PHIL STURGEON

CATAPULT INTO PYROCMS

efendi books

Catapult into PyroCMS

PyroCMS is one of the most popular and widely used CMSs out there. Learn how to use it.

Phil Sturgeon

This book is for sale at
http://leanpub.com/catapultintopyrocms

This version was published on 2014-02-19

Efendi Books publish smart, succinct eBooks for web developers, designers and entrepreneurs.

Tweet This Book!

Please help Phil Sturgeon by spreading the word about this book on Twitter!

The suggested hashtag for this book is #pyrocms.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#pyrocms

Contents

Chapter One

Core Concepts

Themes

Themes are the way that Pyro collects and organises front-end code. They're the outer container for the dynamic pages; the shell of your website. Your theme organises the layouts, partials and assets for your design into a coherent package that Pyro can understand.

Open up the **/addons** folder at the root of your Pyro install and you'll see a folder called **default**. This is the site reference; a per-site name Pyro uses to organise multiple websites with the multi-site manager. Inside your site folder (**default**) you'll see a **themes** folder.

Yep, you guessed it, this is the home of your PyroCMS themes. Themes can also be found in **addons/shared_addons/themes**–anything inside **shared_addons** will be available to every site in the multi-site manager. Inside you'll see a few folders - **base**, **minimal** and **conjunction** - that contain some sample themes that are shipped with Pyro. It can be helpful to dig through these for a quick reference when building your own theme, so keep them around for the time being.

I'd also like to draw the **base** theme specifically to your attention. This theme contains the basic structure of an HTML5 website. It can be great to use when you're starting out as it contains the essentials, all set up and ready to go. In this book, we'll build a theme from scratch so we can learn what goes on under the hood, but in the future, when creating a new Pyro theme, **base** can be a great starting point.

Themes are set up and installed in Pyro by logging in to the admin panel and heading to *Design -> Themes*. You'll see our previously observed three themes - **base**, **minimal** and **conjunction** - as well as a fourth, active theme. This is

Pyro's default theme, and its files are found in **system/cms/themes/default**; Pyro uses the same templating system under the hood as the system we're about to get to grips with.

You'll see an *Options* button next to the *PyroCMS Theme* name. Click on it. You'll be redirected to a theme settings page, where you can configure specific bits about your theme, such as a 'Show Breadcrumbs?' boolean and a layout dropdown.

What is this useful for?

It's ideal when you're writing your own base theme and you'd like to re-use your code across multiple websites; you can have a stylesheet dropdown, different types of layouts available and a series of components you can switch on and off. You can even use the settings page to create customisable themes that can be sold or given away for other developers online.

Themes contain two major divisions of presentation files: layouts and partials.

Layouts

Layouts are the wrappers for your pages. Your layout is the outer container for specific view files; templates for your pages or blog are embedded inside the layout. Layouts allow you to contain the HTML structure, common page elements like sidebars, headers and footers, and, importantly, embedding partials. They're usually used to wrap the page content with a header, footer and a sidebar, but can, of course, be anything.

You can create as many layouts as you like, and this can be *really* useful when you want to have specially branded marketing pages, for instance, or a mobile-only version of your site.

Partials

Partials are the individual parts of your layout. They're re-usable snippets of view code that you can use across different layouts. You can also use them to simply clean things up: using multiple, smaller files means your templates are easier to scan through quickly and make small adjustments.

Usually, you'll find that you'll want to separate your header and footer away from your layouts, as well as sections like sidebars. Partials can also be useful for sections of metadata or the area in your `<head>` tag where you link to your assets (CSS and JavaScript files).

A simple rule for a partial is this: if it contains HTML that is either re-used, or getting too unwieldy to handle in one file, it probably should go in a partial.

Pages

Pyro's page system is the mechanism for entering, displaying and managing both static and dynamic content. Static pages allow your clients to create new content and modify existing content in a predictable and conventional editing system. Page types allow pages to be tied to streams–giving you an utterly customisable editing form powered by a robust set of template tags.

Pages are even more useful in 2.2, because pages can now be linked to your streams. Let's say you have a site of products, where each product has its own dedicated page. You could create a project display page in the page tree, and then display that product using streams tags in the page layout, pulling the data from a separate stream. You still may want to do that, but it's the messier option; it's now possible to represent stream entries *directly in the page tree*.

You tie your streams to a page type, so when you create a page, you fill out the data for a stream entry as well as the page fields. Then, you can simply use the fields from the page/entry in your layout. No pulling data from the URL or messing around with stream tags.

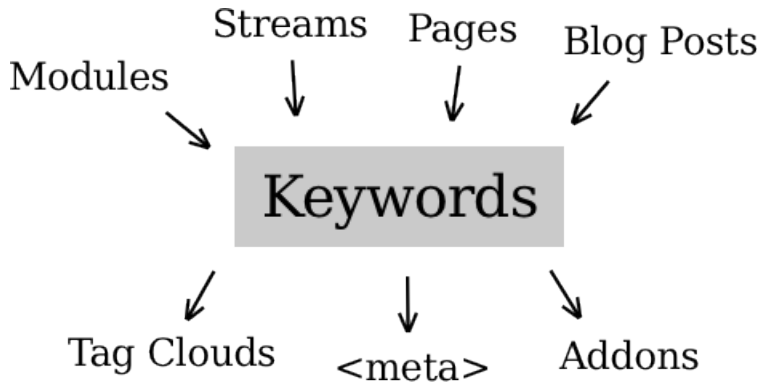
Plus, your data is still a stream, so you can display it in a cycle loop elsewhere, create entry forms for it, do anything you can do normally with streams. You get the powerful content management of streams - which we'll find out more about soon - baked straight into the page system.

We can customise the way that pages are outputted through the page type's layout. A page's layout is the way of customising template at an individual level. They allow you to

structure a specific page in the master layout in a specific way–overriding any default layout that may be set up.

Keywords

PyroCMS also contains a centralised list of keywords that enable the categorisation and labelling of content. Keywords are supported across the entire system, from generating tag clouds on the blog through to setting `<meta>` keywords for static content.



What's in a keyword?

A keywords field type exists that allows you to add keywords to any stream entries. Keywords are perfect for adding tags to posts, categories to shopping cart products or generating a tag cloud based on custom data entered by your clients. You can access the central list by heading to *Data -> Keywords* in the control panel.

Streams

The idea of streams may be familiar - they're known as channels or sections in other systems - and they're one of the most powerful elements inside PyroCMS. A stream is essentially a series of entries; multiple items of content. This differs from a blog insofar as streams are totally customisable. You can cherry-pick different fields of different types to make sure Pyro suits your website's content, not the other way round.

An example: your client runs a series of events and they'd like to display them on their website. They've got a name and a description, but they've also got an attached date, a number of available tickets and a venue. This custom data is too complex to put into the blog module, so instead, we can create a stream and set up exactly the right fields for this type of content. This is a typical example of a need to display custom content in a dynamic way, and this is precisely what streams are used for.

Streams aren't only used by your website: they allow you to add custom fields to page types, user profiles, and the blog. They're also available to developers via the Streams API, so third-party addons you use with PyroCMS may have the ability to add custom fields.

Streams are customized by assigning fields to them, and each fields has a different field type. There are a bunch of core field types to choose from: WYSIWYG editors, date pickers, numbers, entry relationships and keywords. There are also a number of other addon field types available.

Once you've added some fields to a stream, you can then add and modify entries within that stream, much like you

do in other content entry areas in Pyro. Like we mentioned a moment ago, pages can be tied to streams through page types.

Here's another good rule: if you have custom content directly related to a page, use a stream and a page type. If you've got custom content not directly linked to a page and its URL, use a stream and a template.

When we get to writing some code - soon, I promise! - we'll take a look at how to use streams in your templates.

Addons

Addons are the way that custom developer code integrates into the core of PyroCMS. They lie at the centre of Pyro and allow you, the developer, to modify the CMS and extend it to do whatever you wish. With addons you can extend Pyro's support for third-party services like Twitter and Facebook, create complex booking and invoicing systems, hook into events to effect the behaviour of internal modules and – well, pretty much anything you (or your clients) can dream of.

While writing addons is simple, there are a few different types of addon components to learn about. Even if you don't plan on writing much custom code, it's a good idea to acquaint yourself with the different types and get a feel for how the addon system works.

Plugins

Plugins are used in your templates to embed simple content. They can represent files, assets, streams, entries, whatever you like. They use a simplistic tag syntax much like Smarty or

Mustache, with the simplicity of single tags and the ability to display more complex, involved data with tag pairs and conditionals.

Plugins can be used on their own or complimenting a larger module. They're just a basic PHP class - Pyro's templating system takes all the tag parsing work out of your hands - so are very easy to build and extend.

Plugins sit in a **plugin.php** file at the root of the addon. They're comprised of a class called `Plugin_Something` (where 'Something' is replaced with the name of your addon) and they extend from `Plugin`. Like this:

```
1  class Plugin_Math extends Plugin
2  {
3      public function random_number()
4      {
5          return round(rand(0,100));
6      }
7  }
```

The `random_number()` method corresponds to the function called in the template tag, and whatever's returned from that function is displayed in the template:

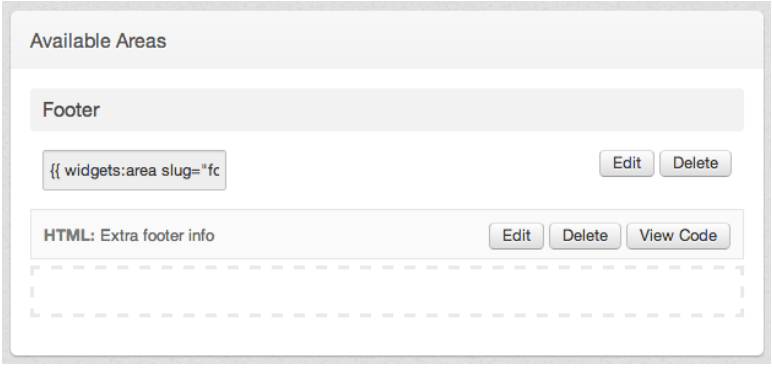
```
1  <p>Your random number is: {{ math:random_number }}\
2  </p>
```

Plugins can do much more than this, of course, but we'll look more at the way they work later when we write our first proper addon.

Widgets

I like to view widgets as "intelligent partials". They accept a few options and give you back some pre-defined HTML output. The reason why they're so clever is that you can define widget areas for your widgets to sit in. Your client can then easily manage intelligent, dynamic content without having to call you for help.

Let's illustrate this with an example: the "Twitter Feed" widget built into the core of Pyro is configured with a Twitter username, is placed in a widget area, then displays a list of tweets in your template:



Widgets sit in areas, like the one above

Widget areas aren't necessary - you can call a widget directly with a template tag - but they're very useful, because they give your client the ability to move things around to their specifications with a clean, simple, drag-and-drop admin interface.

You can write your own widgets, or use one of the several built in - including RSS feeds, Twitter, generic HTML and social bookmarking.

At first Widgets seem a little confusing, but once you get the hang of them you will come to the only logical conclusion: they are epic.

Modules

Modules are the most powerful, and most complex type of addon. Modules support a frontend interface as well as a full backend admin interface, which makes them perfect for more involved systems like shopping carts and download managers. The admin panels you interact with most of the time - blog and pages, for instance - will be modules.

Modules are built like mini-CodeIgniter applications and have their own controllers, models and views, which means that anybody with CodeIgniter experience will be able to dive straight into developing them. A big part of the MVC (Model View Controller) separation is that your module code is very organised and clean.

Modules aren't just comprised of the MVC stack, though. They can have their own libraries and config files too… as well as plugins and widgets. Plugins work with modules to provide a simple template interface - although modules have their own interface too - and using widgets lets your module piggyback on the Pyro drag-and-drop widget system for your clients to display and rearrange dynamic content in their templates.

The core operates through modules (check out **system/cms/modules** to see more) and any custom modules are placed in your addon folder (**addons/default/modules** in our case). You'll need to create a couple of module files in order to get a module working properly, but we'll come onto that later.