Community Experience Distilled

Website Development with PyroCMS

Quickly and efficiently develop and deploy impressive websites with PyroCMS

Zachary Vineyard

[PACKT] open source*
PUBLISHING
community experience distilled

Website Development with PyroCMS

Quickly and efficiently develop and deploy impressive websites with PyroCMS

Zachary Vineyard

[PACKT] open source*
PUBLISHING
community experience distilled

BIRMINGHAM - MUMBAI

Website Development with PyroCMS

First published: October 2013

Production Reference: 1081013

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK..

ISBN 978-1-78328-223-4

`www.packtpub.com`

Cover Image by Aniket Sawant (`aniket_sawant_photography@hotmail.com`)

Credits

Author
Zachary Vineyard

Reviewers
Stephen Coogan
Kefeng Deng

Acquisition Editor
Amarabha Banerjee

Commissioning Editor
Mohammed Fahad

Technical Editor
Faisal Siddiqui

Project Coordinator
Esha Thakker

Proofreader
Samantha Lyon

Indexer
Tejal Soni

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Zachary Vineyard is a web developer that specializes in frontend web design, content management, and application development. He's been building websites with PHP for the last 10 years. He lives in Meridian, Idaho, with his wife and twin girls.

This book would not exist without the support of my wife, Dana. I am blessed to have such a loving spouse and companion.

About the Reviewers

Stephen Coogan is a frontend Web Developer with many years of experience living in Dublin, Ireland. When he's not neck-deep in code, he can usually be found engrossed in this month's latest videogame released. He is also the developer behind Promethean Gaming (`http://www.prometheangaming.ie/`) where he occasionally writes articles, too. He also loves talking to people about the latest and greatest in web technologies, and is more than happy enough for people to reach out to him at: `hello@coog.ie`.

Kefeng Deng is an IT professional with over 9 years of work experiences in IT industry in various stages, including both Java-based and PHP-based. He is currently working for the University of Auckland as a Software Engineer, and mostly focuses on web application development using Groovy and Spring technologies. More details about him are available at `http://www.linkedin.com/in/dengkefeng`.

Table of Contents

Preface

PyroCMS is a content management system that has been growing in popularity because of its intuitive backend design and lightweight, modular architecture. Described as "a simple, flexible, community driven content management system," PyroCMS is easy to learn, understand, and own. In this book, we'll look at the primary features of PyroCMS and start building a website together.

What this book covers

Chapter 1, *Installing PyroCMS*, covers all the steps for installing PyroCMS.

Chapter 2, *The Control Panel*, covers how to use PyroCMS's backend administration area called the Control Panel.

Chapter 3, *Creating Pages and Page Types*, covers how to create pages and page types, the foundations of adding and manipulating text content, in PyroCMS.

Chapter 4, *Plugin and Module Add-ons*, covers how to build plugin and module add-ons for PyroCMS using MVC.

Chapter 5, *Creating a PyroCMS Theme*, covers how to create a theme for PyroCMS, including how to do multiple layouts.

Chapter 6, *Using PyroCMS Streams*, teaches you how to create and use streams of data in PyroCMS using the PyroStreams module.

Chapter 7, *Building a Website with PyroCMS*, shows you how to put everything you've learned into building a basic website from scratch using PyroCMS.

What you need for this book

Running PyroCMS has a few software dependencies. These dependencies usually come pre-loaded on most web hosts.

- A web server (Apache 2.x, Nginx, and so on)
- PHP 5.2 or above
- MySQL 5.x
- GD 2
- cURL 7.10.5 or above

Who this book is for

This book is for PHP developers who are looking for a great content management system. PyroCMS is built on CodeIgniter, so system adopters will need to have some familiarity with OOP and the MVC programming pattern, especially if they want to extend PyroCMS by building add-ons. PyroCMS is also for web developers looking to speed up their development times. PyroCMS comes pre-charged with great tools, such as the Streams module, to save developers time.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');

class Plugin_Hello extends Plugin
{
  // Start of a plugin
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
git clone https://github.com/pyrocms/pyrocms.git
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1 Installing PyroCMS

In this chapter, we'll walk through the entire PyroCMS installation process. While it might appear from the size of this chapter that the installation process for PyroCMS is complicated, it is, in fact, quite simple. After walking through the system pre-requirements and a few topical recommendations for PyroCMS developers, we'll follow an easy-to-navigate four-step process that gets you on your way to enjoying PyroCMS. On the way, we'll also look into a few small problem areas that might arise while working through the installation process.

By the end of this chapter, we will have also taken a look at some alternate troubleshooting problems you might face, so that, while it is unlikely, you'll be prepared for minor problems that may arise when you are trying to get PyroCMS up and running. Not to worry though, the installation process for PyroCMS is very stable.

Getting started

PyroCMS, like many other content management systems including WordPress, Typo3, or Drupal, comes with a pre-developed installation process. For PyroCMS, this installation process is easy to use and comes with a number of helpful hints just in case you hit a snag while installing the system. If, for example, your system files don't have the correct permissions profile (writeable versus write-protected), the PyroCMS installer will help you, along with all the other installation details, such as checking for required software and taking care of file permissions.

Before you can install PyroCMS (the version used for examples in this book is 2.2) on a server, there are a number of server requirements that need to be met. If you aren't sure if these requirements have been met, the PyroCMS installer will check to make sure they are available before installation is complete. Following are the software requirements for a server before PyroCMS can be installed:

- HTTP Web Server
- MySQL 5.x or higher

- PHP 5.2.x or higher
- GD2
- cURL

Among these requirements, web developers interested in PyroCMS will be glad to know that it is built on CodeIgniter, a popular MVC patterned PHP framework. I recommend that the developers looking to use PyroCMS should also have working knowledge of CodeIgniter and the MVC programming pattern.

Learn more about CodeIgniter and see their excellent system documentation online at `http://ellislab.com/codeigniter`.

CodeIgniter

If you haven't explored the **Model-View-Controller** (**MVC**) programming pattern, you'll want to brush up before you start developing for PyroCMS. The primary reason that CodeIgniter is a good framework for a CMS is that it is a well-documented framework that, when leveraged in the way PyroCMS has done, gives developers power over how long a project will take to build and the quality with which it is built. Add-on modules for PyroCMS, for example, follow the MVC method, a programming pattern that saves developers time and keeps their code dry and portable.

Dry and portable programming are two different concepts. Dry is an acronym for "don't repeat yourself" code. Portable code is like "plug-and-play" code—write it once so that it can be shared with other projects and used quickly.

HTTP web server

Out of the PyroCMS software requirements, it is obvious, you can guess, that a good HTTP web server platform will be needed. Luckily, PyroCMS can run on a variety of web server platforms, including the following:

- Abyss Web Server
- Apache 2.x
- Nginx
- Uniform Server
- Zend Community Server

If you are new to web hosting and haven't worked with web hosting software before, or this is your first time installing PyroCMS, I suggest that you use Apache as a HTTP web server. It will be the system for which you will find the most documentation and support online. If you'd prefer to avoid Apache, there is also good support for running PyroCMS on Nginx, another fairly-well documented web server platform.

MySQL

Version 5 is the latest major release of MySQL, and it has been in use for quite some time. It is the primary database choice for PyroCMS and is thoroughly supported. You don't need expert level experience with MySQL to run PyroCMS, but you'll need to be familiar with writing SQL queries and building relational databases if you plan to create add-ons for the system. You can learn more about MySQL at `http://www.mysql.com`.

PHP

Version 5.2 of PHP is no longer the officially supported release of PHP, which is, at the time of this book, Version 5.4. Version 5.2, which has been criticized as being a low server requirement for any CMS, is allowed with PyroCMS because it is the minimum version requirement for CodeIgniter, the framework upon which PyroCMS is built. While future versions of PyroCMS may upgrade this minimum requirement to PHP 5.3 or higher, you can safely use PyroCMS with PHP 5.2.

Also, many server operating systems, like SUSE and Ubuntu, install PHP 5.2 by default. You can, of course, upgrade PHP to the latest version without causing harm to your instance of PyroCMS. To help future-proof your installation of PyroCMS, it may be wise to install PHP 5.3 or above, to maximize your readiness for when PyroCMS more strictly adopts features found in PHP 5.3 and 5.4, such as namespaceing.

GD2

GD2, a library used in the manipulation and creation of images, is used by PyroCMS to dynamically generate images (where needed) and to crop and resize images used in many PyroCMS modules and add-ons. The image-based support offered by this library is invaluable.

cURL

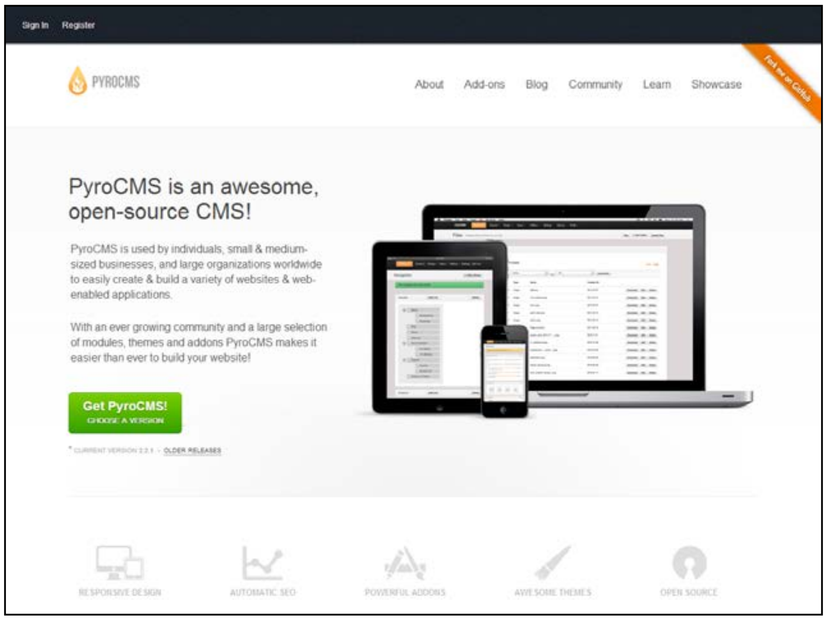
As described on the cURL project website, cURL is "a command line tool for transferring data with URL syntax" using a large number of methods, including HTTP(S) GET, POST, PUT, and so on. You can learn more about the project and how to use cURL on their website `http://curl.haxx.se`. If you've never used cURL with PHP, I recommend taking time to learn how to use it, especially if you are thinking about building a web-based API using PyroCMS.

> Most popular web hosting companies meet the basic server requirements for PyroCMS.

Downloading PyroCMS

Getting your hands on a copy of PyroCMS is very simple. You can download the system files from one of two locations, the PryoCMS project website and GitHub.

To download PyroCMS from the project website, visit `http://www.pyrocms.com` and click on the green button labeled **Get PyroCMS!** This will take you to a download page that gives you the choice between downloading the Community version of PyroCMS and buying the Professional version. If you are new to PyroCMS, you can start with the Community version, currently at Version 2.2.3. The following screenshot shows the download screen:

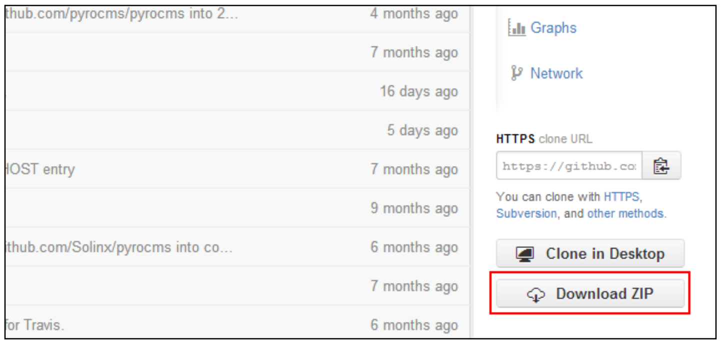


PyroCMS is an awesome, open-source CMS!

PyroCMS is used by individuals, small & medium-sized businesses, and large organizations worldwide to easily create & build a variety of websites & web-enabled applications.

With an ever growing community and a large selection of modules, themes and addons PyroCMS makes it easier than ever to build your website!

To download PyroCMS from GitHub, visit `https://github.com/pyrocms/pyrocms` and click on the button labeled **Download ZIP** to get the latest Community version of PyroCMS, as shown in the following screenshot:



If you know how to use Git, you can also clone a fresh version of PyroCMS using the following command. A word of warning, cloning PyroCMS from GitHub will usually give you the latest, stable release of the system, but it could include changes not described in this book. Make sure you checkout a stable release from PyroCMS's repository.

```
git clone https://github.com/pyrocms/pyrocms.git
```

As a side-note, if you've never used Git, I recommend taking some time to get started using it. PyroCMS is an open source project hosted in a Git repository on Github, which means that the system is open to being improved by any developer looking to contribute to the well-being of the project. It is also very common for PyroCMS developers to host their own add-on projects on Github and other online Git repository services.

The difference between the Community and Professional versions of PyroCMS

While both the Professional and Community versions of PyroCMS are nearly identical, there are some feature benefits from the Professional version that are worth your one-time $90 investment (at the time of this book). The following is a short list of features that, by default, come with the Community version of PyroCMS:

- Blog module
- Pages module (with drag-and-drop ordering and so on)

- Theme, Widget, and Module add-ons
- Responsive control panel (accessible from many devices)
- Analytics, Akismet, Storage Provider (Amazon S3 and Rackspace), and SMTP integration
- Powered by CodeIgniter and the modular MVC pattern

The Professional version of PyroCMS comes with all the preceding listed features and few other very powerful additions, including the following:

- PyroStreams (custom data management)
- Multi-site hosting
- White-labeling (permission for custom branding)

At first glance, it may seem that the Professional version of PyroCMS just offers you a few more advanced features, but in reality, these features can transform the way you think about web development. PyroStreams and Multi-site hosting, in particular, scale your ability to run more than a handful of websites at one time. Because of this, I think it is important to talk about why the PyroCMS modules play such a pivotal role in which version of PyroCMS you should run for your next website project.

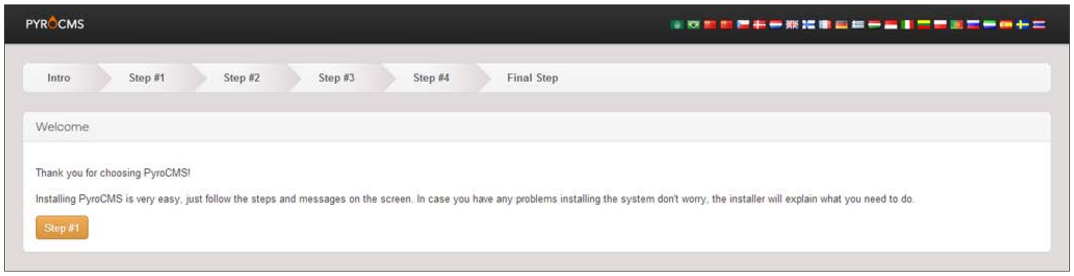
PyroStreams

If you could describe the PyroStreams module for PyroCMS in one sentence, it would probably sound like this, PyroStreams is a smart interface for creating custom data streams. This makes it incredibly simple to manage large sets of data associated with your website. I'm not going to explore this feature too much more in this chapter, but I will emphasize that this feature alone is worth the small fee you pay for the professional version of PyroCMS.

The installer

The installer is a step-by-step tool web developers can use to install PyroCMS. It is provided with every download of PyroCMS and is located in the root directory of the project.

The initial screen of the installer looks like the following screenshot:



Welcome

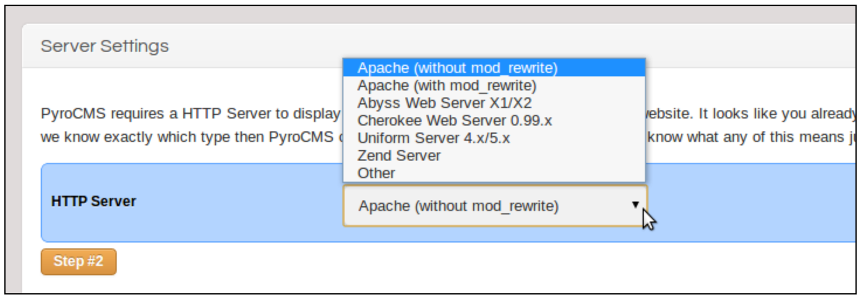
Thank you for choosing PyroCMS!

Installing PyroCMS is very easy, just follow the steps and messages on the screen. In case you have any problems installing the system don't worry, the installer will explain what you need to do.

Take note of the orange button leading you to the first step of the PyroCMS installer. Enlisted are the steps found in the PyroCMS installer:

1. The first step of the PyroCMS installer allows you to define the database that will support the system. The first step has a form with seven input fields broken into two small sections. While fairly self-explanatory, here you can define the name of your database and the credentials with which you connect to that database, including MySQL hostname, port, password, and more. These settings should be fairly straight-forward if you've worked with other content management systems before.

   One installation feature that's unique to PyroCMS is that it lets you choose the type of HTTP web server platform on which to run, to help make your instance of PyroCMS run as smoothly as possible. You can select your HTTP server from a drop-down menu located at the bottom of the screen of the first step of the PyroCMS installer, as shown in the following screenshot:

   Server Settings

   PyroCMS requires a HTTP Server to display ebsite. It looks like you already
   we know exactly which type then PyroCMS know what any of this means j

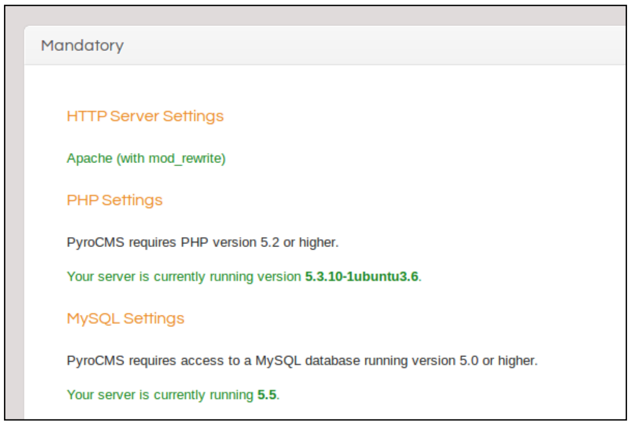
   HTTP Server: Apache (without mod_rewrite)

   Apache (without mod_rewrite)
Apache (with mod_rewrite)
Abyss Web Server X1/X2
Cherokee Web Server 0.99.x
Uniform Server 4.x/5.x
Zend Server
Other
Step #2

   Once you've filled out the basic form in step one, including choosing a HTTP web server, it's time to move to step two of the installer. Be careful what you choose from this menu. If you choose the wrong system, troubleshooting problems with URLs that don't resolve (and other similar problems) may not be very easy.

2. The second step on the PyroCMS installer checks to make sure that your server has all the system pre-requisites (discussed earlier in this chapter) installed. By default, it checks to make sure you have at least PHP Version 5.2 installed. While some developers might see this as a version of PHP that should no longer be supported, it is still used as a default version of PHP by a variety of server operating systems. CodeIgniter 3, the framework that PyroCMS currently uses, still complies with PHP 5.2.

   Step two of the install process checks to make sure all the other pre-requisites are installed, including MySQL, GD, cURL and more. If this part of the installation process determines that all the necessary software is installed to run PyroCMS, it will automatically skip to step three, as shown in the following screenshot:



Mandatory

HTTP Server Settings

Apache (with mod_rewrite)

PHP Settings

PyroCMS requires PHP version 5.2 or higher.

Your server is currently running version **5.3.10-1ubuntu3.6**.

MySQL Settings

PyroCMS requires access to a MySQL database running version 5.0 or higher.

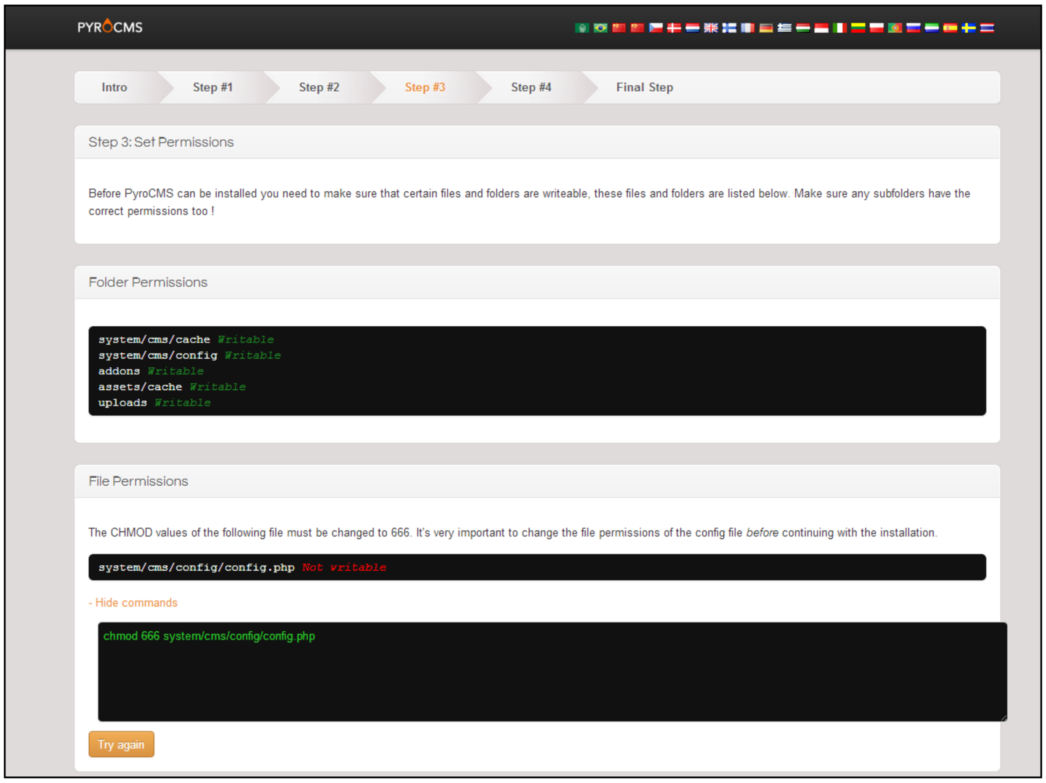
Your server is currently running **5.5**.

3. The third step of the PyroCMS installer is dedicated to make sure that files and folders associated with your instance of PyroCMS have the appropriate file-level permissions. Essentially, the installer checks to make sure that the directories for uploaded files and cache are writeable, and that files like your main PyroCMS configuration file aren't writeable. Your main PyroCMS configuration file is located at `system/cms/config/config.php`.

   If the installer finds that your file permissions are not correctly set for the system, it will offer you commands to run (Linux only) that will help correct the situation. Following is an example of what those commands look like:

```
chmod 777 system/cms/cache
chmod 666 system/cms/config/config.php
```

The following screenshot shows the installer asking you for commands to run:



Step 3: Set Permissions

Before PyroCMS can be installed you need to make sure that certain files and folders are writeable, these files and folders are listed below. Make sure any subfolders have the correct permissions too !

Folder Permissions

```
system/cms/cache Writable
system/cms/config Writable
addons Writable
assets/cache Writable
uploads Writable
```

File Permissions

The CHMOD values of the following file must be changed to 666. It's very important to change the file permissions of the config file *before* continuing with the installation.

```
system/cms/config/config.php Not writable
```

- Hide commands

```
chmod 666 system/cms/config/config.php
```

An exhaustive list of which file/folder permissions will need to be changed will be provided to you in this step of the installation. The most critical folders and files the system checks are enlisted as follows:

- assets/cache
- system/cms/cache
- system/cms/config
- system/cms/config/config.php
- system/cms/logs
- uploads

Be sure that the most sensitive files (usually `system/cms/config/config.php` and `system/cms/config/database.php`) are protected from being read by your website visitors.

4. The final step of the PyroCMS installer is comprised of two parts, one of which you complete upon submission of step four's form, while the other runs in the background.

   Create Your Admin Account: Step four is partially dedicated to creating the administrator account connected to your instance of PyroCMS. The user account you create in this step will be the default super-admin account, so it is important to use a strong password.

   Run the Installation: The second part of the fourth step is running the installation. Up until this point in the installation process, you've been completing the steps necessary for running the final installation step. Once you click on the **Install** button at the end of step four, the installer will check to make sure that all the information you provided will work for the installation. It will also, if you chose it to do so, try and create the database, before it creates and populates any database tables, needed for your instance of PyroCMS. If the installer cannot create the database, this final validation check will fail. But don't worry, that just means you'll need to create the database manually and then click on the **Install** button again.

Once you complete step four of the installation process, you'll be prompted to either visit the front-end of your new website or click through to the control panel, the administrative area of PyroCMS. We'll cover the control panel in the following chapter.

Troubleshooting installation

If you've followed all the instructions in this chapter but still run into errors trying to get access to the control panel or to the homepage of your site, do not worry. If you are using Apache with `mod_rewrite` (a popular Apache module) enabled, you may need to change a few configuration variables or adjust the settings in your `.htaccess` file to accommodate your host server's requirements. Please make sure you have enabled the `mod_rewrite.c` module in Apache's `httpd.conf` configuration file. You can learn more in the section of the installation guide in the PyroCMS documentation at `http://docs.pyrocms.com/2.2/manual/reference/troubleshooting`. Often, you can find other developers describing solutions to problems on the PyroCMS forums at `https://forum.pyrocms.com`.

Summary

This chapter covered the entire installation process of PyroCMS, from step one's lead-in to creating and connecting to the database supporting your installation, to the final step where the super-admin user account is created. We also covered some basic installation troubleshooting and configuration tips. In between these points, we also talked about MVC, CodeIgniter, and the required software of which you'll need a basic understanding to run PyroCMS. In the following chapter, we are going to explore PyroCMS's control panel.

2
The Control Panel

The PyroCMS control panel is the tool where you (or your system administrator) will handle all the administration of your PyroCMS website. In this chapter, we're going to review the primary features of PyroCMS's control panel, including the area where you can modify all the settings of your system installation. As you'll find out, many of the details of setting up your PyroCMS installation are easy to use. There are a few other settings and control panel ideas, however, that really come into light after a little exploration.

Control panel access

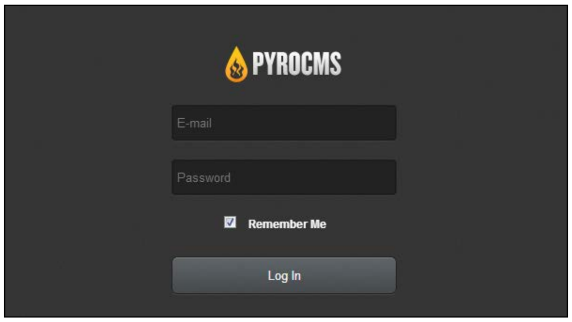
After you have finished installing PyroCMS, you'll want to jump right into website administration. To access your PyroCMS installation, visit the following address (using, of course, your own domain instead of `example.com`):

```
http://www.example.com/admin
```

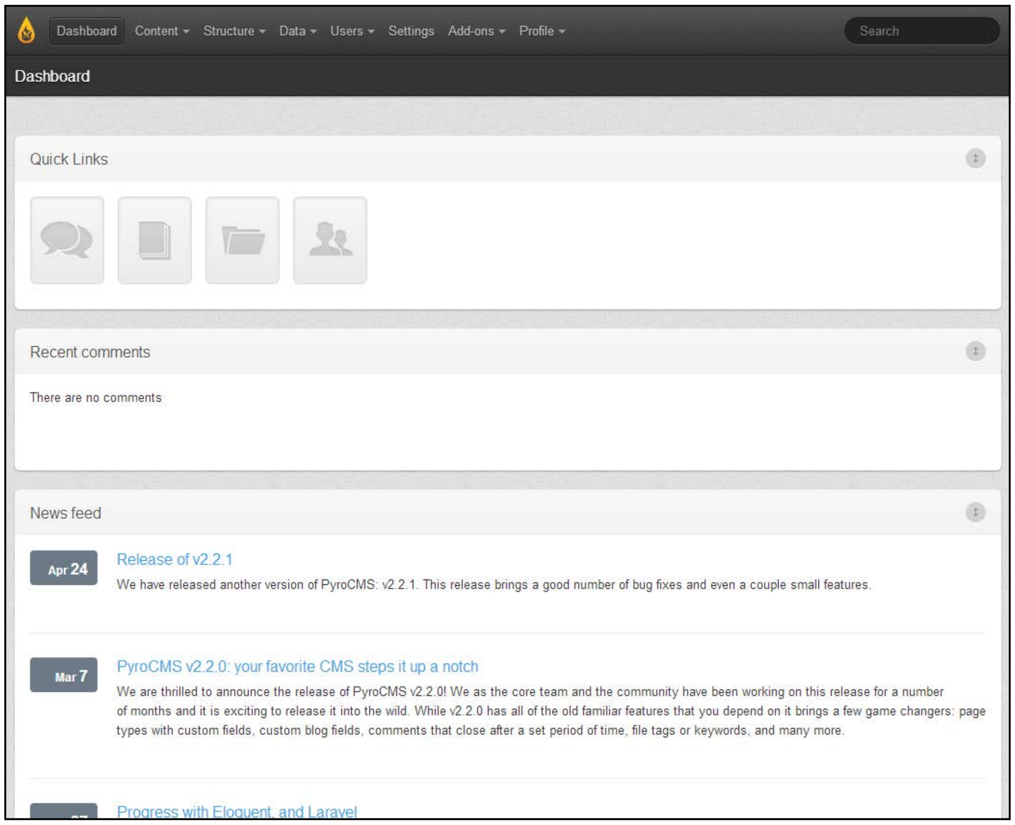
If you aren't using Apache with the `mod_rewrite` module enabled, the URL to the control panel of your PyroCMS installation will be the following URL:

```
http://www.example.com/index.php/admin
```

At this URL, you'll be greeted with the standard PyroCMS login form, as seen in the following screenshot:



Log in to this form using the administrator username (usually an email address) and password you created during step four of the installation process. After you've logged in, you'll be taken to the initial page of the control panel (called the dashboard), as shown in the following screenshot:



There are no comments

News feed

Apr 24 Release of v2.2.1

We have released another version of PyroCMS: v2.2.1. This release brings a good number of bug fixes and even a couple small features.

Mar 7 PyroCMS v2.2.0: your favorite CMS steps it up a notch

We are thrilled to announce the release of PyroCMS v2.2.0! We as the core team and the community have been working on this release for a number of months and it is exciting to release it into the wild. While v2.2.0 has all of the old familiar features that you depend on it brings a few game changers: page types with custom fields, custom blog fields, comments that close after a set period of time, file tags or keywords, and many more.

Progress with Eloquent, and Laravel

If you haven't done so already, it is very important that you remove the PyroCMS `install` directory from your web server. Leaving this directory on your server compromises the security of your site, leaving it available to be taken over by anyone.

The dashboard

The dashboard is the first screen you see after logging into your PyroCMS control panel. It is designed to give you a quick overview of recent changes that may have happened on your website, such as whether there are new blog comments. The dashboard also provides you with a news stream of data from the PyroCMS blog. This stream will help keep you updated about changes in PyroCMS, including security patches, updates, and more.

Layout and navigation

The PyroCMS control panel is broken up into three sections, the main menu at the top of the screen, a sub-menu containing supporting links and primary actions buttons, and the primary content and system administration area below the sub-menu. In these three areas you'll find a familiar user experience pattern that makes the PyroCMS control panel simple and intuitive.

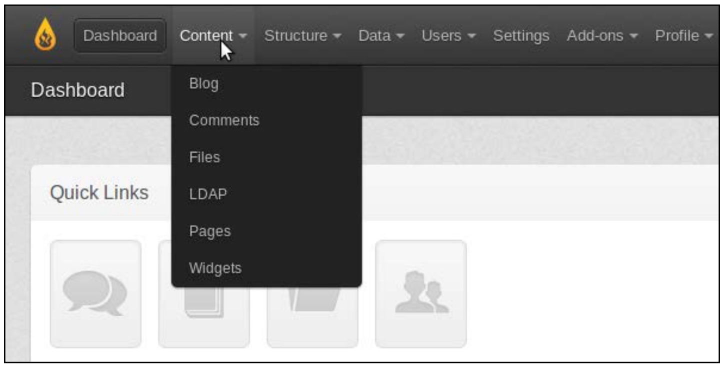
The primary PyroCMS navigation is broken into seven major sections, as shown in the following list. The control panel also includes a button that links you back to the dashboard, and a logo which when clicked on will open the frontend of your website in a new browser tab. The PyroCMS navigation also includes a search function you can use to help locate specific website data. This search feature auto-completes, and can quickly point you to a blog post or a page on your website.

- Content
- Structure
- Data
- Users
- Settings
- Add-ons
- Profile

The most commonly used control panel modules, for most PyroCMS users, are the ones connected to content, which is why the **Content** navigation item is listed first in the control panel's main menu.

Content

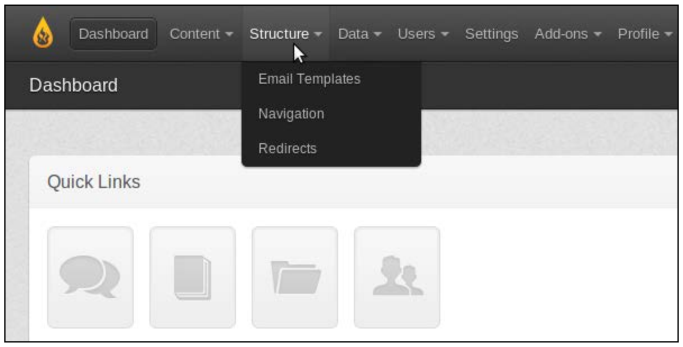
The following screenshot shows the **Content** option on the **Dashboard**:



The **Content** navigation section of the control panel is for modules that deal with your website's content, including blogs, pages, widgets, uploaded files and other media. This section will be the primary area of focus for you and the other people who use the control panel on your website.

Structure

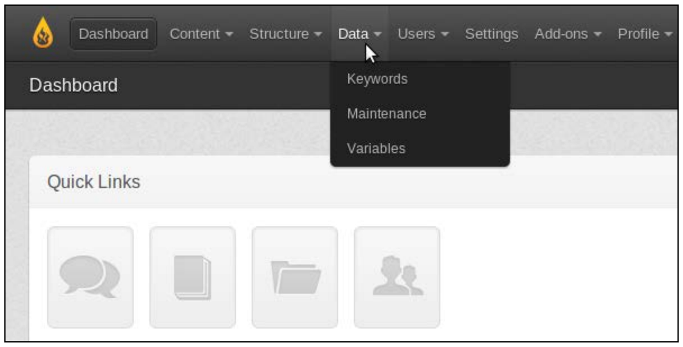
The following screenshot shows the **Structure** option on the **Dashboard**:



Especially important to your efforts in content strategy and information architecture, the **Structure** section of the control panel is where you'll find links to modules that control navigation elements, redirects, and email templates.

Data

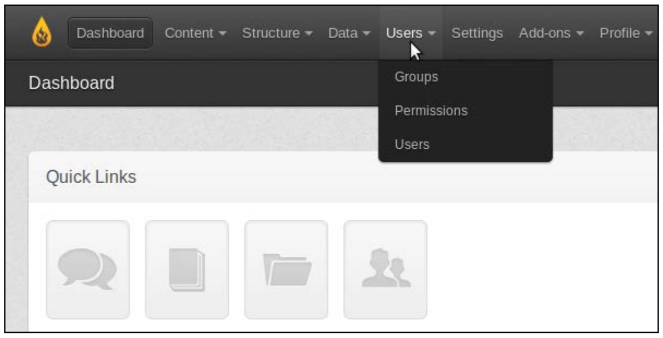
The following screenshot shows the **Data** option on the **Dashboard**:



A lesser-used control panel navigation area, this section is reserved for modules that help control keywords (such as blog post tags) and global variables for your website. While it is true that you won't visit this control panel section often, don't underestimate how much power the modules to which it links have. The `Variables` module, for example, can help you control specific pieces of information (such as prices or phone numbers), however small, from a central location.

Users

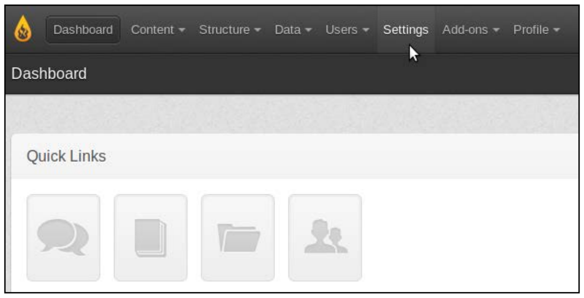
The following screenshot shows the **Users** option on the **Dashboard**:



The **Users** section is obviously where you'll look for modules that help with user and permissions administration. Here you'll find ways to build user groups, change group permissions, and add new users to the system.

Settings

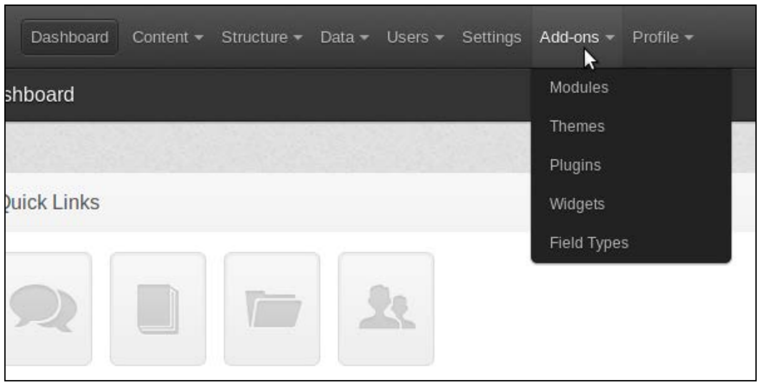
The following screenshot shows the **Settings** option on the **Dashboard**:



The **Settings** navigation area is, at first, one of your most visited areas in the control panel. But once you have the primary settings for your website all plugged in, you'll rarely find the need to use this navigation area. As you can imagine, though, this area holds settings for the entire website (like website name, third-party integration settings, and language preferences). Also, in this area, individual module preferences are editable.

Add-ons

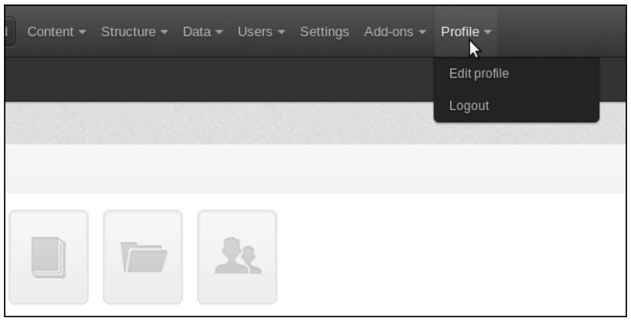
The following screenshot shows the **Add-ons** option on the **Dashboard**:



This area of the control panel contains modules that list the currently installed modules, field types, plugins, themes, and widgets. This is where you control which theme is being used on your website, as well as where you can install and upgrade third-party modules. If a module you install doesn't add a link in one of the primary navigation areas in the control panel, you can find a link to that module through this menu.

Profile

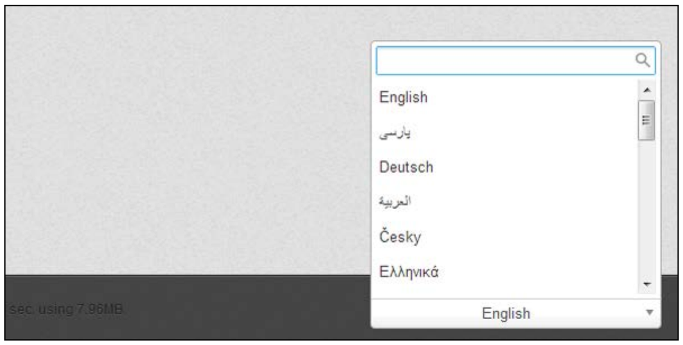
The following screenshot shows the **Profile** option on the **Dashboard**:



The **Profile** navigation area of the control panel is reserved for modules that help you control your own website's profile information, including your password. The option to log out from the control panel is also located under this navigation item.

Choosing a default language

The PyroCMS control panel and some add-ons have been translated into many different languages. You can switch between languages in PyroCMS by using the language menu in the footer of the control panel, as shown in the following screenshot:



Summary

In this chapter we explored PyroCMS's control panel, the system area where users like you visit to manage your website. We also quickly explored all the main navigation areas of the control panel to understand better how it is organized. Having solid knowledge of how the control panel is organized and designed helps inform how you should build PyroCMS add-ons and where to quickly find and edit information on your website. In the following chapter we'll cover how to create pages and page types.

3
Creating Pages and Page Types

Pages and page types can be described as the architectural foundations of most of the content in your website. This means, basically, that your website's content, in one way or another, will revolve around the creation of pages. Some content, such as blog posts, for example, doesn't need to interact with the `Pages` module in PyroCMS, but most other content will. In this chapter we'll explore how to create various types of pages in PyroCMS. We'll also walk through how to create page types in the system. By the end of this chapter you will have a good grasp on various ways to work with pages in PyroCMS.

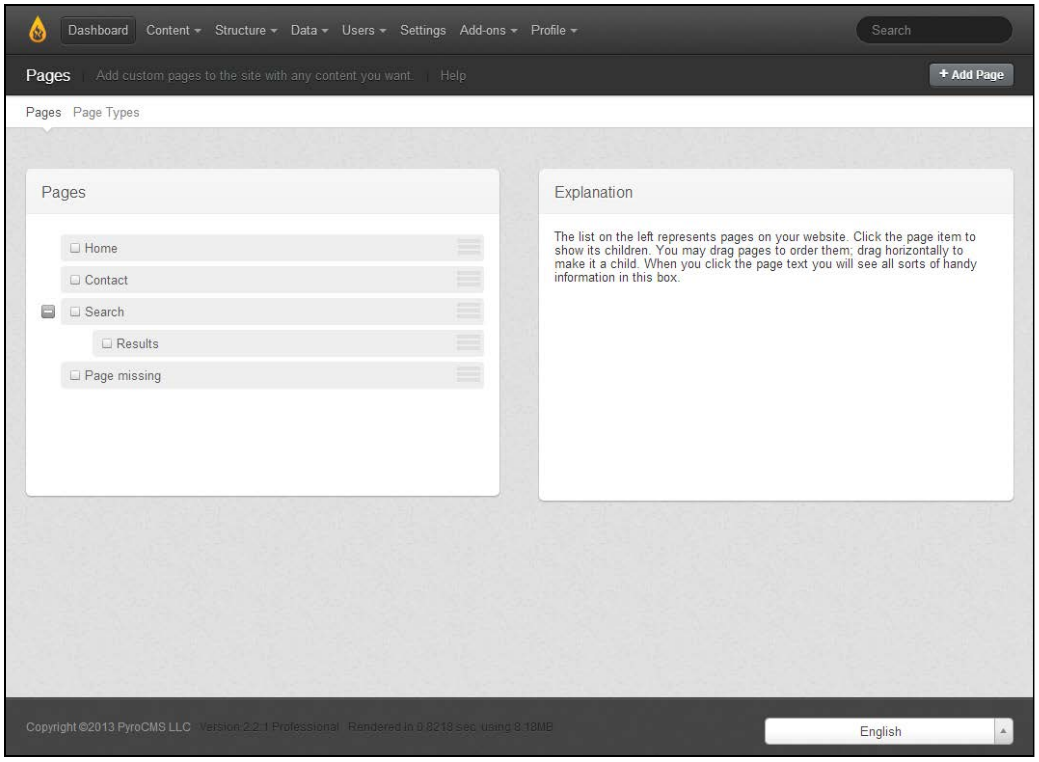
Pages

If you've worked with other content management systems in the past, you're likely to be familiar with the idea of pages. Pages are singular collections of data usually represented by a single URL. Pages, in general, have a purpose and these purposes vary widely. Pages can, for example, list contact information, product details, a portfolio of work, or just some text. While every page will likely have unique content, they can also have common elements, including metadata, permissions or privacy settings, and more.

Pages in PyroCMS, like in other systems, are usually broken into one of two categories, basic content pages and dynamic pages. Both these categories of pages are built in different ways in PyroCMS, but each touch PyroCMS's `Pages` module.

The Pages module

In PyroCMS, basic pages are created through the `Pages` module. This module is found under the **Content** area of the main menu of your website's control panel. When you open the `Pages` module, you'll be greeted with an easy-to-use interface that lets you organize your pages into a hierarchy. The following interface (at times nicknamed the page tree) literally lets you drag-and-drop pages underneath other pages to form a page hierarchy:



The list on the left represents pages on your website. Click the page item to show its children. You may drag pages to order them; drag horizontally to make it a child. When you click the page text you will see all sorts of handy information in this box.

Copyright ©2013 PyroCMS LLC Version 2.2.1 Professional Rendered in 0.8218 sec. using 8.18MB

When you select a page in the page tree, you're given a few actionable choices for that page, which are as follows:

- Add child
- Duplicate
- Edit
- Delete

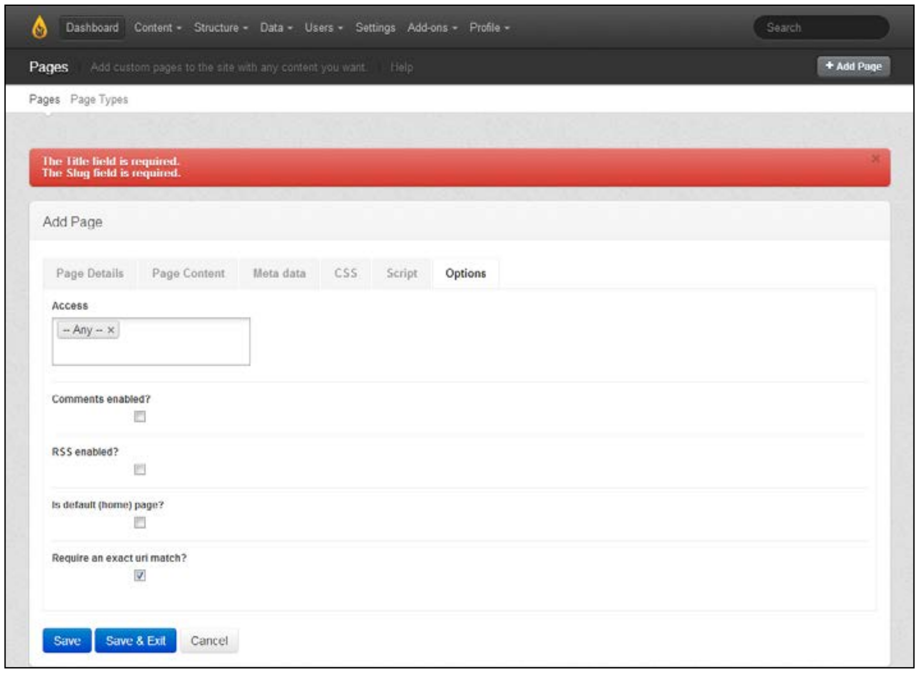
It is obvious to deduce what these actions will do to the page. But before we explore page editing, it will be good for you to understand how to create pages in PyroCMS.

How to create a basic page

To create a basic page in PyroCMS, choose the `Pages` module out of the main control panel menu, and then click on the **Add Page** button that appears near the top-right corner of the screen. After clicking on this button, you're presented with a form that helps you build a page. By default, this form is split over the following six tabs:

- Page Details
- Page Content
- Meta Data
- CSS
- Script
- Options

Throughout these six tabs you'll find input fields, each of which are for a different part of a page's settings or content. By default only two of these input fields are required to create a new page, **Title** and **Slug**. The **Title** input, of course, is where you'll put the title of the page. The **Slug** input is where you'll create the unique URL slug for the page which, because of the way CodeIgniter handles URLs, becomes one way that you can uniquely identify a page:



The Title field is required.
The Slug field is required.

Add Page

Access: -- Any --

Comments enabled? ☐

RSS enabled? ☐

Is default (home) page? ☐

Require an exact uri match? ☑

Page Details tab

The inputs on the page details tab include the following:

- Title
- Slug
- Status
- Add to Navigation

We've already learned about the first two input fields (**Title** and **Slug**) in the **Page Details** tab of PyroCMS's page creation form. The other two input fields, **Status** and **Add to Navigation**, handle two important settings for a page, whether the page is publically viewable (that is live or in draft) and into which menu a link to this page will be placed, respectfully.

Page Content tab

The **Page Content** tab is where you'll find the default WYSIWYG for adding the primary content to your page. This input can take a variety of text inputs, including HTML and plugin Lex parser tags. We'll talk more about the Lex parser later in the book.

Meta Data tab

The **Meta Data** tab of the new page creation form is where you can basically SEO meta tag information to a page, including a meta title, meta keywords, and a meta description.

Please note that the meta title input overrides the page title assigned when you created the page.

On this tab, you can choose whether or not to allow all of the links within the page's content, to include the no-follow attribute and whether or not robots (such as Google) will be able to index the page.

Not all robots are created equal. Some content web crawlers will ignore this setting. As a security feature, it is recommended that you block those robots from scanning content on your server.

CSS and Script tabs

Both the **CSS** and **Script** tabs within the page creation form allow you to add custom CSS or JavaScript to any page. This can be a handy fix for small CSS or JavaScript needs, but it is discouraged to over-use these input fields. It is usually better to include custom CSS and JavaScript in your site's template files.

Options tab

The **Options** tab in the page creation form contains a few auxiliary form inputs that control some page settings, including the following:

- Access
- Comments enabled?
- RSS enabled?
- Is default (home) page?
- Require and exact uri-match?

Each of these input fields describe basic choices you can make about this page. The **Access** input area, for example, allows you to choose which groups (permissions) area allowed to see the page.

Save the page

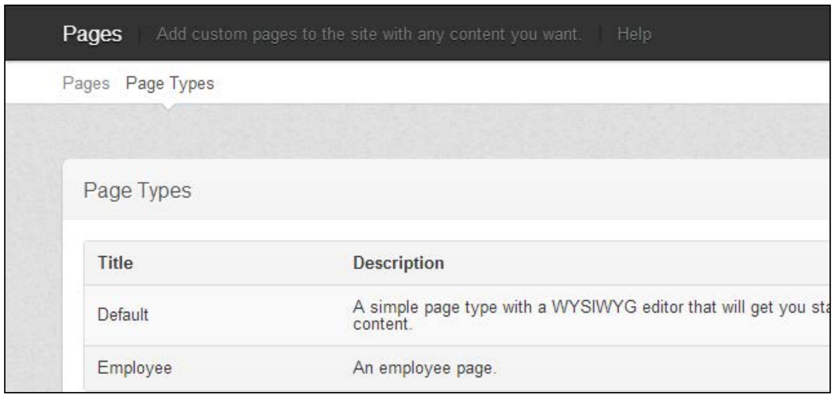
Once you've completed the required (and hopefully many of the optional) input fields on the page creation form, you can save the page by clicking on one of the blue save buttons at the bottom of the window. The page should now appear in the page tree in the `Pages` module.

Page types

PyroCMS Version 2.2 introduced a new feature to its users and developers, page types. Page types are a very flexible way to store content. You could basically describe page types as administrative templates for content. Essentially, page types give you a way to organize a variety of types of data into different types of PyroCMS pages. The best way to learn how page types can become an integral part of your website is to put them into practice.

Create a new page type

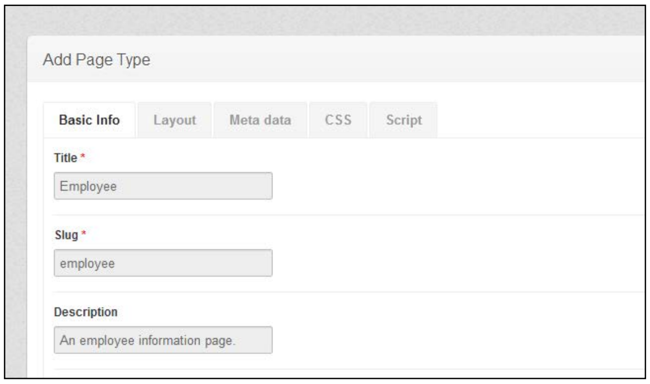
To create a new page type, select **Pages** from the **Content** section in main control panel menu. Then click on **Page Types** in the sub-menu section:



Page Types

Title	Description
Default	A simple page type with a WYSIWYG editor that will get you st content.
Employee	An employee page.

From here, clicking on the **Add Page Type** button near the top-right corner of your browser window will lead you to a form that allows you to build a new page type. As an example, we'll create a page type for listing information about a business's employee:



Title *
Employee

Slug *
employee

Description
An employee information page.

For the moment, it isn't important to focus on any fields of this form, other than the **Layout** input field on the **Layout** tab. This input is where you'll define the basic HTML markup for an employee information page. As you can imagine, markup on these page types can get pretty complicated, but we are going to keep things easy. We're going to create a page type that lists an employee's name, bio, and email address.

Markup for a page like that would look like the following code in PyroCMS:

```
<h2>{{ title }}</h2>
<p>{{ bio }}</p>
<p>{{ email:safe_mailto_link }}</p>
```

Once you've added this markup to the **Layout** input field, make sure you save your work by clicking on the blue **Save** button at the bottom of the page.

This markup, if you can't tell, simply lists the name of the employee wrapped in an H2 HTML tag, the person's bio, and a safe e-mail link. What you've probably noticed, though, is that this isn't your traditional HTML markup. That's because PyroCMS takes advantage of a very powerful template tag parsing system called the Lex Tag Parser. At this point, all you need to know about the Lex parser is that it maps data to special Lex tags. To make our custom page type work, we need to build a way for you to add data to a page that maps to those Lex parser tags.

Lex parser tags are powered by a library added to PyroCMS that can parse variables and other logic into PyroCMS layouts and templates. We will cover the use of tags in both chapters four and five.

Add custom page type fields

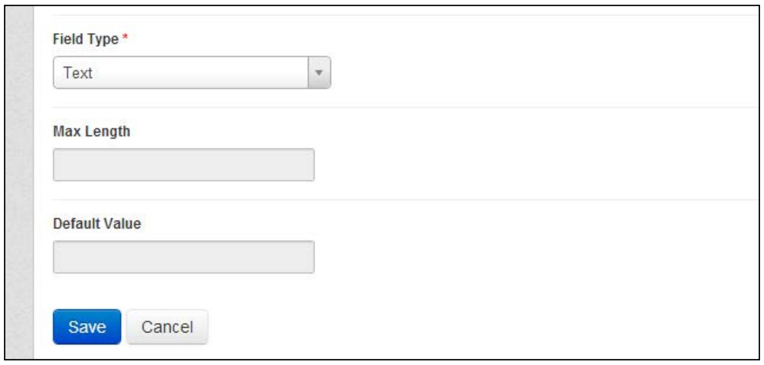
After you create your page type (by clicking on the blue **Save** button), you'll land up on the page type field page, where the system will likely tell you that you don't have any custom fields yet. To create a new field, click on the **New Field** button near the top-right of your browser window. Upon that click, you'll be taken to a form that allows you to create a custom field for your page type. You can start by creating a `name` field. In doing so, you'll be given choices on whether you can make this input field required for the user and so on. The most important choice you'll make about this input filed is its type.

By default PyroCMS gives you sixteen different field types to choose from, which are as follows:

- Choice
- Country
- DateTime
- Email
- Encrypt
- File

- Image
- Keywords
- Relationship
- Slug
- Text
- Textarea
- URL
- US State
- User
- WYSIWYG

While some of these input types have obvious implications, such as File, Image, and Textarea, some are a little more cryptic. You can learn everything you need to know about these input fields from the PyroCMS documentation at `http://docs.pyrocms.com/2.2/manual/field-types`. For our `name` field, we're going to select the **Text** type:

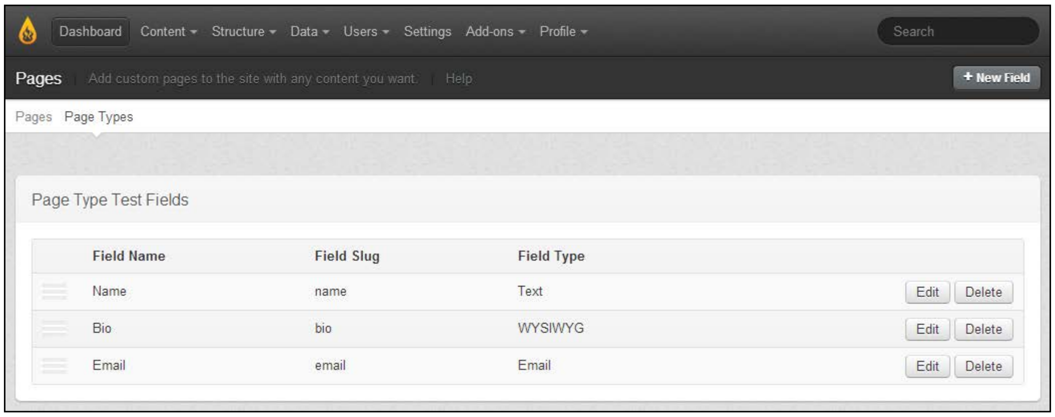


Field Type *: Text

Max Length

Default Value

Once you create this input field, you should get a success message from the system that reminds you about adding the Lex parser tag to your page layout, "...before its data will be output you must insert its tag into the Page Type's Layout textarea." Since we've already added the name `Lex tag` to the layout, our data is already aligned and should show up on our page. To add an employee's bio and e-mail address to the layout, follow the same process with which we made the name input field:

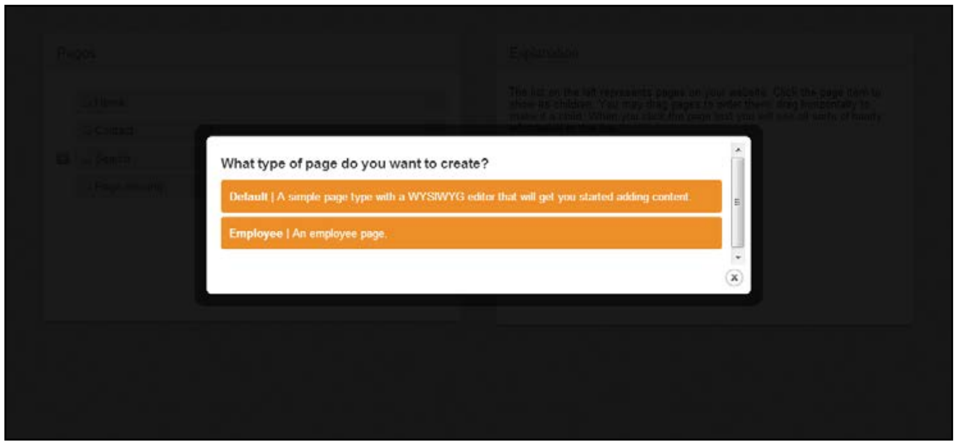


Page Type Test Fields

Field Name	Field Slug	Field Type	
Name	name	Text	Edit Delete
Bio	bio	WYSIWYG	Edit Delete
Email	email	Email	Edit Delete

Create a new page with custom data

Now that we've added the custom input fields to our new page type and added our page type into the system, we can create a new page in PyroCMS that lists, based on our example, an employee's information. To do this, return to the `Pages` module in the main menu in the control panel. Now when you click on the **Add Page** button, a modal window will pop up that gives you the option of which type of page you'd like to create, as shown in the following screenshot:



What type of page do you want to create?

Default | A simple page type with a WYSIWYG editor that will get you started adding content.

Employee | An employee page.

Choosing the **Employee** page type will go back to the page creation form, but now it has some distinct differences. Now when you click on the **Page Content** tab, you'll now see input fields for an employee's name, bio, and e-mail address.

[Using drag-and-drop, you can re-order the custom input fields of a page type by clicking on the **Fields** button next to a page type's listing on the **Page Types** page.]

Summary

In this chapter we covered how to create a new page in PyroCMS, including the various options and settings associated with a new page. These options included text information that is helpful for SEO. We also explored how to create a new page type in PyroCMS, giving you the ability to shape how content is organized and displayed on pages on your website. Custom page types added great value beyond the basic WYSIWYG data entry mode found in many other content management systems. In the following chapter we'll discover how create plugin and module add-ons for PyroCMS.

4
Plugin and Module Add-ons

Like many content management systems and other web applications, PyroCMS comes pre-loaded with a way to extend the core system. In this chapter we're going to explore two primary ways with which you can extend PyroCMS, which are as follows:

- Plugins
- Modules

Plugins

If you've ever worked with other content management systems (such as WordPress, for example) then you're likely to be familiar with the idea of plugins. In PyroCMS, plugins are usually built to accommodate specialized content. Examples of this type of content would be a Google calendar parsed into HTML or a three day weather forecast. Plugins in PyroCMS are usually built to accommodate a single task and they can be built in conjunction with modules (a type of plugin we'll also look at in this chapter). At heart, plugins are the simplest type of add-on a PyroCMS developer can create.

Tags

One of the central PyroCMS functionality concepts is Tags (also called Lex parser tags). We quickly covered how tags work in PyroCMS in the previous chapter, but we didn't cover the fact that tags are powered by plugins. The following tag is an example of a simple tag that returns the current URL:

```
{{ url:current }}
```

The code that powers this tag is a plugin, which is really a PHP file that can be called via PyroCMS tags. Plugins are simple to write and made incorporating complex functionality into PyroCMS clean and organized. The best part about PyroCMS plugins is that they can be used everywhere, including WYSIWYG inputs, template files, and layouts and page types. This type of plug-and-play action for PyroCMS plugins is what makes them so powerful and distinct from plugins you find in other systems.

Although they are identical in structure, a plugin can either be standalone file or be a `plugin.php` file within a larger module.

How to create a plugin

Building a plugin for PyroCMS is very easy. To get started building your first plugin, create a new PHP file in one of the two places that plugins may reside within an instance of PyroCMS. We're going to create a basic `Hello World` plugin inside a plugin file named `hello.php`. Create this file in one of the following two locations:

`addons/shared_addons/plugins/hello.php` (for plugins available to all websites)

OR:

`addons/[your-site-name]/plugins/hello.php` (for themes available to only one specific website)

Plugin PHP files usually contain only one PHP class. In this case the start of the `Hello World` class in our plugin will look like the following code. Remember, PyroCMS runs on CodeIgniter, so our plugins can use CodeIgniter libraries and methods.

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');

class Plugin_Hello extends Plugin
{
  // Start of a plugin
}
```

The start of this class is the default way in which you should start building a custom class in CodeIgniter. The first line of this code keeps the script from being loaded if you aren't running it as part of your CodeIgniter system. This line of code isn't necessary but it is recommended that you use it. After this, we create a new class called `Plugin_Hello` that extends the plugin class. Please note that the class name is `Plugin_` followed by the plugin name (lowercase with the first letter in uppercase). Also note that the class name is the same as the file name. Now we'll implement our first class method called `say_hi`, using the following code:

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');

class Plugin_Hello extends Plugin
{
  function say_hi()
  {
    $name = $this->attribute('name');
    if($name)
    {
      return 'Hello '.$name;
    }
    else
    {
      return "Hello Dude";
    }
  }
}
```

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

This `say_hi` method is very simple. It looks for an attribute called `name`. If that attribute exists, it returns it with the appended string `Hello`. If the attribute doesn't exist, it returns the string `Hello Dude`. Also, in the preceding code, please note that the data of each function is returned, not echoed.

While this example is very simple, the preceding code is all you need for a plugin in PyroCMS. Using this plugin in PyroCMS is centered on how to use tags in the system. For example, the following code is the tag you would use to hook our `Hello World` plugin:

```
{{ hello:say_hi name="Fred" }}
```

The preceding tag could be placed in any text or WYSIWYG input, or any template or layout. Where this tag exists we would see the output string `Hello Fred`.

> You can learn more about plugins in the PyroCMS documentation found online at `http://www.pyrocms.com/documentation`.

Modules

For PyroCMS, modules are usually the largest and most intensive type of add-on you can create. They are also, however, very flexible. With a module you can have the following features:

- MVC
- Install/Uninstall functions
- Backend interfaces
- Frontend URIs
- Libraries, helpers, configs, and routes
- A plugin file (see the preceding section on plugins)
- Widgets
- Settings

Because PyroCMS is a modular system, once you start building a module, you'll see how it fits into the overall structure of the system.

Folder structure

PyroCMS modules consist of PHP files arranged into the following supported folders:

- config/
- controllers/
- helpers/
- libraries/
- models/
- views/
- js/
- css/
- img/

If you have worked with CodeIgniter, you should feel right at home with these folders. If not, you'll notice that these folders support the MVC programming pattern, separating the concerns of your module into its most important parts, config values, controllers, models, views, and so on. As a developer, you have full power to use MVC at your disposal. Not only does this give your module the flexibility it needs to be strong, but it keeps your code extremely well organized.

[PyroCMS is built using object-oriented PHP. If you have not studied OOP or MVC, you may have trouble building module add-ons.]

How to create a module

Creating modules in PyroCMS is pretty straight-forward. Like plugins, modules are built in one of two locations in your instance of PyroCMS, shown as follows:

`addons/shared_addons/modules/[module-name]` (for modules available to all websites)

OR:

`addons/[site-name]/modules/[module-name]` (for modules available to only one specific website)

Once you have your base module folder containing the supported folder structure created, the first file that you'll want to add to your modules is named `details.php`. Every module add-on in PyroCMS has a `details.php` file. This file tells the system everything about the module, including a version number, what it should do when it is installed (that is, create some database tables), and what is should do when uninstalled (remove some database tables). Without this file, PyroCMS will not be able to load your module.

To help you get moving with module development, we're going to step through creating a module from scratch. This module will just be a sample, but it gets you on the road to making something great!

Just like a plugin, we'll start out project by creating a PHP class. The class's name, which will contain the name of the module, should be stored in a folder with the same module name. For example, if your module is named `Sample` it will be stored in a folder named `sample` and the class name defined in our `details.php` file should be `Module_Sample`. The following code will open our class:

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');
```

```
class Module_Sample extends Module {

}
```

The info() method

Now let's create our first method for this class. There are five methods required to be in our `details.php` class. The methods are as follows:

- info()
- install()
- uninstall()
- upgrade()
- help()

There is also one variable in our class, the version number. First, let's add the version number variable and the `info()` method to our module's base class, using the following code:

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');

class Module_Sample extends Module {

    public $version = '1.0';

    public function info()
    {
      return array(
          'name' => array(
              'en' => 'Sample'
          ),
          'description' => array(
              'en' => 'This is a PyroCMS module sample.'
          ),
          'frontend' => true,
          'backend' => true,
          'menu' => 'content',
          'sections' => array(
            'items' => array(
              'name'  => 'Items',
              'uri'   => 'admin/sample',
              'shortcuts' => array(
                'create' => array(
                  'name'  => 'Create an Item',
                  'uri'   => 'admin/sample/create',
                  'class' => 'add'
                )
              )
            )
          )
      );
    }
}
```

The `info()` method of our module class returns an array of information about our module, including (as you can see from the array keys) the name, description, whether or not it is a frontend or backend module, and where it will appear in the control panel's menu. The values in the `sections` area are returned as part of the `info()` method build links for the module in the sub-menu area of the control panel. This allows you to link to different parts (sections) of your module.

The install() method

The `install()` method in our `details.php` file will handle the initial database changes that may need to happen to help support your module. It can also add database information to support settings your module may need. To implement the `install()` method, add the following code as a method in the class we created in our module's `details.php` class:

```
public function install()
{
    $this->dbforge->drop_table('sample');
    $this->db->delete('settings', array(
      'module' => 'sample'));

    $sample = array(
      'id' => array(
      'type' => 'INT',
          'constraint' => '11',
          'auto_increment' => true
      ),
      'name' => array(
        'type' => 'VARCHAR',
        'constraint' => '100'
      ),
      'slug' => array(
        'type' => 'VARCHAR',
        'constraint' => '100'
      ),
    );

    $sample_setting = array(
      'slug' => 'sample_setting',
      'title' => 'Sample Setting',
      'description' => 'A Yes or No option for
        the Sample module',
      'default' => '1',
      'value' => '1',
      'type' => 'select',
      'options' => '1=Yes|0=No',
      'is_required' => 1,
      'is_gui' => 1,
      'module' => 'sample'
    );

    $this->dbforge->add_field($sample);
    $this->dbforge->add_key('id', true);

    if (!$this->dbforge->create_table('sample')
      OR !$this->db->insert('settings',
      $sample_setting))
    {
      return false;
    }

    return true;
}
```

Let's look at this method in detail. To prevent database errors, it starts by dropping any database tables associated with this module, just in case you've installed it before. Then two arrays, one named `$sample` and one named `$sample_setting`, are built to hold all the information necessary for the database forge library to make the database changes needed by our module, including the creation of a new database table called `sample`. It also adds place-holders for our module's custom system settings into our database's `settings` table (this feature is optional).

The uninstall() method

The `uninstall()` method you add to our `details.php` file is not nearly as complicated as the `install()` method. In fact, it usually only handles removing the database tables created by your module in the `install()` method. That is, for example, exactly what the following code is doing:

```
public function uninstall()
{
  $this->dbforge->drop_table('sample');

  $this->db->delete('settings', array('module' => 'sample'));

  return true;
}
```

Nothing too impressive here, just a simple database drop-and-delete to remove our `sample` database table and the settings values added into the `settings` table.

The upgrade() method

Adding an `upgrade()` method to your module's base class allows you to provide systematic upgrades to other developers who might use your module. Just include any upgrade logic/programming you might need to support version upgrades. If no update logic is required, you can just add the following method code to your class:

```
public function upgrade($old_version)
{
  // Your Upgrade Logic
  return true;
}
```

This basic method, which just returns `true`, will work for you until you need a way to upgrade your module for developers.

The help() method

The last method you'll add to the class in our `details.php` file is called `help()`. This method allows you to give some basic in-module pointers to PyroCMS users. Often over-looked, this is an easy way to help system users understand your module. A very basic help method looks like the following code:

```
public function help()
{
  return "Enter help info as HTML with paragraph
    tags, etc.";
}
```

Putting it together

Now that we've walked through all the methods needed in the `details.php` file, let's look at the code all put together as follows:

```
<?php defined('BASEPATH') or exit('No direct script
  access allowed');

class Module_Sample extends Module {

  public $version = '1.0';

  public function info()
  {
    return array(
      'name' => array(
        'en' => 'Sample'
      ),
      'description' => array(
        'en' => 'This is a PyroCMS module sample.'
      ),
      'frontend' => true,
      'backend' => true,
      'menu' => 'content',
      'sections' => array(
        'items' => array(
          'name'  => 'Items,
          'uri'   => 'admin/sample',
          'shortcuts' => array(
            'create' => array(
              'name'  => 'Create an Item',
              'uri'   => 'admin/sample/create',
              'class' => 'add'
            )
          )
        )
      )
    );
  }

  public function install()
  {
    $this->dbforge->drop_table('sample');
    $this->db->delete('settings', array('module' => 'sample'));
    $sample = array(
      'id' => array(
        'type' => 'INT',
        'constraint' => '11',
        'auto_increment' => true
      ),
      'name' => array(
        'type' => 'VARCHAR',
        'constraint' => '100'
      ),
      'slug' => array(
        'type' => 'VARCHAR',
        'constraint' => '100'
      ),
    );

    $sample_setting = array(
      'slug' => 'sample_setting',
      'title' => 'Sample Setting',
      'description' => 'A Yes or No option for the
        Sample module',
      'default' => '1',
      'value' => '1',
      'type' => 'select',
      'options' => '1=Yes|0=No',
      'is_required' => 1,
      'is_gui' => 1,
      'module' => 'sample'
    );

    $this->dbforge->add_field($sample);
    $this->dbforge->add_key('id', true);

    if (!$this->dbforge->create_table('sample') OR
      !$this->db->insert('settings', $sample_setting))
    {
      return false;
    }

    return true;
  }

  public function uninstall()
  {
    $this->dbforge->drop_table('sample');

    $this->db->delete('settings', array('module' => 'sample'));

    return true;
  }

  public function upgrade($old_version)
  {
    // Your Upgrade Logic
    return true;
  }

  public function help()
  {
    return "Here you can enter HTML, etc.";
  }
}
```

It might feel like this is a lot of code, but it is well organized, and it accomplishes a lot of tasks without too much effort. Building this `details.php` file is the first step in getting a module working. All we need now is a controller file to kick start this module and get it working.

Add a controller

There are two types of controllers you can add to a PyroCMS module, admin controllers and public controllers. As you can imagine, admin controllers can only be reached once you are logged into PyroCMS. Public controllers, though, are part of the public side of your website (that is, no login required). Only in rare cases will modules not have some type of admin controller. To create the base admin controller for your module, create a file named `admin.php` in your module's controller folder. Once you've created the file, we need to add the following code to get this controller working. Once again, this code is a PHP class.

```
<?php if (!defined('BASEPATH')) exit('No direct script
  access allowed');

class Admin extends Admin_Controller
{
  public function index()
  {
    // Start putting your module to work!
  }
}
```

Things you need to know about this class include seeing that it extends the `Admin_Controller` class. If this were a public controller, it would extend the `Public_Controller` class. Also, the `index()` method we've added to this class is the method run by default, making the `index()` method the first place you touch while building more advanced controller logic. From here you can begin to scale your module into a full-powered PyroCMS module add-on. You can start by adding a view to your controller and passing some variables to that view. The following code is a basic example:

```
<?php if (!defined('BASEPATH')) exit('No direct script
  access allowed');

class Admin extends Admin_Controller
{
  public function index()
  {
    $message = "You are logged in!";

      // Loads from addons/modules/blog/views/admin/
        view_name.php
      $this->template
      ->set('message', $message)
      ->build('admin/view_name');
  }
}
```

Passing data to a view is a foundational piece of using MVC. In this example, we pass the string, `You are logged in!`, to the view, but that data could just as easily be an array of values queried by a model.

Plugins and widgets in modules

One thing to remember is that module add-ons in PyroCMS can contain their own plugins and widgets. This is often how content is delivered from a module to the frontend of the website. For example, the controllers in your module will compile data ready to be sent to the client (browser) as HTML. This data can be used in a plugin to generate the HTML and place it within a page. Similarly, widgets can be added to modules to accommodate widget style content that's connected to your module.

Summary

In this chapter we learned how to create basic plugin and module add-ons for PyroCMS. Both of these tools are very frequently used in the system and can be the keys to helping you make PyroCMS, beyond the core features, do what you need. In the following chapter we'll explore what it takes to create a PyroCMS theme.

5
Creating a PyroCMS Theme

Frontend templates are always a central part of building a website on a content management system, and PyroCMS is no different. In this chapter, we'll explore all the details of making your own theme for PyroCMS, including where to place your theme files, how to use tags (Lex Parser tags) in your theme, and how to make your themes maintainable and efficient.

Folder structure

Before we start writing any theme code, it is important to know where how your theme files should be organized. Your PyroCMS theme can consist of HTML, JavaScript, CSS, and images arranged into the following folder structure:

- css
- img
- js
- views
- views | layouts
- views | partials
- views | modules

If you've ever built a theme for a CMS, then these folder names will look very familiar. Also, remember that with PyroCMS, we are always writing code in the MVC programming pattern, and themes are no exception. When building a theme for PyroCMS, you are really building the views (including assets) of a MVC patterned application. Views consist of a master layout file and multiple partial files (that is, a `header.html` or `footer.html`) that share presentation logic between different layouts. While that might sound confusing at first, it makes more sense as you put together your own PyroCMS theme.

Getting started

PyroCMS themes can be built in one of two places in the system. To get started building your first theme, create the supported folder structure in one of the following two places that themes may exist:

`addons/shared_addons/themes` (for themes available to all websites)

OR:

`addons/[site-name]/themes` (for themes available to only one specific website)

Once you have the base theme folder containing the supported folder structure built, you need to create a file named `theme.php` in your theme's folder. The following path, for example, is a file path that, from your base PyroCMS directory, will work for your `theme.php` file:

`addons/shared_addons/themes/[my-theme-name]/theme.php`

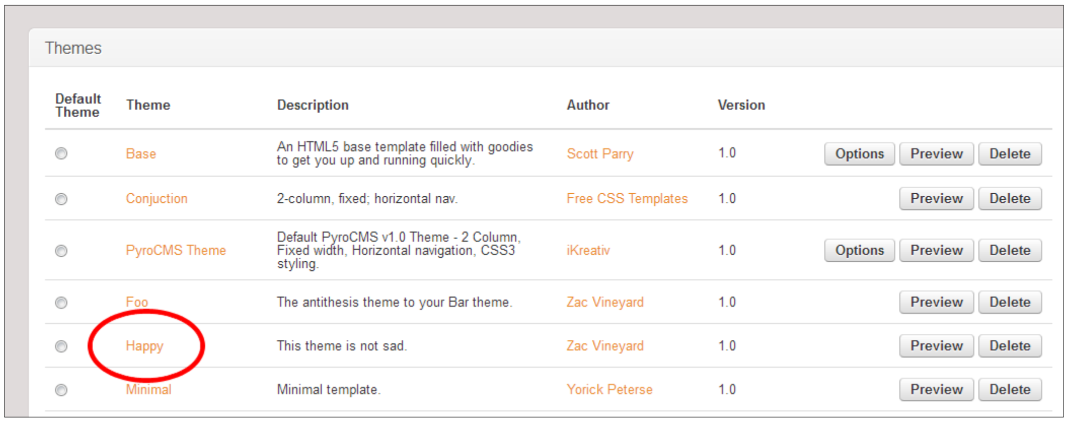
This `theme.php` file contains all the details of your theme for the system. These details include the theme's name, version number, author, author's website, the theme's website (often useful if the theme needs documentation), and a description. This `theme.php` file will also contain the necessary system hooks to enable advanced theme options.

Creating a theme.php file

A simple `theme.php` file's code looks like the following code:

```
<?php defined('BASEPATH') OR exit('No direct script
  access allowed');

class Theme_Happy extends Theme
{
  public $name = ' Happy ';
  public $author = 'Zac Vineyard';
  public $author_website = 'http://zacvineyard.com';
  public $website = 'http:// zacvineyard.com/themes/happy';
  public $description = 'This theme is not sad.';
  public $version = '1.0';
}
```

Like many of the other files you'll create for PHP, the `theme.php` file is a PHP class (surprise!). As shown, this class has a few specific properties that must be included (name, author, and so on.). How this class is named and stored determines whether or not the PyroCMS system will recognize it. The name of this theme is `Happy`, which means that it needs to be stored in a folder named `happy` (as shown in the following screenshot) and that the class in our `theme.php` file should be named `Theme_Happy`. If after you create your theme you don't see it listed in the **Theme** section of the control panel, check to make sure that the theme's folder name and class name are correct.



Themes

Default Theme	Theme	Description	Author	Version	
○	Base	An HTML5 base template filled with goodies to get you up and running quickly.	Scott Parry	1.0	Options Preview Delete
○	Conjuction	2-column, fixed; horizontal nav.	Free CSS Templates	1.0	Preview Delete
○	PyroCMS Theme	Default PyroCMS v1.0 Theme - 2 Column, Fixed width, Horizontal navigation, CSS3 styling.	iKreativ	1.0	Options Preview Delete
○	Foo	The antithesis theme to your Bar theme.	Zac Vineyard	1.0	Preview Delete
○	Happy	This theme is not sad.	Zac Vineyard	1.0	Preview Delete
○	Minimal	Minimal template.	Yorick Peterse	1.0	Preview Delete

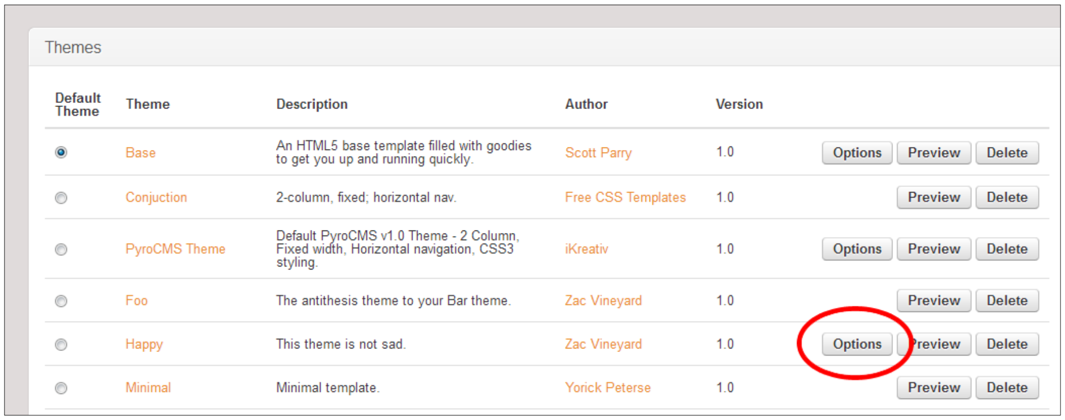
Theme options

One powerful feature in PyroCMS is the ability for your themes to have their own set of options (similar to the idea of module settings). These options can help your users control many dynamic features you could create into a theme. Your theme could, for example, give users the power to choose which color scheme should load or whether or not to include a breadcrumb navigation element at the top of a layout. To make use of options, we need to define the `$options` property in the main theme class in our `theme.php` file. Adding that property to the code in our file looks like the following code:

```
<?php defined('BASEPATH') OR exit('No direct script
  access allowed');

class Theme_Happy extends Theme
{
  public $name = ' Happy ';
  public $author = 'Zac Vineyard';
  public $author_website = 'http://zacvineyard.com';
```

```
     public $website = 'http:// zacvineyard.com/themes/happy';
     public $description = 'This theme is not sad.';
     public $version = '1.0';
     public $options = array(
       'show_breadcrumbs' => array(
         'title' => 'Show Breadcrumbs',
         'description' => 'Display breadcrumbs?',
         'default' => 'yes',
         'type' => 'radio',
         'options' => 'yes=Yes|no=No',
         'is_required' => TRUE
       )
     );
   }
```

The `$options` property we've added is an array of values that describe the type of options you are giving to the user. In our example, the options we're giving the user is named `Show Breadcrumbs` and will display a radio button input in the theme options window for them to make that choice. The following screenshot shows the **Options** tab highlighted:


Themes

Default Theme	Theme	Description	Author	Version	
●	Base	An HTML5 base template filled with goodies to get you up and running quickly.	Scott Parry	1.0	Options Preview Delete
○	Conjuction	2-column, fixed; horizontal nav.	Free CSS Templates	1.0	Preview Delete
○	PyroCMS Theme	Default PyroCMS v1.0 Theme - 2 Column, Fixed width, Horizontal navigation, CSS3 styling.	iKreativ	1.0	Options Preview Delete
○	Foo	The antithesis theme to your Bar theme.	Zac Vineyard	1.0	Preview Delete
○	Happy	This theme is not sad.	Zac Vineyard	1.0	Options Preview Delete
○	Minimal	Minimal template.	Yorick Peterse	1.0	Preview Delete

Because our `$options` property is an array, we can add multiple theme options for users. To do that, just append another array item into the `$options` array that helps users make a choice with some of your theme's settings.

Building the theme options form

In our `$options` array, you'll notice an array key named `options` after the `type` key and the `is_required` key. The value of that `options` key may look a little cryptic to you, but I can assure you that it is an easy code to break. You can customize field values (as seen with our radio button) by defining a list of values and their text labels separated by a pipe character (`|`).

Theme layouts

All layouts files for a PyroCMS theme exist in one of the following two locations:

`addons/[site-name]/themes/[my-theme-name]/views/layouts/` (for themes available to all websites)

OR:

`addons/shared_addons/themes/[my-theme-name]/views/layouts/` (for themes available to all websites)

Every theme should have a layout file named `default.html` in the preceding locations listed. Let's review the contents of a layout file. Layout files in PyroCMS are built using HTML and a tag parser, the Lex Parser. The following code is what a very basic PyroCMS layout file looks like:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ template:title }}</title>
    {{ template:metadata }}
  </head>
  <body>
    <h1>{{ template:title }}</h1>
    {{ template:body }}
  </body>
</html>
```

The first thing you should notice about this code is that it frequently makes use of tags. These tags, as seen in the header of this document, give you a number of content and logic options for your theme's layout files. In the case of this code, the `template` tag is creating paths to CSS and JavaScript files included in this template. Let's dive further into the use of tags in PyroCMS.

Tags explained

Earlier in this book we took some time to look at PyroCMS Tags (Lex Parser tags). While what we covered was important, it wasn't very thorough. Knowing that you'll be using tags heavily in templates, it will be good to have an in-depth knowledge of how they work. Tags allow you to use advanced logic, with a special syntax, right inside your layouts, page layouts, and pages (that is a WYSIWYG). Tags keep you from filling up your templates with large PHP loops or other clutter, keeping your templates clean and useable.

Basic tag

A basic tag can do something simple, like return a variable, or echo a variable into your HTML. The following tag a basic example:

```
{{ some_var }}
```

PyroCMS tags are expressed with the preceding syntax, two curly brackets followed by variable declarations or attributes, and closing brackets. You must remember, though, tags like this are usually powered by a plugin, which is, in PyroCMS, a PHP class with a method that will return data. So, you can have a plugin return data for you. The following tag, for example, is a tag that will return the URL a user is currently using:

```
{{ url:current }}
```

Truly, this tag could be connected to any module in PyroCMS, but it is connected to a plugin called `url` which is a class. That class has a method named `current` that will return the URL of the current page. This type of functionality is part of the reason why PyroCMS is so powerful. The thing is, though, it doesn't stop here. Tags can take attributes and handle more advanced logic.

Tag attributes

Lex Parser tags give you the power to modify tag output based on input data. That input data comes from tag attributes. Building on our URL plugin example, the following tag is a tag that will return just the first path segment of a URL:

```
{{ url:segments segment="1" }}
```

Just as before, this tag hooks a plugin that returns some data, but this time takes into account the value we send using the `segment` attribute. If our URL were something like `http://example.com/hello/world`, then this tag returns the string `hello`. You can also use multiple attributes with the same tag, as seen in the following example:

```
{{ url:segments segment="1" item="first" }}
```

Adding another attribute to a tag is as simple as it is in HTML.

Using tags in tag attributes

This topic title might sound a little confusing but it makes more sense when you see the following tag example:

```
{{ url:segments segment="1" default="{{ page:slug }}" }}
```

What you are seeing with this example is that it is possible to nest tags inside other tags. This means that you can, for example, use the output from one tag as an attribute (input) value for another tag. That's what's happening in the preceding string of code. We're using the output from `{{ page:slug }}` as the attribute value `default`. This type of tag usage, as you can imagine, is what makes tags so flexible and powerful.

Tag pairs

Speaking of HTML, there is a way beyond attributes in which PyroCMS tags are like HTML. Tags can be parsed with or without closing pair tags. Very similarly to what's found in HTML and XML, PyroCMS tags can also take advantage of closing pair tags. Take a look at the following code example:

```
{{ blog:posts limit="2" order-by="title" order-dir="desc" }}
    <h2>{{ title }}</h2>
{{ /blog:posts }}
```

There are a number of things to consider when looking at this code. First, this tag obviously hooks a plugin in a blogging module that returns some data from a method named `posts`. The attributes in this tag define some parameters for the method, and, of course, we get some data output. One new feature you see is the closing pair tag `{{ /blog:posts }}`. Because of this closing pair tag, we can use PyroCMS tags to loop over data, as seen in our example. This leads us to other ways we can express logic using PyroCMS tags.

Tag conditionals

Tag conditionals allow us to write basic conditional statements using tags. This feature is very simple to use and has a syntax that's very similar to many of the if/else statements found in programming languages, and is shown as follows:

```
{{ if user:logged_in }}
<p>You are logged in.</p>
{{ endif }}
```

The conditional statement in this tag is, of course, looking to see if a user is logged in to your website. Nothing too fancy here, but tags can get more complicated by handling if/else logic shown in the following example:

```
{{ if user:logged_in }}
<p>You are logged in.</p>
{{ else }}
<p>You are not logged in.</p>
{{ endif }}
```

There are a number of other ways you can use tags in PyroCMS, but this covers the basics. If you want to learn more about tags and other advanced tag syntax, you can find more info in the PyroCMS documentation at `http://www.pyrocms.com/documentation`.

Tags in templates

The special tags you see in our layout HTML are Lex Parser tags. Because we are writing code in the MVC pattern, the primary benefit to using Lex Parser tags in your layout files is that you don't have to put PHP directly in your views, which gives you the best chance of creating PyroCMS themes that follow the don't repeat yourself (DRY) protocol.

Let's take a look at a more complex `default.html` layout file, shown as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ template:title }}</title>
    {{ template:metadata }}
    {{ theme:favicon file="favicon.png" }}
    {{ theme:css file="style.css" }}
    {{ theme:js file="site.js" }}
  </head>
  <body>
    <div class="header">
      <div class="logo">
```

```
          {{ theme:image file="logo.jpg" alt="Your
            Happy Logo" }}
        </div>
        <div class="nav">
          {{ navigation:links group="header" }}
        </div>
      </div>
      <div class="content">
        <h1>{{ template:title }}</h1>
        {{ template:body }}
      </div>
    </body>
  </html>
```

As I've already stated, you'll notice that this layout file uses tags, including assets like CSS, JavaScript, and images. Using tags in HTML allows PyroCMS developers to build layout files very quickly.

Layouts versus page types

Layouts in PyroCMS are, in a way, best described as the most traditional route to organizing content in a theme. The way in which layouts are built and used in PyroCMS, for example, parallels the process we find in many other content management systems. So, in order to see a custom set of data from a database, we usually build a layout file as part of our theme to handle the display of that data.

Taking a less traditional route, PyroCMS 2.2 included a new feature, page types. A page type is a layout that's connected to a specific stream of data, so that when you create a page, you also input the data needed for a database entry. Page types connect the data's point of entry to the page, instead of having to keep it separated in a module (or another part of the system), as in our traditional layout example.

Theme partials

Partials are partial layout files in PyroCMS, and they allow you to break layouts into reusable sections. These sections can then be loaded by different layout files. This keeps you from typing the same code (header, footer, and so on) into multiple layout files. Depending on where you've placed your theme files, partials are created in one of the following two locations:

`addons/[site-ref]/themes/[my-theme-name]/views/partials/`

OR:

`addons/shared_addons/themes/[my-theme-name]/views/partials/`

Partials are loaded into layouts using a tag shown as follows:

```
{{ theme:partial name="partialname" }}
```

This tag operates exactly like a PHP include statement, similar to one that you would find in themes of other content management systems. The following code is a simple example of a PyroCMS layout that takes advantage of partials:

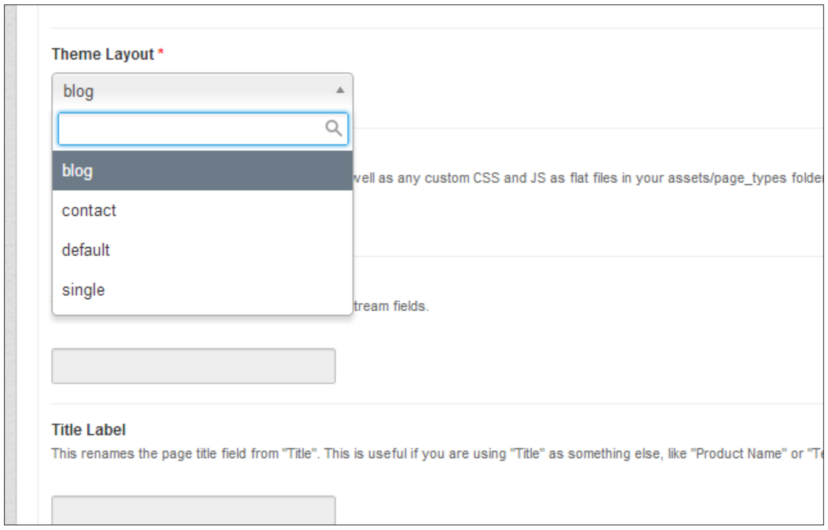
```
{{ theme:partial name="header" }}

  <div class="content">
    <h1>{{ template:title }}</h1>
    {{ template:body }}
  </div>

{{ theme:partial name="footer" }}
```

The contents of the `header.html` partial and `footer.html` files are, of course, the HTML we'll need to reuse from the template in our preceding code example. One quick pointer: there is no limit to the number of partials that you can use in one layout. Additionally, partial files may contain any combination of valid HTML and tags.

Multiple layouts

Templates in PyroCMS aren't limited to just one layout file. You can make use of as many layout files as you want, as long as you create them in the right way. To create a specific layout for your website's about page, for example, you can create another file like your `default.html`, but naming it `about.html`. It is important to give this file a descriptive name so that users do not become confused when choosing a layout for their pages in the system.

When you edit or create a `Page Type` in your PyroCMS control panel (navigate to **Control Panel | Pages | Page Types**) and select your desired file from the dropdown, all the layouts in your theme's layout file will be available to use, as shown in the following screenshot:



vell as any custom CSS and JS as flat files in your assets/page_types folde

tream fields.

Title Label

This renames the page title field from "Title". This is useful if you are using "Title" as something else, like "Product Name" or "Te

Mobile layouts

PyroCMS is able to easily display separate layouts for mobile and desktops. To use this feature, move your layout files into a folder, called `web` within the `views` folder, so that your default layout will be located in the following location:

```
[your-theme]/views/web/layouts/default.html
```

When a user accesses your website using a desktop browser, the primary layout files in this location will be used. If the user accesses your website using a mobile device browser, users will be supplied with the mobile layouts that you've created in the following location:

```
[your-theme]/views/mobile/layouts/default.html
```

This feature works with multiple layout files, but please take note of this warning found in the PyroCMS documentation: "PyroCMS does not consider the iPad a mobile device, so it will not load your mobile layouts if the user is accessing your website using an iPad." If, however, on your website, you'd like to make an iPad recognized as a mobile device, you can change the `user_agent.php` file within the system's `config` directory to make the iPad recognized a mobile device.

The folder structure of your theme's directory should look as follows, after you adopt mobile layouts:

- css
- img

- js
- views
- views | mobile | layouts
- views | mobile | partials
- views | mobile | modules
- views | web | layouts
- views | web | partials
- views | web | modules

Module view overloading

Don't like the default theme layout for the blog module in PyroCMS? Hate the way the comments are output by the system? No problem. You can customize them anyway you wish by overloading the views associated with those modules. PyroCMS allows you to replace any module view with a view inside your theme. If, for example, you'd like to use a custom layout for the initial page where blog posts are displayed, simply copy the blog module's `posts.php` view file to the theme location described as follows:

```
system/cms/modules/blog/views/posts.php
```

to

```
addons/[site-ref]/themes/[theme-name]/views/modules/blog/posts.php
```

Once this view file is in your theme, you can change it to fit your needs. Then your new view will be used instead of the default view included in with the blog module.

Summary

While the information in this chapter may look tedious at first glance, once you start diving into building PyroCMS themes, you won't look back. As our code example at the end of this chapter shows, you can build themes for PyroCMS quickly, and the code stays organized! When I'm faced with working in a system that isn't PyroCMS, I'm always jealous of how easy it is to work in this theme development environment. In the following chapter, we'll discover how to take all the hassle out of managing database tables and interaction with streams.

6

Using PyroCMS Streams

In earlier versions of PyroCMS (the current version is 2.2), PyroStreams was a module add-on only found in the professional version. Recently though, the system has changed quite a bit. One substantial change is that the system has been depending more on Streams, as they're now known, to power many of the advanced, database driven features of PyroCMS. Streams are powered by a `data stream` module in PyroCMS that takes all the hassle out of managing database tables. This module was invented because websites so often require sets of data that aren't just HTML, such as a directory of employees or a database of products and product features.

In this chapter, we are going to learn how to use streams in PyroCMS. Using this module and its plugin will help you manage and display structured data on your website.

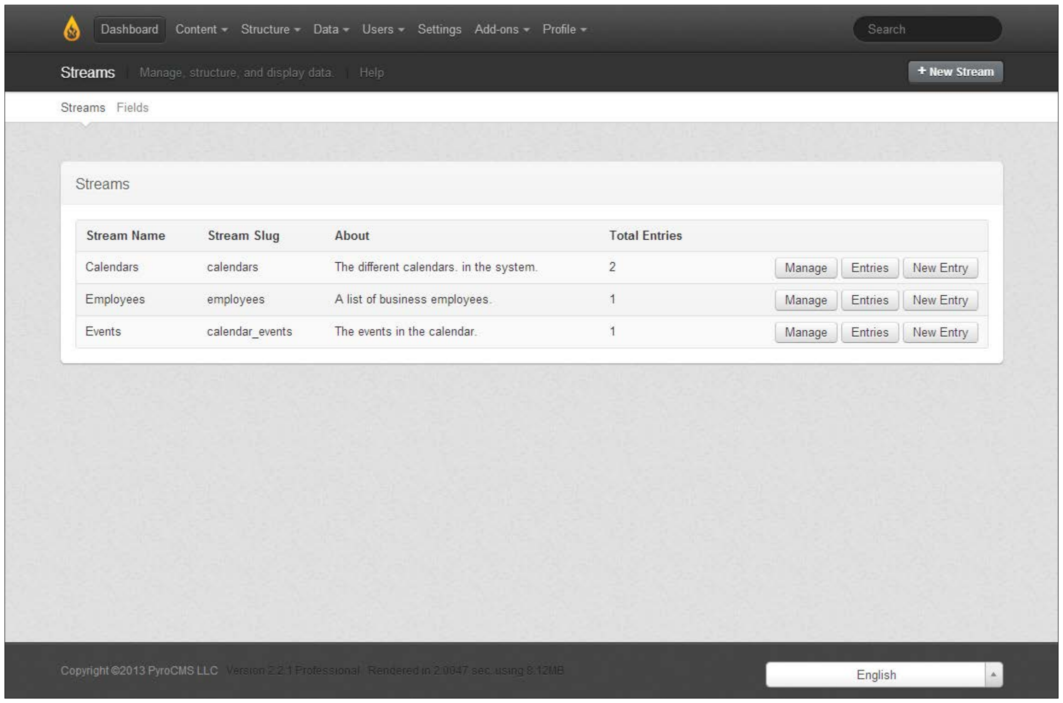
How to get Streams

To have all the features of the `Streams` module, you'll want to download and use the professional version of PyroCMS. The core of the `Streams` module is part of the community version of PyroCMS (page types, for example, are powered by streams), but you can't get the full benefit from streams without the professional version. If you'd prefer to keep the community version, you can purchase the `Streams` module from the PyroCMS add-on store (`https://www.pyrocms.com/store`) and install it in PyroCMS.

Also, it is important not to get confused over the similarities between page types and the traditional use of streams. In some ways, page types can replace streams (they are powered by the same module, after all), but many developers prefer using page types and the `Streams` module for different purposes. As we learn about streams, and when you start using them for your own project, you'll gain a preference for how you'd like to use streams, even though the ways you can use streams are all very similar.

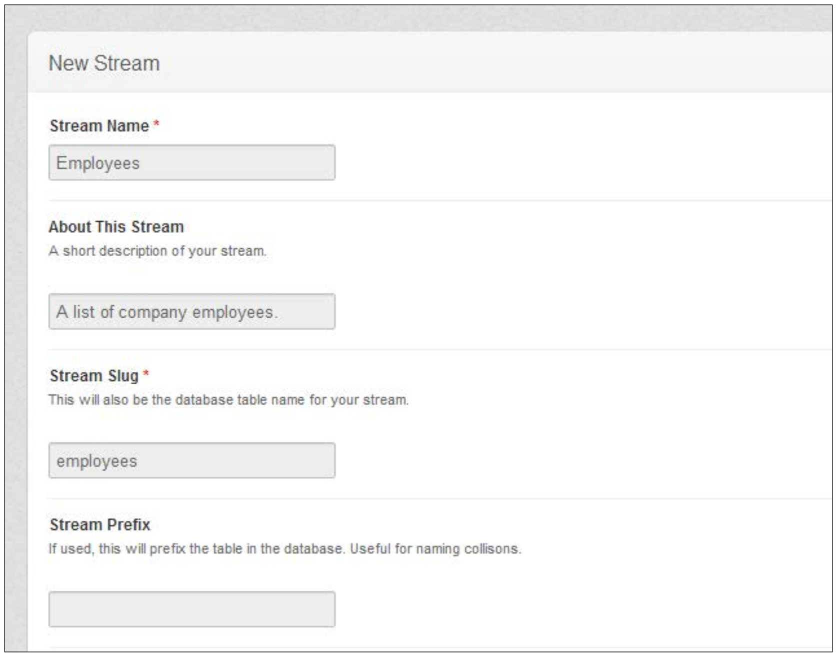
Creating a stream

To create a stream in PyroCMS, you'll use the control panel interface for streams just as you would for any module. This interface allows you to easily manage your streams and their input fields. At first, it might not be completely obvious how you use the interface, but once you put it into practice, you won't forget. The following screenshot shows how to create a new stream:



Stream Name	Stream Slug	About	Total Entries			
Calendars	calendars	The different calendars. in the system.	2	Manage	Entries	New Entry
Employees	employees	A list of business employees.	1	Manage	Entries	New Entry
Events	calendar_events	The events in the calendar.	1	Manage	Entries	New Entry

To create a new stream after you've selected the `Streams` module from the main control panel menu, click on the **New Stream** button near the top-right of your window. You'll be prompted to fill out a form with the details of your stream, including its name, description, slug, stream prefix, and menu path. For now, just give this new stream a name, description, and slug (must be unique). Remember, a stream is just a set of relational database tables that store data. The following screenshot shows the data that you need to enter for the stream:


New Stream

Stream Name *

Employees

About This Stream

A short description of your stream.

A list of company employees.

Stream Slug *

This will also be the database table name for your stream.

employees

Stream Prefix

If used, this will prefix the table in the database. Useful for naming collisons.

Once you have a stream created, we'll need to create some data input fields so that you or your users can add data to the stream. In our stream example, we're creating a list of business employees. So, you can imagine that we'll need input fields for an employee's name, title, e-mail address, and phone number. The data we'll be adding is fake, of course, but this serves as a basic example of the type of data you'll need. So, let's dive in to making these fields.

Creating fields

To create a new input field for your stream, click on **Fields** in the `Streams` module's submenu, then click on the **New Field** button near the top-right of the window. This button leads you to a form that asks you for a field's name, slug, and type. We'll need to complete this form to create a new input field for every column of data (remember, this is just a database table) needed in our stream. One thing you'll notice, though, is there are a number of different input types. While some input types are obvious, such as country and date/time, some input types have a purpose that is a little more confusing. Just refer the following list for each input type and its purpose:

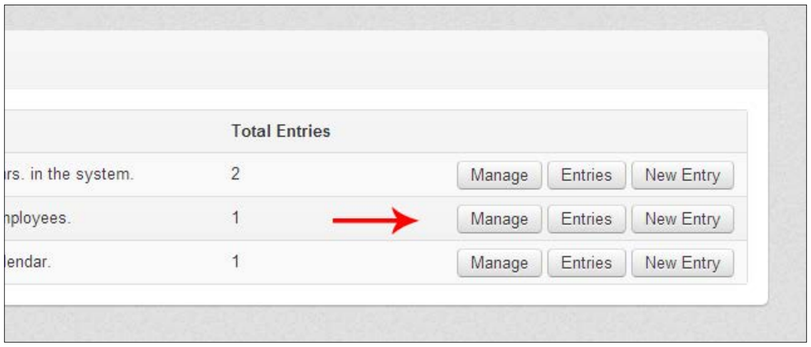
- **Country**: presents the user with a list of countries.
- **Date/Time**: gives the user a group of drop-downs for inserting a date and time.

- **Email**: gives the user a way to enter and validate an e-mail address as part of the entry.
- **Encrypted**: gives the user a way to store a string as encrypted data in the database. Data can be decrypted for output.
- **File**: gives the user a way to upload a file as part of a streams entry.
- **Image**: presents the user with a way to upload and resize an image as part of a stream entry.
- **Integer**: allows the user to store an integer value.
- **Keywords**: the keywords field type gives you a text field into which you can enter keywords or tags for your entries.
- **Multiple Relationship**: presents the user with a list of the other streams to create a one-to-multiple relationship between streams. For example, you could connect more than one phone number to an employee.
- **PyroCMS Language**: shows a drop down of languages from which to choose.
- **Relationship**: presents the user with a list of the other streams to create a one-to-one relationship between two streams.
- **Slug**: presents the user with a single-line text input field that can be hitched to another text input field for slug creation.
- **Text**: gives the user a simple, single-line text input field.
- **Textarea**: gives the user a large `textarea` input field, commonly used for larger blocks of text.
- **URL**: gives the user a text input field specifically used for URLs.
- **US State**: presents the user with a list of states that are part of the United States of America.
- **User**: presents the user with a list of users in the PyroCMS system.
- **WYSIWYG**: gives the user a textarea input field coupled with either a simple or an advanced WYSIWYG.
- **Year**: presents the user with a list of years from which to pick.

Assigning fields

After you have taken time to create all the fields for your stream of data, you'll need to assign those fields to your stream. To help keep things efficient, fields have been engineered inside the `Streams` module to be reusable. This means that any field can be reused as part of any stream. So, we need to pick the fields we want to use for our employee stream and assign them to that stream. To do that, click on the **Manage** button next to your stream on the stream module page.

The following screenshot shows how to assign fields to the stream:



	Total Entries	
rs. in the system.	2	Manage Entries New Entry
nployees.	1	Manage Entries New Entry
endar.	1	Manage Entries New Entry

Click on the **Stream Field Assignments** link on your stream's management page. Then click on the **New Field Assignment** button near the top-right of the window. Take time to assign each of the fields you'd like to have as input to our employee stream. When making the assignment, you can choose to make the input field required (as part of the input form) or a unique value. You can also define help instructions for a field and choose to make the field a title column.

Each stream should have a `title` column. This column is the field that most represents the data rows within each stream, such as a name or a title. Among a number of things, it is primarily used to represent the rows of a field when displaying posts in a relationship drop-down.

Ordering fields

Ordering fields on your streams input form is done easily. On the input assignment page you can simply drag-and-drop your fields to reorder them, which will automatically reorder the fields on the input form.

Default columns (fields)

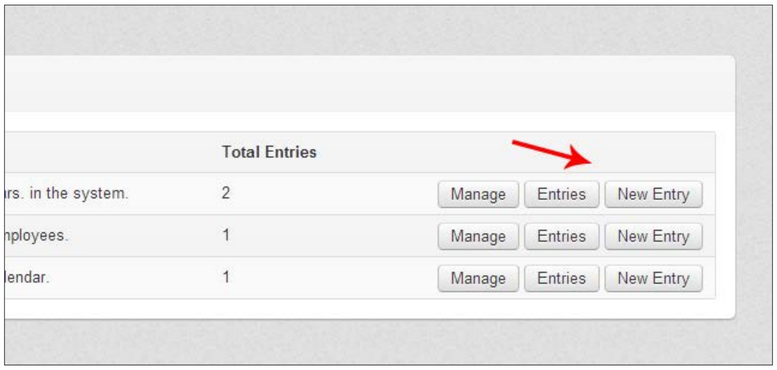
Each stream comes with some default columns that are always created. You don't need to create these because they will automatically be available as a part of your stream. The default columns are as follows:

- **ID**: incremental numerical ID
- **created**: date of when the entry was created
- **updated**: date of when the entry was last updated
- **created_by**: ID of the user that created the entry

These fields can be used just like any other field in your stream, which means that they can be and displayed in templates, and so on.

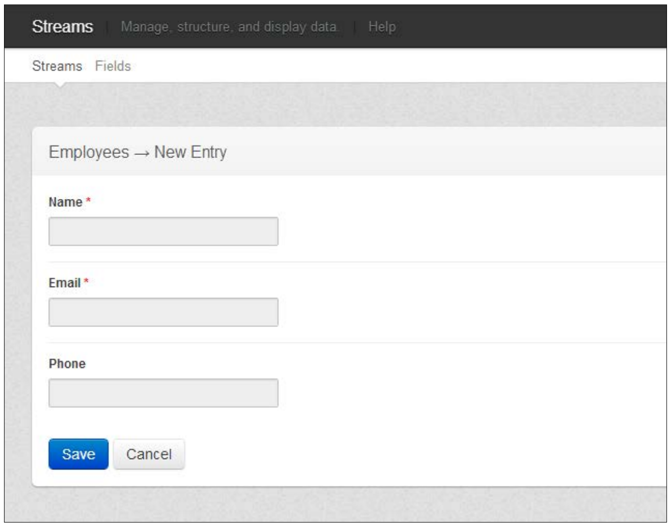
The backend input form

After you've assigned your input fields to your steam, you'll be able to start inputting data using the provided input form. To input an entry in your stream, click on the **New Entry** button next to the **Manage** button on the `Streams` module page. The following screenshot shows the **New Entry** option that you need to click on:



	Total Entries			
rs. in the system.	2	Manage	Entries	New Entry
nployees.	1	Manage	Entries	New Entry
lendar.	1	Manage	Entries	New Entry

You should then be taken to a form that contains all the input fields you assigned to your stream. All you need to do, then, is fill out the form to add data to your stream. In my example, I've assigned a name and e-mail address field to our `employees` stream so that for every employee record I add, each person will have an e-mail address and a name. The following screenshot shows the fields for your stream:



Displaying your data

Once you have data stored in a stream, the next obvious step is to display that data back out for users, in accordance with specific queries. Using the stream of employee data we've created as an example in this chapter, I'll show you how to use this data on your website.

The key to displaying data on your website is a powerful plugin that comes packaged with the `Streams` module.

Streams plugin

This plugin, like all other plugins, lets us use tags to embed data into any page or theme layout we want! It also has tools for creating entry forms and other types of content.

Common variables

Before we dive too deep into using the `Streams` module, it is important to know that there are a number of commonly used variables available to use in your plugin tags. These variables are wrapped in square brackets and are as follows:

- **[segment_1]**: value in the first URI segment
- **[segment_2]**: value in the second URI segment
- **[segment_3]**: value in the third URI segment
- **[segment_4]**: value in the fourth URI segment
- **[segment_5]**: value in the fifth URI segment
- **[segment_6]**: value in the sixth URI segment
- **[segment_7]**: value in the seventh URI segment
- **[user_id]**: currently logged in user's ID
- **[username]**: currently logged in user's username

You can imagine times where these variables may come in handy, so don't forget about them.

Entry looping

The most fundamental way in which you will display stream data to a user is to use a tag to loop over stream entries and print data to a page. There are a number of stream plugin features that help with handling stream data, each of which is explained, with code examples, in the following paragraphs.

The loop cycle

The most basic and common way of interacting with streams data is looping through it using the `cycle` plugin function. The following code is a very simple example where we display five entries from our `employees` stream:

```
{{ streams:cycle stream="employees" limit="5" }}

  <h2>{{ name }}</h2>

{{ /streams:cycle }}
```

The preceding code, as you can plainly see, will cycle (or loop) through our stream entries and output the name we've given each employee. This is a simple example. Tags that output stream data can get much more complex.

Filter by date

One way in which tags for stream data get more complex is by adding a date filter. The following code is an example of a tag that uses a day, month, and year to return specific stream records created within a date range:

```
{{ streams:cycle stream="employees" year="2012"
  month="01" day="15" }}
```

You can also restrict the entries shown by telling the `cycle` plugin to show or not show entries before or after the current date. You do this by using the `show_upcoming` or `show_past` parameters, shown as follows:

```
{{ streams:cycle stream="employees" show_past="no" }}
```

Other parameters

Dates are not the only parameters you can use in stream tags. The following is a comprehensive list of all the parameters available for stream tags. Some of these parameters are required.

- **stream**: slug of the stream you want to cycle through.
- **namespace**: by default, streams work off the core streams `namespace`, but you can change this to use the `cycle` function with other namespaced streams.
- **limit**: how many results to which your query is limited.
- **offset**: the result set offset number.
- **order_by**: the field used to order your results.
- **sort**: the sorted order of your results, either ascending (asc), descending (desc), or random.
- **date_by**: the field through which date parameters are run. The default is `created`.
- **year**: restricts results to a year (in the `date_by` field).
- **month**: restricts results to a month (in the `date_by` field).
- **day**: restricts results to a day (in the `date_by` field).
- **show_upcoming**: choose yes or no to show entries dated in the future.
- **show_past**: choose yes or no to show entries dated in the past.
- **where**: used to restrict results, just like it is in a SQL query.
- **exclude**: IDs of entries to exclude separated by a pipe character (|).
- **exclude_called**: set to `yes` to limit entries to ones not already displayed on the same page.
- **include**: value to include in a `where=` clause. Used with `include_by` to filter by a single data point.
- **include_by**: the field to use when using the `include` parameter.
- **disable**: allows you to disable fields and their formatting. You can specify multiple fields by separating them with a pipe character (|).
- **no_results**: message that displays when no results are found, such as **No results were found**.
- **partial**: allows separation of results into separate segments.

How to use the "where" parameter

If you have a fair bit of experience writing SQL queries, then using the `where` parameter in your stream tags will feel very comfortable. Writing out this `where` parameter is very much like what an actual SQL query feels like. The following code is an example:

```
{{ streams:cycle stream="employees"
  where="`email`='test@test.com'" }}
```

Appending a `where` parameter into your tag parallels the `where` statement in a query, giving you power over displaying small or single sets of data.

Nested variables

Until now, we've been looking at stream tag examples that are only working with a single dimension of data. That is, each of the tag examples we've used so far have only been using a single-dimension array. Stream tags can output data from multidimensional arrays, just in case you have variables that return an array of values. For example, if you have a user fiend named `person`, you could access the e-mail shown as follows:

```
{{ streams:cycle stream="employees" limit="5" }}

    <h2>{{ person:email }}</h2>

{{ /streams:cycle }}
```

The preceding tag, as part of a stream loop, pulls a value from the second array dimension. This type of data output implies, of course, that your stream data is output as a multidimensional array.

Pagination

A feature often used by developers when displaying data is pagination. Paginating data allows users to more easily consume and use data. The good news is stream tags support pagination. The following parameters are used in the `cycle` function for creating pagination:

- **paginate**: setting this to `yes` will enable pagination
- **pag_segment**: the URI segment to take the pagination offset from (use an integer value, such as 2)

By default, when you enable pagination, the system will return 25 entries. You can override this by setting a limit parameter. The following code is a complete streams tag example where pagination is being put to use:

```
{{ streams:cycle stream="employees" limit="5"
  paginate="yes" pag_segment="2" }}
  {{ entries }}
    <h2>{{ name }}</h2>
  {{ /entries }}
  {{ pagination }}
{{ /streams:cycle }}
```

Stream data entry form

One of the most powerful features of PyroCMS's `Streams` module is the ability for data entry forms to be generated using a simple, powerful plugin. Being able to build forms for each stream takes away a lot of the horror of working with forms, especially on large websites. In the following small sections, I'm going to show you how to generate and use stream input forms.

How to build the form

Because stream entry forms are powered by a plugin, you can embed them into any page layout, page (whether public or private), or theme layout file. The tag we'd use to generate an input form for our `employees` stream looks like the following code:

```
{{ streams:form stream="employees" mode="new" }}
  {{ form_open }}
    <table>
      {{ fields }}
        <tr class="{{ odd_even }}">
          <td width="250">{{ input_title }}{{ required }}
            <small>{{ instructions }}</small></td>
          <td>{{ error }}{{ input }}</td>
        </tr>
      {{ /fields }}
    </table>
  {{ form_submit }}
  {{ form_close }}
{{ /streams:form }}
```

The most important things to realize about this tag are the two parameters, `stream` and `mode`. The `stream` parameter on this tag defines which stream the input will be generated for. The `mode` parameter, which can either be set to `new` or `edit`, defines whether you are creating a new entry for the stream or editing an existing entry. This form is creating new entries for our `employees` stream. You can, of course, modify the HTML in this tag to make the form more useable for your website or application.

When we look at this tag in more detail, you'll notice that it essentially loops over the fields assigned to a stream and builds the corresponding input element for that field. There are a number of tag parameters, however, that can change how your form operates. The following is a list of parameters you can use when using a tag to generate a stream input form:

- **stream**: required. Stream slug for the stream we are editing or creating data for.
- **mode**: set either to `new` or `edit`, depending on if you are creating or editing data.
- **edit_id**: required if editing an entry. ID of the entry to edit.
- **where**: allows you to specify a `where` parameter using `field_slug==value`.
- **required**: string that populates the required variable if the field is required.
- **return**: the location to which the user will be redirected once the form action is complete.
- **error_start**: string that prepends error messages.
- **error_end**: string that appends error messages.
- **include**: fields to include in the form, separated by pipe characters (|). If you specify fields here, all other fields (excluding default columns) will be excluded.
- **exclude**: fields to exclude from the form, separated by pipe characters (|).
- **use_recaptcha**: activates reCAPTCHA. Set to either `yes` or `no`.
- **creator_only**: when using the form in edit mode, setting this to `yes` will only show the form and allow editing of an entry if the `creator_id` matches the logged in user's ID.

Custom success and error messages

Good news! Most PyroCMS themes have built-in displays for flash data (that is, data that is only available on the next page refresh, and usually contains a message about the success/failure of the previous action). With regard to error messages output by stream forms (validation errors), you can change what these flash messages say by making use of the following two tag parameters:

- **success_message**: overrides flash success message content
- **failure_message**: overrides flash failure message content

Form assets

Many form fields have CSS or Javascript that needs to be loaded. Depending on your theme, the assets may not be automatically added to your page. In this case, you can add them manually with the `form_assets` function. Just place the following code below your form tags:

```
{{ streams:form_assets }}
```

E-mail notifications

When an entry form is submitted and processed successfully, you can send an e-mail to a user. The entry form e-mail notification feature hooks into the native PyroCMS e-mail template module. To get started, you'll need to navigate to **Design | Email Templates** in the control panel and create a new e-mail template. After you've built an e-mail template, you'll need to define the following parameters on your form tag:

- **notify_{ID}**: the e-mail address (or addresses) that will receive the notification. Multiple addresses can be separated by a pipe (|) character. You can also use form input values here, so if you have a field called `user_email`, you can use `user_email` as an e-mail value.
- **notify_template_{ID}**: the slug of the e-mail template to use.
- **notify_from_{ID}**: optional custom "from" e-mail. This can be a single e-mail address or an e-mail address and name separated by a pipe (|) character.

When you put it all together, the form tag that sends a confirmation e-mail resembles the following code:

```
{{ streams:form stream="messages" mode="new"
  notify_a="admin@example.com|user_email"
  notify_template_a="new_feedback"
  notify_from_a="noreply@example.com|Example" }}
```

Summary

Using streams in PyroCMS can take a lot of the hassle and development time out of building the forms and database interactions required by your website or web application. Without much effort, you can be building forms, database tables, and more by using the `Streams` module in PyroCMS.

7 Building a Website with PyroCMS

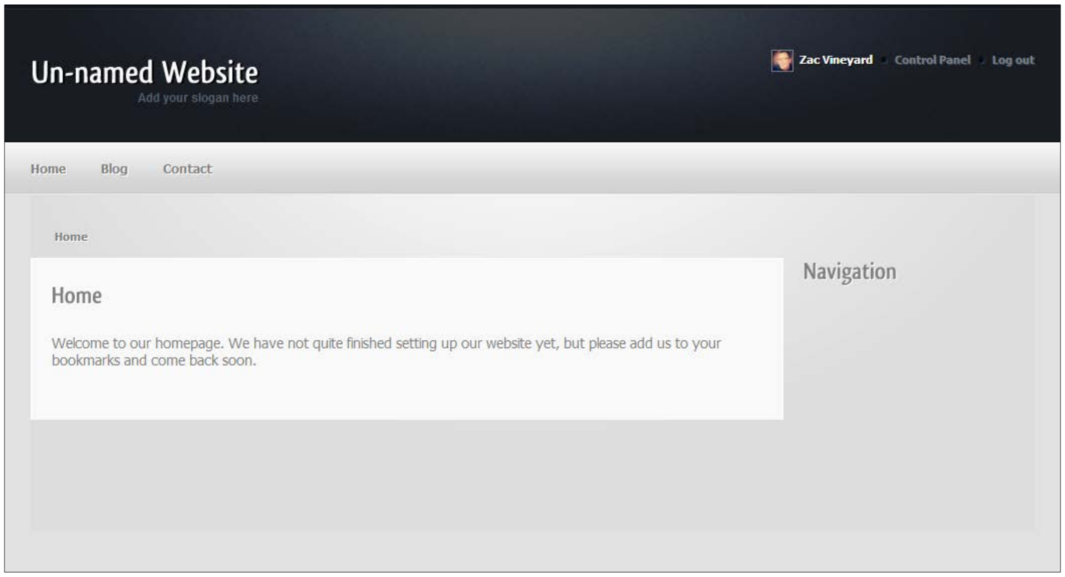
In all of the previous chapters of this book, we learned how to install and use all the primary features of PyroCMS. We took time to explore how to get the system running, how to create themes using both page layouts and template files, how to create add-ons for the system, and how to use streams to power data on our website. In this chapter, we're going to step through all of these processes one more time, giving you a consolidated website development experience using PyroCMS. We're going to build a website as if you were starting from scratch.

Installation

As part of this walk-through, I'm going to assume that you've downloaded PyroCMS (either from `www.pyrocms.com` or GitHub) and installed it on a local development server. For the following code examples, we're going to assume that your PyroCMS installation is in a web directory shown as follows:

```
http://localhost/pyrocms/
```

Starting out, after you've stepped through PyroCMS's built-in installation process, you should be able to load PyroCMS's default theme (as seen in the following screenshot):



Home

Welcome to our homepage. We have not quite finished setting up our website yet, but please add us to your bookmarks and come back soon.

If you can't get your website to load, the PyroCMS forums at `https://forum.pyrocms.com` often has posts that can help you troubleshoot. Once your website is loading, you can start developing a custom theme for your website.

Creating a custom theme

To start building your theme, there are a few folders you need to create in your default theme folder. Your default theme folder should be created at the following location:

`addons/shared_addons/themes` (for themes available to all websites)

OR:

`addons/[site-name]/themes` (for themes available to only one specific website)

Create the following folders inside your theme directory:

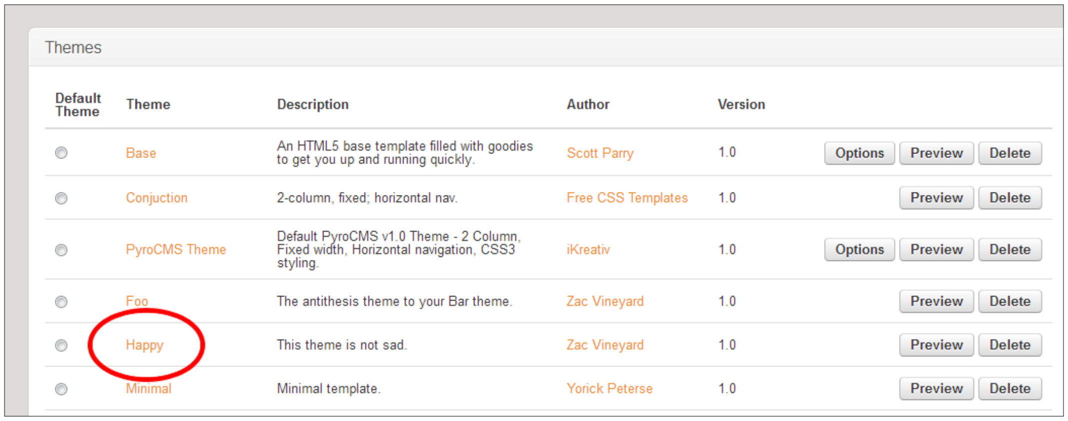
- css
- img
- js

- views
- views | layouts
- views | partials
- views | modules

Once you have these base directories created, you need to create a file called `theme.php` inside your theme's default directory. Remember, this file will contain all the details about your theme for the system. The following is an example of what a `theme.php` file contains:

```
<?php defined('BASEPATH') OR exit('No direct script
  access allowed');

class Theme_Happy extends Theme
{
  public $name = ' Happy ';
  public $author = 'Zac Vineyard';
  public $author_website = 'http://zacvineyard.com';
  public $website = 'http:// zacvineyard.com/themes/happy';
  public $description = 'This theme is not sad.';
  public $version = '1.0';
}
```

The `theme.php` file is just a PHP class that defines some values for your theme, including the name, version, and options (an optional feature). Once you've added these properties to your theme's base class, your theme should be recognized by the system and available for use inside the control panel, as shown in the following screenshot:



Themes

Default Theme	Theme	Description	Author	Version	
○	Base	An HTML5 base template filled with goodies to get you up and running quickly.	Scott Parry	1.0	Options Preview Delete
○	Conjuction	2-column, fixed; horizontal nav.	Free CSS Templates	1.0	Preview Delete
○	PyroCMS Theme	Default PyroCMS v1.0 Theme - 2 Column, Fixed width, Horizontal navigation, CSS3 styling.	iKreativ	1.0	Options Preview Delete
○	Foo	The antithesis theme to your Bar theme.	Zac Vineyard	1.0	Preview Delete
○	Happy	This theme is not sad.	Zac Vineyard	1.0	Preview Delete
○	Minimal	Minimal template.	Yorick Peterse	1.0	Preview Delete

At this point, you'll need to create at least one layout file for your theme. Remember that creating multiple layout files may be necessary for your website, depending on the number of different layouts you'll need.

Creating a layout file

Depending on where you're creating your theme, all layouts files for a PyroCMS theme exist in one of the following two locations:

`addons/[site-name]/themes/[my-theme-name]/views/layouts/` (for themes available to all websites)

OR:

`addons/shared_addons/themes/[my-theme-name]/views/layouts/` (for themes available to all websites)

The first layout file you'll need to create is named `default.html`. So, create this file in one of the preceding locations. This default file, which is required for your theme to function properly, will contain some HTML markup that is mixed with PyroCMS tags. These tags, as we've discussed in earlier chapters, are code (usually HTML) that's returned from plugins attached to the system. The following code is the example I provided earlier in the book for your `default.html` file:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ template:title }}</title>
    {{ template:metadata }}
  </head>
  <body>
    <h1>{{ template:title }}</h1>
    {{ template:body }}
  </body>
</html>
```

This is, of course, ultra-simplistic, and will need various amounts of expansion based on your needs, especially when it comes to including tags. A more detailed default theme file will look more like the following code, which you can add to your `default.html` as part of this last-chapter tutorial:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ template:title }}</title>
```

```
      {{ template:metadata }}
      {{ theme:favicon file="favicon.png" }}
      {{ theme:css file="style.css" }}
      {{ theme:js file="site.js" }}
    </head>
    <body>
      <div class="header">
        <div class="logo">
          {{ theme:image file="logo.jpg" alt="Your Logo" }}
        </div>
        <div class="nav">
          {{ navigation:links group="header" }}
        </div>
      </div>
      <div class="content">
        <h1>{{ template:title }}</h1>
        {{ template:body }}
      </div>
    </body>
  </html>
```

Make sure you take note, as seen in the first few lines of the preceding code example, of how tags power your ability to include layout assets, like CSS and JavaScript. In this code, our assets include a favicon, a CSS file, a JavaScript file, and a logo image. For our website, this basic layout will be the framework for all page content. Once you have your default layout file created, you can start using it in the system.

Brief review of tags

The use of tags in layout files, WYSIWYGs, and other inputs is ubiquitous in PyroCMS. It is good to have an in-depth knowledge of how tags work. They allow you to use advanced logic, with a special syntax, right inside your layouts and so on. Tag features include use of attributes, attribute stacking, tag pairs (opening and closing tags similar to XML), and tag conditionals, as seen in the following example:

```
{{ if user:logged_in }}
  <p>You are logged in.</p>
{{ else }}
  <p>You are not logged in.</p>
{{ endif }}
```

There are, of course, a number of other ways you can use tags in PyroCMS, but this covers the basics. If you want to learn more about tags and other advanced tag syntax, you can find more in the PyroCMS documentation at `http://www.pyrocms.com/documentation`.

Adding theme partials

To make out theme layout more efficient, we're going to make use of theme partials. Partials are partial layout files in PyroCMS, and they allow you to break layouts into reusable sections. These sections can then be loaded by different layout files. This keeps you from typing the same code (header, footer and so on.) into multiple layout files. In the most basic way, we can break our layout into 3 partial files, `header.html`, `default.html`, and `footer.html`. Create each of these files in your theme folder, adding the `header.html` and the `footer.html` files into the respective partials directory. You can then include those partial files into your default layout file by using a tag shown as follows:

```
{{ theme:partial name="partialname" }}
```

Now that we are making use of partials, the following is an example of what each of our layout files should include:

- Our `header.html` file, found at `addons/shared_addons/themes/[my-theme-name]/views/partials/`, should contain the following chunk of code formerly found in our default layout file:

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ template:title }}</title>
    {{ template:metadata }}
    {{ theme:favicon file="favicon.png" }}
    {{ theme:css file="style.css" }}
    {{ theme:js file="site.js" }}
  </head>
  <body>
    <div class="header">
      <div class="logo">
        {{ theme:image file="logo.jpg" alt="Your
          Happy Logo" }}
      </div>
      <div class="nav">
        {{ navigation:links group="header" }}
      </div>
    </div>
```

- Our default.html file, found at [your-theme]/views/layouts/default.html should now contain the following code:

  ```
  {{ theme:partial name="header" }}

    <div class="content">
      <h1>{{ template:title }}</h1>
      {{ template:body }}
    </div>

  {{ theme:partial name="footer" }}
  ```

- Our `footer.html` file, found at `addons/shared_addons/themes/[my-theme-name]/views/partials/`, should now contain the following code formerly found in the default layout file:

  ```
  </body>
  </html>
  ```

See how, when using partials, all the necessary files get included into the default layout file? This is a common way to introduce efficiencies into your template code so that, as a developer, you save time, which is what PyroCMS is all about.

Now that we have our basic theme put together, we can add content to our website. You can do this by using the `Pages` module in the system (in the control panel). Two things we'll add our website, beyond basic page content, are a contact form and stream data.

Adding a contact form

A contact form is a basic feature of many, many websites. This feature comes built into PyroCMS, and can be added to any page in the system simply because it is a plugin tag you add to your website. The tag that creates this form is more complicated than most tags, but it makes simple work out of building and validating a contact form. The following tag is a simple example of a tag that generates a contact form:

```
{{ contact:form
  name = "text|required"
  email = "text|required|valid_email"
  subject = "dropdown|required|hello=Howdy|
    support=Support Ticket"
  message = "textarea|required"
  attachment = "file|jpg|png|zip"
}}
```

```
    {{ name }}
    {{ email }}
    {{ subject }}
    {{ message }}
    {{ attachment }}
  {{ /contact:form }}
```

This contact form generates five input fields for a user. These inputs are `name`, `email`, `subject`, `message`, and `attachment`. Each parameter on this form tag can take a pipe-delimited set of values. For most inputs, this pipe-delimited value follows a specific format. This first value in the delimited string is, for example, the name of the input field. The second delimited value determines whether or not the form input should be required for the user. The third delimited value can either be an extension of validation rules or another set of delimited values that determine the options included in a dropdown. The subject field, for example, will be output as a dropdown of values that include `Howdy` and `Support Ticket`. These forms follow the form validation patterns used by CodeIgniter. If you are unfamiliar with how forms get validated by CodeIgniter, you can refer CodeIgniter's User Guide for form validation documentation at `http://ellislab.com/codeigniter/user-guide/libraries/form_validation.html`.

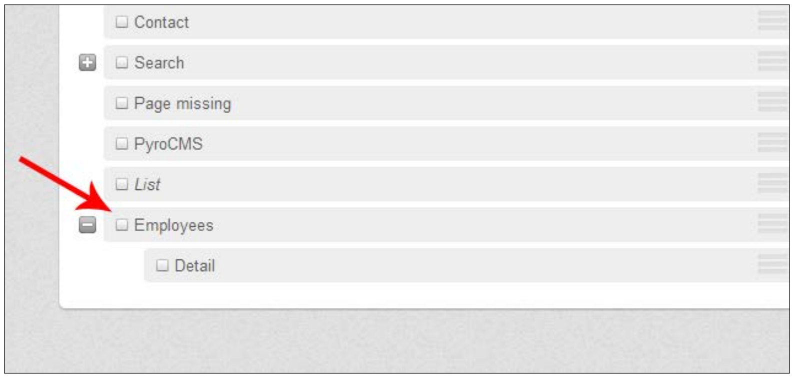
Adding stream data to your website

We covered streams in the previous chapter of this book. Building upon our efforts there, we are going to reuse the `employees` stream we built in that chapter. That stream is a directory of people's names and phone numbers, one that can be output to a website. To build that stream, we used the `Streams` module that comes built into the professional version of PyroCMS. If you didn't follow the steps put together in *Chapter 6*, *Using PyroCMS Streams*, then you'll want to create a new stream of data called `employees` in PyroCMS using the `Streams` module. We're going to take that stream, embed it into a page so that it can be seen by a user. Then, taking it a step further, we're going to build a `details` page for a single stream entry. This type of interaction, as I'm sure you know, is very common among websites.

First, let's take a look at the stream tag that's going to loop through our employee entries and output the values from our database, shown as follows:

```
{{ streams:cycle stream="employees" limit="5" }}
  {{ entries }}
    <p>{{ name }}</p>
  {{ /entries }}
{{ /streams:cycle }}
```

This code acts exactly like a `for` loop in PHP, looping over our data. You can add this streams tag to any page you create using the `Pages` module. In the case of this specific example, let's create a page called `Employees` that will list all of the employee entries in the stream:



We'll modify our stream tag a little bit by adding a link around the `name` variable, so that each record in the stream links to the single, detailed stream entry. Later, we'll embed the code for that single entry on a nested page.

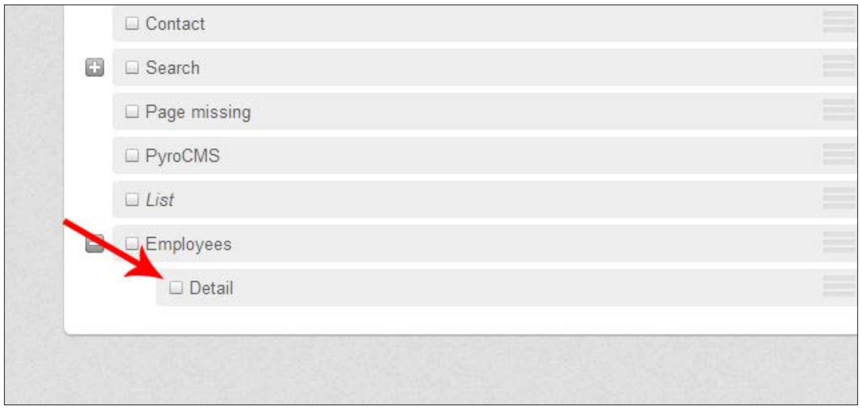
The code we'll need for our details page is very similar to the code we put on the base employees page, but it uses a URL value and the `id` parameter to limit the data returned to the user, shown as follows:

```
{{ streams:cycle stream="employees" limit="5" }}
  {{ entries }}
    <p><a href="employees/detail/{{ id }}">{{ name }}</a></p>
  {{ /entries }}
{{ /streams:cycle }}
```

As part of the details page, you can display the phone number we've entered for each entry in the employees stream, as shown in the following code sample. We'll take the following code and add it to a page that is nested underneath our base employee page in the `Pages` module:

```
{{ streams:cycle stream="employees" id="[segment_3]"
  limit="1" }}
  {{ entries }}
    <h2>{{ name }}</h2>
    <p>{{ phone }}</p>
  {{ /entries }}
{{ /streams:cycle }}
```

If however, you try to get these pages to connect without doing one more step, you'll see that is doesn't work. You'll be given a "404 page not found" error. To get all of our URLs to line up correctly, we're going to define a custom URL route in PyroCMS that will make this common/details page setup work, which will then appear as shown in the following screenshot:



Defining a custom route

Building a custom route in PyroCMS is quite easy to do. In a way, though, adding this route to the system is modifying core code that could be wiped out with future updates to the system, so take care of these routes. The file where you'll add your custom route is located within your PyroCMS project files, shown as follows:

```
system/cms/config/routes.php
```

To accommodate the employee/detail page we have created, you need to add the following route to your `routes.php` file:

```
$route['employees/detail/(:any)'] =
  'pages/view/employees/detail';
```

This route essentially makes the ID number that appears in your link to the details page, the parameter with which we can make a query against the database. Once you have this route installed, you should be able to link from an employee record in your stream to a detailed employee record page.

Summary

What I've shown you in the last few pages of this book are the essential steps to getting a website running using PyroCMS. We covered getting the system installed, creating a theme, and using streams to help power database driven content on your website. What I hope you've learned is that it doesn't take much effort to start building a smart website on PyroCMS. Because of its ability to leverage programming patterns, an easy-to-use PHP framework, and wicked fast theme development workflow, PyroCMS, as I hope you've seen, is an extremely well-outfitted system that's ready to be used on any website project.

Index