What is CakePHP? Why use it?

CakePHP is a free, open-source, rapid development framework for PHP. It's a foundational structure for programmers to create web applications. Its primary goal is to enable you to work in a structured and rapid manner-without loss of flexibility.

CakePHP takes the monotony out of web development. It provides you with all the tools you need to get started coding and what you need to get done: the logic specific to your application.

CakePHP has an active developer team and community, bringing great value to the project. In addition to keeping you from wheel-reinventing, using CakePHP means your application's core is well tested and is being constantly improved.

Here's a quick list of features you'll enjoy when using CakePHP:

- Active, friendly Official CakePHP discussion group
- Flexible licensing
- Compatible with versions PHP 5.2.8 and greater
- Integrated CRUD for database interaction
- Application scaffolding
- Code generation
- MVC architecture
- Request dispatcher with clean, custom URLs and routes
- Built-in validation
- Fast and flexible templating (PHP syntax, with helpers)
- View helpers for AJAX, JavaScript, HTML forms and more
- Email, cookie, security, session, and request handling Components
- Flexible ACL
- Data sanitization
- Flexible caching
- Localization
- Works from any web site directory, with little to no Apache configuration involved

Understanding Model-View-Controller

CakePHP follows the MVC software design pattern. Programming using MVC separates your application into three main parts:

The Model layer

The Model layer represents the part of your application that implements the business logic. It is responsible for retrieving data and converting it into meaningful concepts for your application. This includes processing, validating, associating or other tasks related to handling data.

At a first glance, Model objects can be looked at as the first layer of interaction with any database you might be using for your application. But in general they stand for the major concepts around which you implement your application.

In the case of a social network, the Model layer would take care of tasks such as saving the user data, saving friends' associations, storing and retrieving user photos, finding suggestions for new friends, etc. The model objects can be thought as "Friend", "User", "Comment", or "Photo".

The View layer

The View renders a presentation of modeled data. Being separated from the Model objects, it is responsible for using the information it has available to produce any presentational interface your application might need.

For example, as the Model layer returns a set of data, the view would use it to render a HTML page containing it, or a XML formatted result for others to consume.

The View layer is not only limited to HTML or text representation of the data. It can be used to deliver a wide variety of formats depending on your needs, such as videos, music, documents and any other format you can think of.

The Controller layer

The Controller layer handles requests from users. It is responsible for rendering a response with the aid of both the Model and the View layer.

A controller can be seen as a manager that ensures that all resources needed for completing a task are delegated to the correct workers. It waits for petitions from clients, checks their validity according to authentication or authorization rules, delegates data fetching or processing to the model, selects the type of presentational data that the clients are accepting, and finally delegates the rendering process to the View layer.

CakePHP request cycle

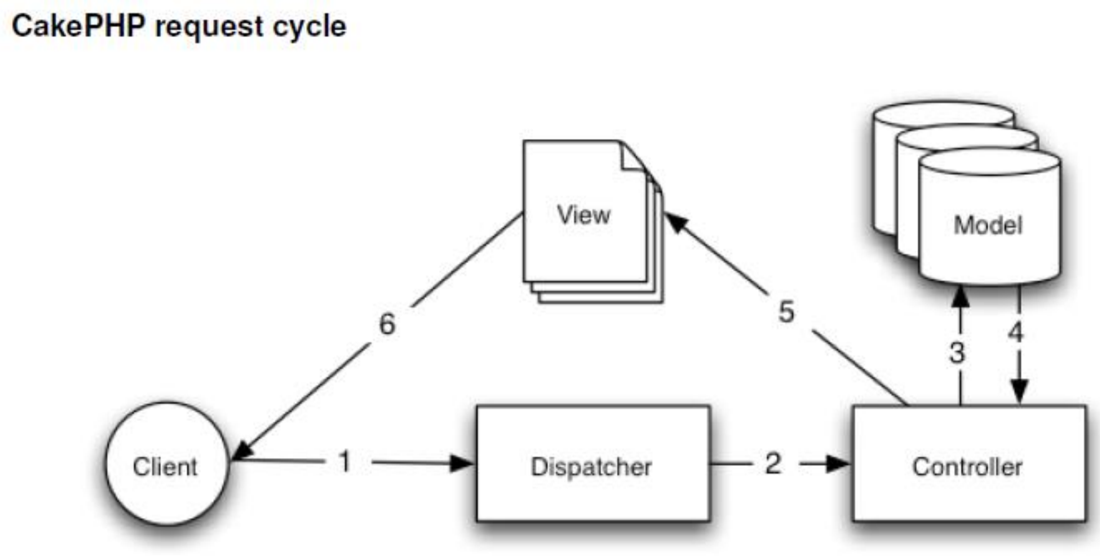


Figure: 1: A typical MVC Request in CakePHP

The typical CakePHP request cycle starts with a user requesting a page or resource in your application. This request is first processed by a dispatcher which will select the correct controller object to handle it.

Once the request arrives at the controller, it will communicate with the Model layer to process any data-fetching or -saving operation that might be needed. After this communication is over, the controller will proceed to delegate to the correct view object the task of generating output resulting from the data provided by the model.

Finally, when this output is generated, it is immediately rendered to the user.

Almost every request to your application will follow this basic pattern. We'll add some details later on which are specific to CakePHP, so keep this in mind as we proceed.

Benefits

Why use MVC? Because it is a tried and true software design pattern that turns an application into a maintainable, modular, rapidly developed package. Crafting application tasks into separate models, views, and controllers makes your application very light on its feet. New features are easily added, and new faces on old features are a snap. The modular and separate design also allows developers and designers to work simultaneously, including the ability to rapidly prototype. Separation also allows developers to make changes in one part of the application without affecting the others.

Where to Get Help

The Official CakePHP website

http://www.cakephp.org

The Official CakePHP website is always a great place to visit. It features links to oft-used developer tools, screencasts, donation opportunities, and downloads.

The Cookbook

http://book.cakephp.org

This manual should probably be the first place you go to get answers. As with many other open source projects, we get new folks regularly. Try your best to answer your questions on your own first. Answers may come slower, but will remain longer - and you'll also be lightening our support load. Both the manual and the API have an online component.

The Bakery

http://bakery.cakephp.org

The CakePHP Bakery is a clearing house for all things regarding CakePHP. Check it out for tutorials, case studies, and code examples. Once you're acquainted with CakePHP, log on and share your knowledge with the community and gain instant fame and fortune.

The API

http://api.cakephp.org/

Straight to the point and straight from the core developers, the CakePHP API (Application Programming Interface) is the most comprehensive documentation around for all the nitty gritty details of the internal workings of the framework. It's a straight forward code reference, so bring your propeller hat.

The Test Cases

If you ever feel the information provided in the API is not sufficient, check out the code of the test cases provided with CakePHP. They can serve as practical examples for function and data member usage for a class.:

```
lib/Cake/Test/Case
```

Stackoverflow

http://stackoverflow.com/

Tag your questions with `cakephp` and the specific version you are using to enable existing users of stackoverflow to find your questions.

Installation

CakePHP is fast and easy to install. The minimum requirements are a webserver and a copy of CakePHP, that's it! While this manual focuses primarily on setting up on Apache (because it's the most commonly used), you can configure CakePHP to run on a variety of web servers such as LightHTTPD or Microsoft IIS.

Requirements

- HTTP Server. For example: Apache. mod_rewrite is preferred, but by no means required.
- PHP 5.2.8 or greater.

Technically a database engine isn't required, but we imagine that most applications will utilize one. CakePHP supports a variety of database storage engines:

- MySQL (4 or greater)
- PostgreSQL
- Microsoft SQL Server
- SQLite

Note

All built-in drivers require PDO. You should make sure you have the correct PDO extensions installed.

License

CakePHP is licensed under the MIT license. This means that you are free to modify, distribute and republish the source code on the condition that the copyright notices are left intact. You are also free to incorporate CakePHP into any commercial or closed source application.

Downloading CakePHP

There are two main ways to get a fresh copy of CakePHP. You can either download an archived copy (zip/tar.gz/tar.bz2) from the main website, or check out the code from the git repository.

To download the latest major release of CakePHP, visit the main website http://cakephp.org and follow the “Download” link.

All current releases of CakePHP are hosted on GitHub. GitHub houses both CakePHP itself as well as many other plugins for CakePHP. The CakePHP releases are available at GitHub tags.

Alternatively you can get fresh off the press code, with all the bug-fixes and up to the minute enhancements. These can be accessed from GitHub by cloning the GitHub repository:

```
git clone git://github.com/cakephp/cakephp.git
```

Permissions

CakePHP uses the `app/tmp` directory for a number of different operations. A few examples would be Model descriptions, cached views and session information.

As such, make sure the directory `app/tmp` and all its subdirectories in your CakePHP installation are writable by the web server user.

One common issue is that the app/tmp directories and subdirectories must be writable both by the web server and the command line user. On a UNIX system, if your web server user is different from your command line user, you can run the following commands just once in your project to ensure that permissions will be setup properly:

```
HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | head -1 |
cut -d¥  -f1`
setfacl -R -m u:${HTTPDUSER}:rwx app/tmp
setfacl -R -d -m u:${HTTPDUSER}:rwx app/tmp
```

Setup

Setting up CakePHP can be as simple as slapping it in your web server’s document root, or as complex and flexible as you wish. This section will cover the three main installation types for CakePHP: development, production, and advanced.

- Development: easy to get going, URLs for the application include the CakePHP installation directory name, and less secure.
- Production: Requires the ability to configure the web server’s document root, clean URLs, very secure.
- Advanced: With some configuration, allows you to place key CakePHP directories in different parts of the filesystem, possibly sharing a single CakePHP core library folder amongst many CakePHP applications.

Development

A development installation is the fastest method to setup CakePHP. This example will help you install a CakePHP application and make it available at http://www.example.com/cake_2_0/. We assume for the purposes of this example that your document root is set to `/var/www/html`.

Unpack the contents of the CakePHP archive into `/var/www/html`. You now have a folder in your document root named after the release you've downloaded (e.g. cake_2.0.0). Rename this folder to cake_2_0. Your development setup will look like this on the file system:

```
/var/www/html/
    cake_2_0/
        app/
        lib/
        plugins/
        vendors/
        .htaccess
        index.php
        README
```

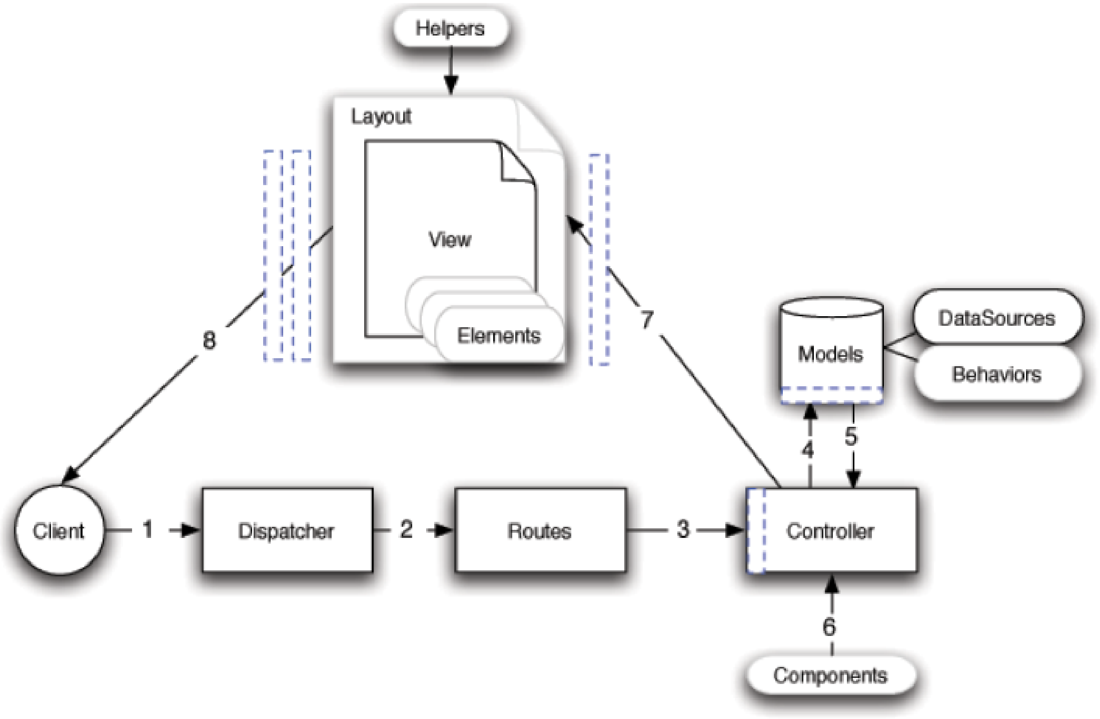
If your web server is configured correctly, you should now find your CakePHP application accessible at http://www.example.com/cake_2_0/.

Advanced Installation and URL Rewriting

- Advanced Installation
- URL Rewriting

A Typical CakePHP Request

We’ve covered the basic ingredients in CakePHP, so let’s look at how objects work together to complete a basic request. Continuing with our original request example, let’s imagine that our friend Ricardo just clicked on the “Buy A Custom Cake Now!” link on a CakePHP application’s landing page.



Flow diagram showing a typical CakePHP request

Figure: 2. Typical CakePHP Request.

Black = required element, Gray = optional element, Blue = callback

1. Ricardo clicks the link pointing to http://www.example.com/cakes/buy, and his browser makes a request to your web server.
2. The Router parses the URL in order to extract the parameters for this request: the controller, action, and any other arguments that will affect the business logic during this request.
3. Using routes, a request URL is mapped to a controller action (a method in a specific controller class). In this case, it’s the buy() method of the CakesController. The controller’s beforeFilter() callback is called before any controller action logic is executed.
4. The controller may use models to gain access to the application’s data. In this example, the controller uses a model to fetch Ricardo’s last purchases from the database. Any applicable model callbacks, behaviors, and DataSources may apply during this operation. While model

usage is not required, all CakePHP controllers initially require at least one model.
5. After the model has retrieved the data, it is returned to the controller. Model callbacks may apply.
6. The controller may use components to further refine the data or perform other operations (session manipulation, authentication, or sending emails, for example).
7. Once the controller has used models and components to prepare the data sufficiently, that data is handed to the view using the controller's set() method. Controller callbacks may be applied before the data is sent. The view logic is performed, which may include the use of elements and/or helpers. By default, the view is rendered inside a layout.
8. Additional controller callbacks (like afterFilter) may be applied. The complete, rendered view code is sent to Ricardo's browser.

CakePHP Conventions

We are big fans of convention over configuration. While it takes a bit of time to learn CakePHP's conventions, you save time in the long run: by following convention, you get free functionality, and you free yourself from the maintenance nightmare of tracking config files. Convention also makes for a very uniform system development, allowing other developers to jump in and help more easily.

CakePHP's conventions have been distilled from years of web development experience and best practices. While we suggest you use these conventions while developing with CakePHP, we should mention that many of these tenets are easily overridden – something that is especially handy when working with legacy systems.

Controller Conventions

Controller class names are plural, CamelCased, and end in `Controller`. `PeopleController` and `LatestArticlesController` are both examples of conventional controller names.

The first method you write for a controller might be the `index()` method. When a request specifies a controller but not an action, the default CakePHP behavior is to execute the `index()` method of that controller. For example, a request for http://www.example.com/apples/ maps to a call on the `index()` method of the `ApplesController`, whereas http://www.example.com/apples/view/ maps to a call on the `view()` method of the `ApplesController`.

You can also change the visibility of controller methods in CakePHP by prefixing controller method names with underscores. If a controller method has been prefixed with an underscore, the method will not be accessible directly from the web but is available for internal use. For example:

```
class NewsController extends AppController {

    public function latest() {
        $this->_findNewArticles();
    }
    protected function _findNewArticles() {
        // Logic to find latest news articles
    }
}
```

While the page http://www.example.com/news/latest/ would be accessible to the user as usual, someone trying to get to the page http://www.example.com/news/_findNewArticles/ would get an error, because the method is preceded with an underscore. You can also use PHP's visibility keywords to indicate whether or not a method can be accessed from a URL. Non-public methods cannot be accessed.

URL Considerations for Controller Names

As you've just seen, single word controllers map easily to a simple lower case URL path. For example, `ApplesController` (which would be defined in the file name 'ApplesController.php') is accessed from http://example.com/apples.

Multiple word controllers *can* be any 'inflected' form which equals the controller name so:

- /redApples
- /RedApples
- /Red_apples
- /red_apples

will all resolve to the index of the RedApples controller. However, the convention is that your URLs are lowercase and underscored, therefore /red_apples/go_pick is the correct form to access the `RedApplesController::go_pick` action.

For more information on CakePHP URLs and parameter handling, see Routes Configuration. If you have files/directories in your `/webroot` directory that share a name with one of your routes/controllers, you will be directed to the file/directory and, not to your controller.

File and Class Name Conventions

In general, filenames match the class names, which are CamelCased. So if you have a class **MyNiftyClass**, then in CakePHP, the file should be named **MyNiftyClass.php**. Below are examples of how to name the file for each of the different types of classes you would typically use in a CakePHP application:

- The Controller class **KissesAndHugsController** would be found in a file named **KissesAndHugsController.php**
- The Component class **MyHandyComponent** would be found in a file named **MyHandyComponent.php**
- The Model class **OptionValue** would be found in a file named **OptionValue.php**
- The Behavior class **EspeciallyFunkableBehavior** would be found in a file named **EspeciallyFunkableBehavior.php**
- The View class **SuperSimpleView** would be found in a file named **SuperSimpleView.php**

- The Helper class BestEverHelper would be found in a file named BestEverHelper.php

Each file would be located in the appropriate folder in your app folder.

Model and Database Conventions

Model class names are singular and CamelCased. Person, BigPerson, and ReallyBigPerson are all examples of conventional model names.

Table names corresponding to CakePHP models are plural and underscored. The underlying tables for the above mentioned models would be `people`, `big_people`, and `really_big_people`, respectively.

You can use the utility library Inflector to check the singular/plural of words. See the Inflector for more information.

Field names with two or more words are underscored: first_name.

Foreign keys in hasMany, belongsTo or hasOne relationships are recognized by default as the (singular) name of the related table followed by _id. So if a Baker hasMany Cake, the cakes table will refer to the bakers table via a baker_id foreign key. For a table like category_types whose name contains multiple words, the foreign key would be category_type_id.

Join tables, used in hasAndBelongsToMany (HABTM) relationships between models, should be named after the model tables they will join, arranged in alphabetical order (apples_zebras rather than zebras_apples).

All tables with which CakePHP models interact (with the exception of join tables) require a singular primary key to uniquely identify each row. If you wish to model a table that does not already have a single-field primary key, CakePHP's convention is that a single-field primary key is added to the table. You must add a single-field primary key if you want to use that table's model.

CakePHP does not support composite primary keys. If you want to directly manipulate your join table data, use direct query calls or add a primary key to act on it as a normal model. For example:

```
CREATE TABLE posts_tags ( id INT(10) NOT NULL AUTO_INCREMENT, post_id
INT(10) NOT NULL, tag_id INT(10) NOT NULL, PRIMARY KEY(id));
```

Rather than using an auto-increment key as the primary key, you may also use char(36). CakePHP will then use a unique 36 character UUID (String::uuid) whenever you save a new record using the Model::save method.

View Conventions

View template files are named after the controller functions they display, in an underscored form. The getReady() function of the PeopleController class will look for a view template in /app/View/People/get_ready.ctp.

The basic pattern is /app/View/Controller/underscored_function_name.ctp.

By naming the pieces of your application using CakePHP conventions, you gain functionality without the hassle and maintenance tethers of configuration. Here's a final example that ties the conventions together:

- Database table: "people"
- Model class: "Person", found at /app/Model/Person.php
- Controller class: "PeopleController", found at /app/Controller/PeopleController.php
- View template, found at /app/View/People/index.ctp

Using these conventions, CakePHP knows that a request to http://example.com/people/ maps to a call on the index() function of the PeopleController, where the Person model is automatically available (and automatically tied to the 'people' table in the database), and renders to a file. None of these relationships have been configured by any means other than by creating classes and files that you'd need to create anyway.

Now that you've been introduced to CakePHP's fundamentals, you might try a run through the Blog Tutorial to see how things fit together.

CakePHP Folder Structure

After you've downloaded and extracted CakePHP, these are the files and folders you should see:

- app
- lib
- vendors
- plugins
- .htaccess
- index.php
- README

You'll notice three main folders:

- The *app* folder will be where you work your magic: it's where your application's files will be placed.
- The *lib* folder is where we've worked our magic. Make a personal commitment **not** to edit files in this folder. We can't help you if you've modified the core. Instead, look into modifying Application Extensions.
- Finally, the *vendors* folder is where you'll place third-party PHP libraries you need to use with your CakePHP applications.

The App Folder

CakePHP's *app* folder is where you will do most of your application development. Let's look a little closer at the folders inside *app*.

Config

Holds the (few) configuration files CakePHP uses. Database connection details, bootstrapping, core configuration files and more should be stored here.

Console

Contains the console commands and console tasks for your application. This directory can also contain a `Templates` directory to customize the output of bake. For more information see Console and Shells.

Controller

Contains your application's controllers and their components.

Lib

Contains libraries that do not come from 3rd parties or external vendors. This allows you to separate your organization's internal libraries from vendor libraries.

Locale

Stores string files for internationalization.

Model

Contains your application's models, behaviors, and datasources.

Plugin

Contains plugin packages.

Test

This directory contains all the test cases and test fixtures for your application. The `Test/Case` directory should mirror your application and contain one or more test cases per class in your application. For more information on test cases and test fixtures, refer to the Testing documentation.

tmp

This is where CakePHP stores temporary data. The actual data it stores depends on how you have CakePHP configured, but this folder is usually used to store model descriptions, logs, and sometimes session information.

Make sure that this folder exists and is writable, or the performance of your application will be severely impacted. In debug mode, CakePHP will warn you if the folder is absent or not writable.

Vendor

Any third-party classes or libraries should be placed here. Doing so makes them easy to access using the App::import('vendor', 'name') function. Keen observers will note that this seems redundant, as there is also a *vendors* folder at the top level of our directory structure. We'll get into the differences between the two when we discuss managing multiple applications and more complex system setups.

View

Presentational files are placed here: elements, error pages, helpers, layouts, and view files.

webroot

In a production setup, this folder should serve as the document root for your application. Folders here also serve as holding places for CSS stylesheets, images, and JavaScript files.

CakePHP Structure

CakePHP features Controller, Model, and View classes, but it also features some additional classes and objects that make development in MVC a little quicker and more enjoyable. Components, Behaviors, and Helpers are classes that provide extensibility and reusability to quickly add functionality to the base MVC classes in your applications. Right now we'll stay at a higher level, so look for the details on how to use these tools later on.

Application Extensions

Controllers, helpers and models each have a parent class you can use to define application-wide changes. AppController (located at `/app/Controller/AppController.php`), AppHelper (located at `/app/View/Helper/AppHelper.php`) and AppModel (located at `/app/Model/AppModel.php`) are great places to put methods you want to share between all controllers, helpers or models.

Although routes aren't classes or files, they play a role in requests made to CakePHP. Route definitions tell CakePHP how to map URLs to controller actions. The default behavior assumes that the URL `/controller/action/var1/var2` maps to Controller::action($var1, $var2), but you can use routes to customize URLs and how they are interpreted by your application.

Some features in an application merit packaging as a whole. A plugin is a package of models, controllers and views that accomplishes a specific purpose that can span multiple applications. A user management system or a simplified blog might be a good fit for CakePHP plugins.

Controller Extensions ("Components")

A Component is a class that aids in controller logic. If you have some logic you want to share between controllers (or applications), a component is usually a good fit. As an example, the core EmailComponent class makes creating and sending emails a snap. Rather than writing a controller method in a single controller that performs this logic, you can package the logic so it can be shared.

Controllers are also fitted with callbacks. These callbacks are available for your use, just in case you need to insert some logic between CakePHP's core operations. Callbacks available include:

- afterFilter(), executed after all controller logic, including the rendering of the view
- beforeFilter(), executed before any controller action logic
- beforeRender(), executed after controller logic, but before the view is rendered

Model Extensions ("Behaviors")

Similarly, Behaviors work as ways to add common functionality between models. For example, if you store user data in a tree structure, you can specify your User model as behaving like a tree, and gain free functionality for removing, adding, and shifting nodes in your underlying tree structure.

Models are also supported by another class called a DataSource. DataSources are an abstraction that enable models to manipulate different types of data consistently. While the main source of data in a CakePHP application is often a database, you might write additional DataSources that allow your models to represent RSS feeds, CSV files, LDAP entries, or iCal events. DataSources allow you to associate records from different sources: rather than being limited to SQL joins, DataSources allow you to tell your LDAP model that it is associated with many iCal events.

Like controllers, models have callbacks:

- beforeFind()
- afterFind()
- beforeValidate()
- afterValidate()
- beforeSave()
- afterSave()
- beforeDelete()
- afterDelete()

The names of these methods should be descriptive enough to let you know what they do. You can find the details in the models chapter.

View Extensions ("Helpers")

A Helper is a class that aids in view logic. Much like a component used among controllers, helpers allow presentational logic to be accessed and shared between views. One of the core helpers, JsHelper, makes AJAX requests within views much easier and comes with support for jQuery (default), Prototype and Mootools.

Most applications have pieces of view code that are used repeatedly. CakePHP facilitates view code reuse with layouts and elements. By default, every view rendered by a controller is placed inside a layout. Elements are used when small snippets of content need to be reused in multiple views.