Controllers

Overview

All plugins will have a primary class extending HubzeroPluginPlugin that contains the logic and events to be triggered.

Structure

Here we have a typical plugin class:

```
<?php
// No direct access
defined( '_HZEXEC_' ) or die();

/**
 * Example system plugin
 */
class plgSystemTest extends HubzeroPluginPlugin
{
 /**
  * Affects constructor behavior.
  * If true, language files will be loaded automatically.
  *
  * @var  boolean
  */
 protected $_autoloadLanguage = false;

 /**
  * Constructor
  *
  * @param  object $subject The object to observe
  * @param  array   $config  An array that holds the plugin configurat
ion
  * @return  void
  */
 public function __construct(&$subject, $config)
 {
  parent::__construct($subject, $config);

  // Do some extra initialization in this constructor if required
 }

 /**
```

```
  * Do something onAfterInitialise
  *
  * @return  void
  */
 public function onAfterInitialise()
 {
  // Perform some action
 }
}
```

Let's look at this file in detail. Please note that the usual Docblock (the comment block you normally see at the top of most PHP files) has been omitted for clarity.

The file starts with the normal check for defined('_HZEXEC_') which ensures that the file will fail to execute if access directly via the URL. This is a very important security feature and the line must be placed before any other executable PHP in the file (it's fine to go after all the initial comment though).

All plugins must extend or be descendants of HubzeroPluginPlugin. The naming convention of this class is very important. The formula for this name is:

plg + Proper case name of the plugin directory + Proper case name of the plugin file without the extension.

Proper case simply means that we capitalise the first letter of the name. When we join them altogether it's then referred to as "Camel Case". The case is not that important as PHP classes are not case-sensitive but it's the convention Joomla! uses and generally makes the code a little more readable.

For our test system plugin, the formula gives us a class name of:

plg + **S**ystem + **T**est = plgSystemTest

Let's move on to the methods in the class.

The first method, which is called the constructor, is completely optional. This is used only when some work is needed performed when the plugin is actually loaded. This happens with a call to the helper method Plugin::import(*<plugin_type>*). This means that even if the plugin is never triggered, for whatever reason, there is still an opportunity to execute code if needed in the constructor.

The remaining methods will take on the name of "events" that are trigger throughout the execution of the Joomla! code. In the example, we know there is an event called

onAfterInitialise which is the first event called after the application sets itself up for work.

The naming rule here is simple: the name of the method must be the same as the event on which you want it triggered. The framework will auto-register all the methods in the class for you.

System Events

One thing to note about system plugins is that they are not limited to handling just system events. Because the system plugins are always loaded on each run of the CMS, you can include any triggered event in a system plugin.

The events triggered are:

Antispam

- onAntispamDetector
- onAntispamTrain

Authentication

- onAuthenticate

Content

- onContentPrepare
- onAfterDisplayTitle
- onContentBeforeDisplay
- onContentBeforeSave
- onContentAfterSave
- onContentBeforeDelete

Cron

- onCronEvents

Editors

- onInit
- onGetContent
- onSetContent
- onSave
- onDisplay
- onGetInsertMethod

Editors XTD (Extended)

- onDisplay

Search

- onSearch
- onSearchAreas

System

- onAfterInitialise
- onAfterRoute
- onAfterDispatch
- onAfterRender

User

- onLoginUser
- onLoginFailure
- onLogoutUser
- onLogoutFailure
- onBeforeStoreUser
- onAfterStoreUser
- onBeforeDeleteUser
- onAfterDeleteUser