

The Official Guide

CakePHP

in Action

Duane O'Brien

MEAP

Unedited Draft

 MANNING





**MEAP Edition
Manning Early Access Program**

Copyright 2007 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Part 1 Grasping the basics

- Chapter 1 Introducing CakePHP
- Chapter 2 Scaffolding and Baking
- Chapter 3 Models—the basic stuff
- Chapter 4 Controllers—the basic stuff
- Chapter 5 Views—the basic stuff
- Chapter 6 Global functions and constants

Part 2 Delving deeper

- Chapter 7 TBD
- Chapter 8 TBD
- Chapter 9 Helpers
- Chapter 10 Components
- Chapter 11 Data validation
- Chapter 12 Securing your CakePHP application.
- Chapter 13 Plugins

Part 3 Advanced topics

- Chapter 14 Access control
- Chapter 15 Caching
- Chapter 16 AJAX
- Chapter 17 Models—advanced
- Chapter 18 Controllers—advanced
- Chapter 19 Views—advanced
- Chapter 20 Command-line Cake
- Chapter 21 Mining the Bakery

Appendices

- Appendix A Installation and configuration
- Appendix B Troubleshooting CakePHP

1

Introducing CakePHP

Who doesn't like cake? You have a piece at least once a year on your birthday, right? Weddings, anniversaries, parties, it seems like whenever it's time to celebrate, it's time for cake. Cake just means it's party time. Party hats, chicken dance, wrapping paper, the whole nine yards.

The same is true of CakePHP: once you start using it, you're going to feel like you're at a party (chicken dance optional).

This book has been officially endorsed and supported by the Cake Software Foundation as the official guide to using CakePHP 1.2. It is assumed that you have a working knowledge of PHP, have written a couple of applications, and are eager to dive in and try your hand at using a framework - specifically CakePHP. Don't worry if you don't have every PHP function committed to memory, or if you can't precisely remember the right format to pass to the date function for English ordinal suffixes. As long as you're comfortable working with PHP and know how to solve your own syntactic difficulties, you're good to go.

If you're new to frameworks, or think that MVC is that evil program from Tron, then you probably should take the time to learn a little about what a framework is, what MVC means, and how CakePHP is different before you proceed. Even if you're already familiar with frameworks and the MVC Software Architecture, you should take the time to learn some of the particulars about CakePHP. We'll cover all that in this chapter, then you can jump right in. There are tons of frameworks out there. In almost every language you can find several options. But this book isn't about any of those other frameworks. This book is about CakePHP. But what is CakePHP and what are its origins?

It's 2004. PHP 5 has just been released to the public. People had been complaining for years that PHP didn't support good object oriented programming, or MVC. With the object support provided by PHP 5, these complaints were set to be answered.

But then something interesting happened. Ruby on Rails hit the scene, and it made a big splash. With the light approach and scaffolding provided by Ruby on Rails, people started looking at web development with a fresh perspective. I remember watching the demo video, seeing Ruby on Rails in action for the first time. It just looked too easy. And in a way it was. Anyone could use Ruby on Rails. All they had to do was start using Ruby as their development language.

Many people did. But many other people said "Those are great ideas. I want to do something like that in the language I love." And so it was that in 2005, CakePHP was born. The idea was not to simply port Ruby on Rails to PHP. Rather, CakePHP was intended to take the strength of the better ideas presented by Ruby on Rails, and combine them with the best parts of PHP, in order to create a world class framework in language many people had grown to enjoy. CakePHP is open source, distributed under the MIT license. But more on that later. Let's step back and look at frameworks a little more generally.

1.1 What is a Framework?

Put aside the fact that you're reading a computer book. Forget everything that you know about writing code. When you see the word 'framework' what do you see in your head? Some 2x4 framing in the shape of a house? Those Polygon wireframes from computer animations? Maybe you see a set of rules intended to govern a conversation. While they might each have very different contexts, each of these things is a framework. If a contractor wants to build several houses of a particular shape, the same framework will be used for each house. If an animator wants to create several robots that move and look similar, the same framework would be used for each robot. If two parties agree to a series of debates, they will typically agree on a framework that will structure each debate.

Generally, a framework sets out the basic shape of something, be it a house, a computer generated robot, or a formal discussion. While the finished house, animated robot or discussion will look significantly different than the framework upon which it was built, one can typically see the shape of the framework in the finished product.

Apply this same thought process to an application framework, and immediately you begin to understand what an application framework is all about. Most web applications, regardless of their complexity, have several things in common. At the fundamental level, there is a need to accept input from the user, act upon that input, and present information to the user. The information presented to the user must be formatted in a particular fashion (for example, a

browser based application need to present information in a format that the browser can render). Often (but not always), the application needs to remember and store what the user did last, or information the user has already submitted.

But beyond the fundamental level there are typically many more similarities. Most applications need to validate incoming data to ensure that its correctly submitted. An application typically requires some sort of interface to a database or other storage construct to keep a record of submitted information long term. Many applications have different features available to different classes of users, therefore requiring some form of access control. Often it is necessary to send emails to the users, either to assist them in registration or notify them of important information. Many applications require the ability to perform tasks through multiple interfaces, such as the ability to accept browser input and command line input (for example, cron jobs). In some circumstances, an application might need to cache data to improve performance.

All of these things represent different tasks that are typical of a web based application. If you've written many applications you've probably already written code to perform some of these tasks, in some cases many times over. How many times have you written code to read or write from the database? Or clean up user submitted data? How may times have you found yourself copying code from an application you wrote earlier, rather than writing it again?

A Web Application Framework, such as CakePHP, aims to provide a structure that addresses these needs for you, so that you can spend more time writing the application code, and less time writing the same database access, presentation, data integrity and validation code.

1.2 What is Model-View-Controller?

The Model-View-Controller architecture breaks your application into three distinct layers called, not surprisingly, the Model, the View and the Controller. Each of these layers deal with distinct tasks within the application, corresponding roughly to Data, Presentation and Logic. It's no coincidence that the logo for CakePHP shows a three layered cake. This simple iconographic representation tells you volumes about MVC.

1.2.1 The Model

The word Model is used to refer to the part of your application that deals with data. This typically means the interface with the database, but don't make the mistake of limiting the

Model to just that. Other data related tasks, such as Data Validation, are part of the Model. The Model is that bottom layer of cake upon which everything else depends.

It is important to remember that Model is a concept. You will implement this concept using one or more files, and these files will be typically referred to as Models. Sometimes that can get confusing, but it's a case where context is everything. Try to keep the Model as a concept separate and distinct from the individual files. To use an analogy, it is similar to the difference between "The development team" and "Beuford the PHP Engineer." I know this may seem a bit pedantic, but it is important to mark the distinction now. There may be times when you are working on code that is part of The Model (the layer of the application that manages your data), but that code is not in a file you would normally refer to as A Model (a file that handles data for a particular table).

1.2.2 The View

The word View is used to refer to the part of your application that deals with presentation. This is where you format anything and everything that's presented to the user. The HTML for a specific page, the XML for an RSS feed, the Javascript that drives your applications AJAX, all of this comes from the View. The View is the top layer of your cake, covered in pretty icing.

As with the Model, you'll need to keep in mind that View is a concept that you will implement using one or more files. Some of these files, specifically the templates that are rendered for a particular action, are called Views. But there are many files that are part of The View (conceptually) that are not A View (a file). Examples of such files include Helpers, CSS, Javascript files, and images.

1.2.3 The Controller

The word Controller is used to refer to the part of your application that processes information. This is where your application or business logic lives. When the user submits data, and decisions are made based on that data, those decisions are made in the Controller. As with the Model and View, you should take care to differentiate between The Controller (conceptually) and A Controller (a specific file). The Controller is that middle layer in your cake. It's a substantial layer, giving your cake volume and flavor.

While the CakePHP icon shows three equally sized layers, the Controller layer will be disproportionately larger as it holds all of your application logic. Don't worry. That's normal. Symmetry isn't everything.

1.2.4 Example

These examples might seem fairly abstract or difficult to grasp if you've never worked with them before. Consider the following crude example:

```
<?php

mysql_connect($params);

?>
<html>
<head>
<title>Submit Your Email</title>
</head>
<body>
<h1>
<?php

if (isset($_POST) && !empty($_POST)) {
    $sql = 'insert into users
            (firstname, lastname, email)
            values
            ("'. $_POST['firstname'] .'", "'. $_POST['lastname'] .'", "'.
$_POST['email'] .'")';
    mysql_query($sql);
    if (!isset(mysql_error())) {
        echo 'Submitted!';
    } else {
        echo 'Failed. Please verify your data.';
    }
} else {
    echo 'All Fields Are Required';
}
?>
</h1>
<form method='get'>
<input name='firstname'> <input name='lastname'><br />
<input name='email'><br />
<input type='submit'>
</form>
</body>
</html>
```

In this page all the tasks jumbled together. The database code, the logic (such as it is) and the HTML code are all tangled together, making it difficult to maintain, especially by more than one person. Consider the potential problems with sending this file to a design team, that knows nothing about PHP, for the purpose of beautifying the layout. What happens when your application consists of many files like this, and you decide you want to use a different database?

Now consider breaking the same page into the three files representing the basic Model-view-controller components. Your Model (model.php) might look like this:

```
<?php

mysql_connect($params);

function saveUser($first, $last, $email) {
    $sql = 'insert into users
           (firstname, lastname, email)
           values
           ("'. $first .'", "'. $last .'", "'. $email .'")';
    mysql_query($sql);
    if (!isset(mysql_error())) {
        return true;
    }
    return false;
}
?>
```

Your Controller would read the input, save if necessary, and set any data to be used in the view:

```
<?php

include('model.php')

if (isset($_POST) && !empty($_POST)) {
    if (saveUser($_POST['firstname'], $_POST['lastname'],
$_POST['email'])) {
        $message = 'Submitted';
    } else {
        $message = 'Failed. Please verify your data.';
    }
} else {
    $message = 'All fields are required.';
}

include('view.php');

?>
```

Finally, your View (view.php) would contain almost entirely HTML code, making it easier to maintain or hand off to someone for design purposes.

```
<html>
<head>
<title>Submit Your Email</title>
</head>
<body>
<h1><?php echo $message ?></h1>
<form method='get'>
<input name='firstname'> <input name='lastname'><br />
<input name='email'><br />
<input type='submit'>
</form>
</body>
</html>
```

Even this crude example shows the benefit of taking the MVC approach when designing your application. Need to redesign the look and feel? Edit the View and you're done. Want to use a different database? Change the Model accordingly. And now that all the logic is in the Controller, you have much less code to wade through to determine why an application isn't behaving as expected.

Don't make the mistake of equating 'Framework' with 'MVC'. A Web Application Framework is a set of files designed to provide a structure upon which you can build your application. Model-view-controller is a software architecture, an approach to building an application. There are frameworks that follow different architectural models. You can build an MVC application without using a framework (in fact, you just did above.) CakePHP is a framework that helps you build applications according to MVC design.

1.3 What makes CakePHP Different?

There are many frameworks out there, for every language. For PHP alone there are at least a dozen. When faced with so many choices, sometimes it can be difficult to know for sure where to start in picking out a framework. In order to understand what makes CakePHP different from other frameworks, you need to understand some of the concepts and features involved.

1.3.1 MVC Push vs Pull

There are two different ways of getting data into the View layer of an MVC application: Push and Pull. In an MVC 'Pull' application, a Controller might process some information, making data available to the View by using some specific variables or objects that are available to

the View. By contrast, in an MVC 'Pull' application, the View has access to a wider range of data via a Controller's actions. The View makes a request of the Controller, information is processed and rendered for output. It's a subtle, but important difference. CakePHP follows the Push approach. Your Controllers will process information and make data available to the Views.

1.3.2 Object Relational Mapping

A database stores data. When you query a database, what you get back is data. It might seem like I'm stating the obvious here, but work with me. In object-oriented programming, if you want to work with the data as an object, you have to turn that data into an object. You've probably never even thought about this before, because at a basic level, PHP does a lot of that for you. You learned that the first time you tried to echo the result of `mysql_query` or `mysql_fetch_row` - you don't get your data, you get a Resource ID# that corresponds to the object holding your data. The act of turning the data into an object is referred to as Object Relational Mapping or ORM. CakePHP goes further by using a design pattern known as Active Record.

The Active Record pattern involves creating a class as a wrapper for a database table. A single instance of an object correlates to a single row in the database. For example, if you have a database table called 'posts', CakePHP will have a class that describes an object of type 'Post.' When you read a row out of the posts database, you get a Post object that corresponds to the row. When you update the Post object, the row in the database is updated. Active Record is a popular approach to handling data in the database.

1.3.3 AJAX Support

Most frameworks provide some kind of structure for simplifying the handling of AJAX requests. Usually this is accomplished by leveraging existing third party libraries, like jQuery, Prototype, script.aculo.us and so forth. CakePHP provides simplified handling of typical AJAX tasks via Prototype, as well as dynamic Javascript effects via script.aculo.us. Neither of these libraries are included when you install CakePHP - you'll have to install them yourself. But it's a cinch. You'll learn more about this in the AJAX Chapter.

1.3.4 Internationalization and Localization

You've probably seen the abbreviations **i18n** and **L10n** before. These abbreviations refer (respectively) to Internationalization and Localization. While usually referred to in tandem, both refer to two distinct concepts, though they are easily confused. Internationalization

refers to an application's ability to support multiple languages, whereas Localization refers more specifically to customizing translations so that they are appropriate for the audience. In other words, making sure your application can handle double byte characters would be Internationalization, while being able to customize labels and button values would be Localization. This is somewhat of an oversimplification of the tasks involved, but it should give you a general idea. Most mature frameworks support i18n and L10n in some fashion. CakePHP is no exception. This will be covered in detail in a later chapter.

1.3.5 Open Source

It may seem patently obvious to state that CakePHP is open source. Of course the framework is open source. Ultimately CakePHP is just a bunch of PHP files. But don't roll your eyes just yet. Just because you can see it doesn't mean you can do anything you want with it. Which raises the bigger question: What License?

CakePHP is developed under the MIT license. A complete dissection of the MIT License is outside the scope of this book. You'll get a copy of the license when you download the code. It is a very permissive license that says, in essence, "You can do whatever you want to with this, as long as you do not remove the copyright and claim CakePHP to be your own work. And you can't sue us if it doesn't work for you."

In other words, CakePHP is free. You don't have to pay for it, you can use it any way you want.

1.4 That's All Well And Good, But...

Very few people buy a particular mobile phone because they like the look of the circuit board inside. If you buy a magazine, it's unlikely that you choose that specific magazine because you like the feel of the paper or the style of the binding. People typically pick out their phone based on the features it provides. You buy magazines because you want the content. In the same vein, developers seldom pick out a technology to investigate based on any one particular. We choose technologies because they appeal to us at some level. In this context, you may not care about the deeper technical particulars of CakePHP. "Who cares what makes CakePHP 'different.' What makes CakePHP Cool?"

1.4.1 CakePHP is Fast

CakePHP is cool because it's fast. It doesn't take very long to figure this out. The first time you build an application in CakePHP, it may seem almost too easy. You may find yourself hesitating and asking yourself "... is that it?" That can be a little disorienting at first, but don't let it throw you off track. We've all gotten used to the idea that building an application either means spending a lot of time building and constantly adapting a structure or framework, or trying to work out how to get the framework you're stuck with to do what you want. CakePHP does an excellent job of providing you with the tools you need to accomplish your goals, and then getting out of your way so you spend more time writing your application.

1.4.2 CakePHP is Easy

In any learning exercise, there is a learning curve - some amount of time spent learning before you can start putting your knowledge to practical use. Shallow learning curves usually have limited payoff. It takes less than a minute to learn how to play Tic-Tac-Toe, but apart from whiling away some spare time or teaching NORAD computers about war, Tic-Tac-Toe has little practical use. Steep learning curves should result in big payoffs. Going through medical school takes a lot of time and effort, but the knowledge gained from the process allows you to save lives.

However, frequently steep learning curves can yield small payoffs. That's a common problem, especially when dealing with frameworks. You spend so much time trying to learn the specific quirks of the framework that you begin to question what the final payoff really is. You may have picked up a framework before, only to put it down because it just seemed to require more overhead than necessary.

But sometimes you get lucky. Once in a while, you find something that's easy to use but has a big payoff.

CakePHP is easy. Yes, there is a learning curve, albeit a shallow one. You don't spend very much time on the learning curve before you can start really putting the knowledge to use.

1.4.3 CakePHP is Robust

CakePHP provides you with a lot of functionality, enough that you can do anything you typically need to do in the context of a web application. If you need something unique, particular or unusual, CakePHP provides you with an interface to plug in anything you might need that isn't already provided. Want Caching? It's there. Testing framework? Yep, it's got that too. Want options for session management? No problem. I could go on, but I will

start to sound like a cheerleader. Suffice it to say, if you can't do it with CakePHP, you probably don't need to do it at all.

1.5 Summary

Ok, so you've got a handle on Frameworks. You understand the basics of MVC. And more importantly, you know why CakePHP is awesome. You can almost taste the cake. You may have even skipped ahead and run through the installation already. If you haven't gone through the installation process (or if you had problems), check out Appendix A - Installation. There you will find everything you need to get CakePHP up and running on your favorite platform. Now that you have a handle on what CakePHP is and where it came from, you are ready to jump into Chapter 2, where you'll get some hands on experience working on the test application.