

Plugins by Example: the eBook plugin

Copyright © 2006 Creole West Productions

Contents

- | | | |
|----------|---------------------------------|----------|
| 1 | First Steps | 5 |
| 2 | Actually doing something | 7 |

Introduction

The very first thing that you have to decide when you're writing a plugin for puma is: how is this plugin going to work? What does it do, and how should it do it?

For this example, I will describe a plugin for ebooks.

ebook is going to use a different method of storage than reservations or forum. There are several reasons for this: 1) ebook's data is (potentially) much larger than the data contained in forum or reservations, and the data's structure will be much simpler, too (plain html). 2) There is already an ebook that has been created in the manner we will use here, and conversion is a pain when there are 200+ pages.

Specification

The ebooks plugin will be as follows:

- A page, with the title of the ebook, is created, and a page of type 'ebook' will be created underneath it.
 - All pages of the ebook will be moved underneath this plugin page, and they will be hierarchical (ie, the main sections will be immediately underneath 'ebook', then the chapters, then the pages themselves). The order of the pages will be determined by their 'position' value.
 - 'Turngrippies' (links to the next page) will be automatically generated for each page, and these will include links to the beginning of the current and next major sections.
-

Chapter 1

First Steps

The first thing that we have to do is create a file, called ‘ebook.inc’, in the /path/to/puma/plugins directory of your test installation. To start, let’s put in:

```
----- ebook.inc
<?php

function plugin_ebook($args, &$page, &$user) {
    $test = "Ebooks1";
    print "Ebooks rock!";
    return $test;
}

?>
-----
```

We need this dummy function to avoid an error (Puma automatically attempts to call functions of ‘plugin_pluginname’ when a page with the page_type ‘pluginname’ exists below a page), and to ensure that the plugin is being loaded and run properly.

Now, let’s create an ‘ebook’ page. Create a regular page, then, using MySQL, change the page’s ‘page_type’ to ‘ebook’. Load up the parent page, and you should see ‘Ebooks rock!’ at the top of the page.

Notice, however, that the value of ‘test’ doesn’t appear anywhere on the page! To fix that, we need to add a section for ‘ebooks’ to our template. You’ll have to modify at least the default template (/path/to/puma/contrib/smarty/templates/page.view.*) and your site-specific template IF you have any that override page.view.

Add the following lines to you page.view.html file, wherever you want it to appear:

```
----- page.view.html
...
{if $ebook}
<hr />
<h3>eBook</h3>
<div id="ebook">
{$ebook}
</div>
{/if}
```

...

Reload the page in your browser... there it is!

Chapter 2

Actually doing something

Next, we'd like the plugin to display something that we have stored in the database. First we need something to display. Create a page, and move it (using MySQL) underneath the ebook page.

To display the contents of the new page, we'll need to retrieve it from the database.

Since we're passed the ebook page itself, we can get all its children by calling `getAllChildren`, like so:

```
$children = $page->getAllChildren($user->permission);
```

This returns an array of pages containing all of the ebook page's children, all those pages' children, and so on, taking into account the user's permission level, meaning that if the user wouldn't normally be allowed to view a page, it isn't returned in the array!

Note There are several useful functions for each of Puma's objects, and you can examine them in `/path-/to/puma/model/`. You have access to all of Puma's internal functions from plugins.

Next, we find each page's previous page and next page, if it has them, using `find_nextprev()`. You can examine `ebook.inc` for the actual implementation; for now, just know that it returns an array like the one it receives, but with each page having a 'prev' and 'next' variable.

Then, we traverse the tree to find the first leaf of whichever page was requested. We retrieve that page (and the most recent version), and climb back up the tree to get the page's ancestors to display for section information. All of these things are stored in our 'content' variable.

Finally, we check to see if there is a forum associated with our 'leaf' page. If there is, we render it and add it to 'content'; if not, then we're done. We return 'content' to the page/view controller.

The page/view controller then sticks it into the Smarty variable 'ebook', and it is rendered on our page.
