9. Data validation first try

Now try it a second time with different missing data (see Figure 10).

Figure 10. Data validation second try

CakePHP: the rapid development php framework

Please fill out the form below to register an account.
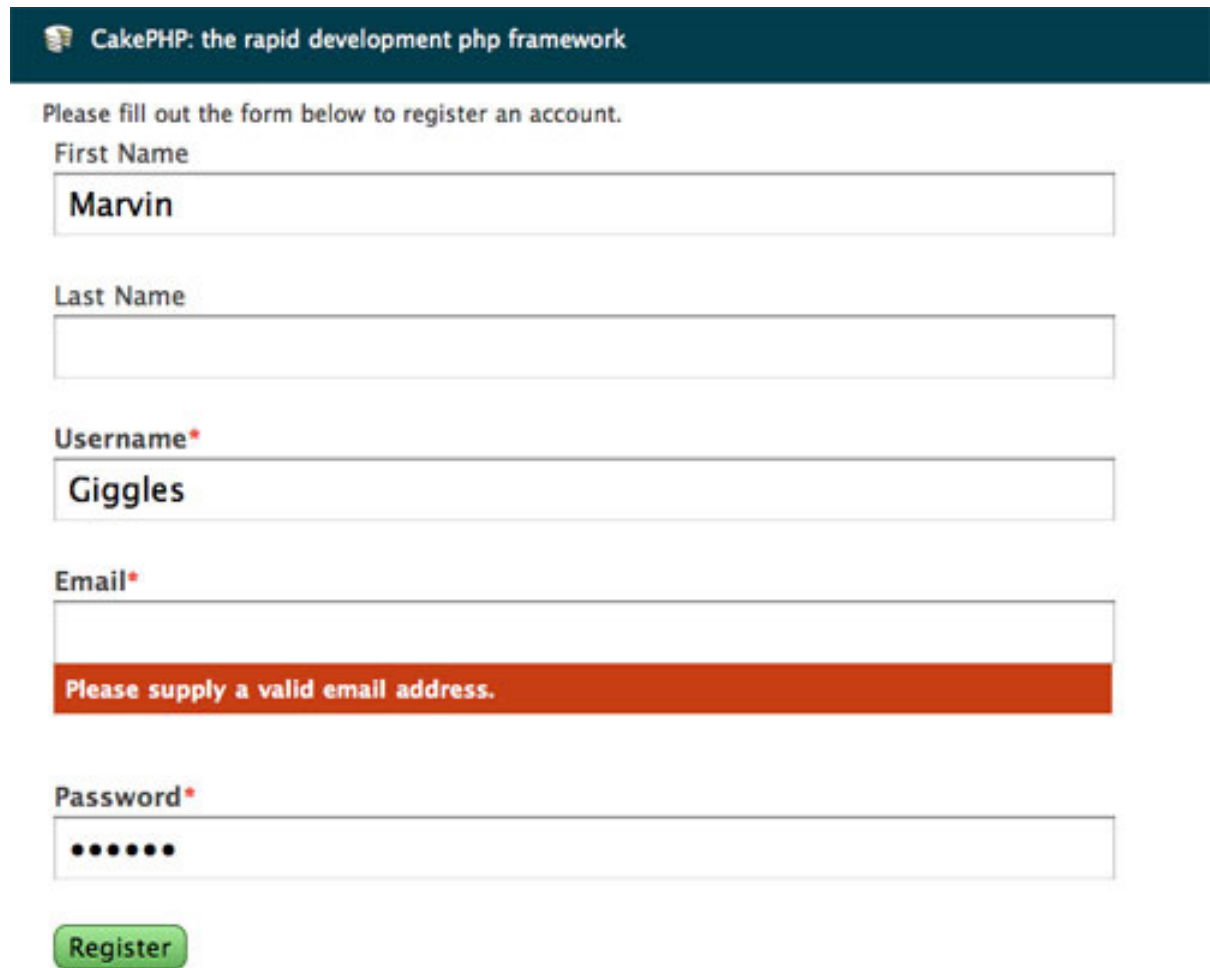
First Name

Marvin

Last Name

Username*

Giggles

Email*

Please supply a valid email address.

Password*

••••••

Register

One thing you should notice is that the CakePHP is turning on and turning off your error messages for you on the fly. Another thing you should notice is that CakePHP is remembering and populating the values for the form fields, without you having to do anything.

That's where the big payoff comes in. What didn't you have to do? For one thing, you didn't have to tell the form fields to repopulate their information from the _POST array. CakePHP did that for you. You didn't have to check each field for an error and conditionally display each message individually. CakePHP did that for you. You didn't have to make sure you formatted your tags into valid xhtml. CakePHP did that for you, too.

Helper notes

This tutorial barely scratches the surface of helpers. A whole tutorial could be written on the subject. Learning how to use helpers well will go a long way toward helping speed up your development in CakePHP. CakePHP includes helpers for Ajax (using prototype.js), JavaScript, number conversion, text handling, dates, times, and more. Review the manual (see Resources) to get more familiar with some of these helpers.

Section 5. CakePHP data validation

You now have a brief look at CakePHP data validation by putting in some basic validation for users based on defined regular expressions. By defining your own regular expressions for data validation, you can exercise more control over the pass/fail criteria for individual form fields within Tor.

The Tor user model

Take another look at the user model (see Listing 23).

Listing 23. The user model

```php
<?php
class User extends AppModel
{
  var $name = 'User';

  var $name = 'User';

  var $validate = array (
    'username' => array (
      'rule' => 'notEmpty',
      'message' => 'This field cannot be left blank.'
      ),
      'password' => array (
      'rule' => 'notEmpty',
      'message' => 'This field cannot be left blank.'
      ),
      'email' => array(
      'rule' => 'email',
      'message' => 'Please supply a valid email address.'
      )
}
?>
```

This is a good start, but it's not enough. You'll want to make sure the field lengths are honored and that the username does not already exist. You will accomplish this by defining your own regular expressions for validation and defining a function to check the users table for a username before saving the user.

Regular expressions (briefly)

A full discussion about how regular expressions is outside the scope of this tutorial. The PHP Manual contains information about regular expressions in PHP and should be reviewed before going too far in rolling your own data validation regular expressions (see Resources).

A *regular expression* is basically a pattern of characters used for comparing one string to another. For example, the character * in a regular expression will match any character, any number of times. If you don't know anything about regular expressions, don't worry. The example below should help get you started.

Roll your own validation

For the username and password fields, let's validate that the submitted data is no longer than 40 characters. It is also helpful to verify that the username and password are at least six characters. And let's also limit the characters to only alphanumerics. A regular expression to match strings with a length between six and 40 characters would look something like this: `/^[a-z0-9]{6,40}$/i`. Reading that regular expression from left to right:

- `/` — Marks the beginning of the regular expression
- `^` — Says *from the beginning of the string*
- `[a-z0-9]` — Says *any one character from a to z or 0 to 9*
- `{6,40}` — Says *at least six times, but no more than 40 times*
- `$` — Says *and the string ends*
- `/` — Marks the end of the regular expression
- `i` — Ignore the casing of any letter

So, read altogether, this regular expression says "from the beginning of the string, a letter or number, at least six but not more than 40, and the string ends, ignore case."

To put the regular expression to use (see Listing 24), replace the instances of `notEmpty` with the regular expression, in single quotes (to prevent PHP from trying to interpret any of the special characters).

Listing 24. Regular expression in PHP script

```
<?php
class User extends AppModel
```

```
{
  var $name = 'User';

  var $validate = array (
      'username' => array (
      'rule' => '/^[a-z0-9]{6,40}$/i',
      'message' => 'This field must have between 6 and 40 alphanumeric characters.'
      ),
      'password' => array (
      'rule' => '/^[a-z0-9]{6,40}$/i',
      'message' => 'This field must have between 6 and 40 alphanumeric characters.'
      ),
      'email' => array(
      'rule' => 'email',
      'message' => 'Please supply a valid email address.'
      )
  );

}
?>
```

Make sure you've saved all of your files, go back to http://localhost/users/register,
and try to register a user with a four-character username. You should see something
like Figure 11.

Figure 11. Data validation

Regular expressions are versatile, but they can't do things like tell you if a username has already been registered.

Taking validation further

Sometimes you can't tell if data is valid just by looking at it. For example, the username may be between six and 40 characters, but you will have to check the database to see if the username is already taken. CakePHP provides the ability to manually mark a field as invalid. Take a look at the beforeValidate method in Listing 25. This method would be added to the user model.

Listing 25. Validate the username

```
function beforeValidate()
```

```
{
  if (!$this->id)
  {
    if ($this->findByUsername($this->data['User']['username']))
    {
      $this->invalidate('username_unique');
      return false;
    }
  }
  return true;
}
```

This method tells the model that before any validation is run, check to see if the submitted data has an ID. If there is no ID, look for other users with the same username. If there are any, mark the username field invalid and skip any remaining validation (return false). You can take full advantage of this by changing the username input line in the register.ctp view to the following shown in Listing 26.

Listing 26. New username input line

```
echo $form->input('username', array('after' => $form->error
    ('username_unique', 'The username is taken. Please try again.')));
```

This tells the register view what to do when encountering error messages called 'username_unique' like you designated in the `beforeValidate` method.

Save your files and try it out. Go to http://localhost/user/knownusers to get a list of existing users. Then go to http://localhost/user/register and try to create one with the same username. You should see the following, in Figure 12.

Figure 12. Data validation successful

Good data validation is an important step in creating any secure application. As you build the Tor application, look for opportunities to improve the data validation. Don't be afraid to put in more data validation than this tutorial demonstrates. Never assume your users are sending you the data you asked for. Validate everything. CakePHP makes it easy.

Section 6. Filling in the gaps

So far, users can register for your application and see who has already registered. The application needs some filling in. Using the skills you've learned so far, try filling in some more functionality. Check out Part 2 for examples of the following.

Login

The login view should gather the user login information and submit it to the users controller. The users controller should look to see if the user is in the database and verify that the password is correct. If the user has correctly logged in, write the username to session and send the user to the `index` action.

Hints:

- Use the built in `$this->User->findByUsername($your_username_variable_here)` to search for the user in the database

- Write the user's name to `Session` with `$this->Session->write('user', $your_username_variable_here)`

Index action

The `index` action should check to see if the user's name has been written to the session. If the user's name has been written to the session, pull that information from the database and present the user with a customized greeting. If the user has not logged in, direct him to the `login` action.

Logout

The `logout` action should delete the user's username from the session and forward the user to the `login` action.

Bonus

Modify the `register` action to automatically log the user into the system and forward the user to the `index` action. Modify the `register` and `login` actions to use hashed passwords, rather than saving your passwords in the database as plain text.

Don't worry too much if you get stuck. Part 2 provides sample solutions to these problems. Then you'll jump right in and build out the Tor application product gallery.

Happy coding!

Section 7. Summary

This tutorial taught you how to install and configure CakePHP, the basics of 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. Part 2 covers writing plug-ins for your application and using CakePHP's access-control lists (ACLs).

Downloads

Description	Name	Size	Download method
Part 1 source code	os-php-cake1.source.zip	3KB	HTTP

Information about download methods

Resources

Learn

- Visit CakePHP.org to learn more about it.

- The CakePHP API has been thoroughly documented. This is the place to get the most up-to-date documentation for CakePHP.

- The Cookbook is Cake's Wiki-driven user manual, and contains lots of helpful information.

- 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 the Wikipedia Model-View-Controller.

- Here is more useful background on the Model-View-Controller.

- Here's a whole list of different types of software design patterns.

- Read 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.

- 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.

- 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 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

- 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

Duane O'Brien

> Duane O'Brien has been a technological Swiss Army knife since the Oregon Trail was text only. His favorite color is sushi. He has never been to the moon.