

Controllers

Overview

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

Directory & Files

Plugin files are stored in a sub-directory of the /plugins directory. The sub-directory represents what type the plugin belongs to. This allows for plugins of the same name but for different types. For example, one could have a plugin named example for both the /system and /search types.

Note: plugins will always be within a type sub-directory and will never be found in the top-level /plugins directory.

```
/hubzero
  /plugins
    /{PluginType}
      {PluginName}.php
      {PluginName}.xml
```

There is no restriction on the file name for the plugin (although it is recommended to stick with alpha-numeric characters and underscores only), but once you decide on the file name, it will set the naming convention for other parts of the plugin.

Structure

Here we have a typical plugin class:

```
<?php
// no direct access
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.plugin.plugin' );

/**
 * Example system plugin
 */
class plgSystemTest extends JPlugin
```

CONTROLLERS

```
{
/**
 * Constructor
 *
 * For php4 compatibility we must not use the __constructor as a constructor for plugins
 * because func_get_args ( void ) returns a copy of all passed arguments NOT references.
 * This causes problems with cross-referencing necessary for the observer design pattern.
 *
 * @access protected
 * @param object $subject The object to observe
 * @param array $config An array that holds the plugin configuration
 * @since 1.0
 */
function plgSystemTest( &$subject, $config )
{
    parent::__construct( $subject, $config );

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

/**
 * Do something onAfterInitialise
 */
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('_JEXEC')` which ensures that the file will fail to execute if accessed 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). The importance of having this check your PHP files cannot be over-emphasised.

Next we use the `jimport` function to load the library file with the definition of the `JPlugin` class.

CONTROLLERS

You will notice that a plugin is simply a class derived from JPlugin (this differs from previous versions of Joomla!). 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. You only require this if you want to do some work when the plugin is actually loaded by Joomla!. This happens with a call to the helper method JPluginHelper::importPlugin(*<plugin_type>*). This means that you even if the plugin is never triggered, for whatever reason, you still have an opportunity to execute code if you need to in the constructor.

In PHP 4 the name of the constructor method is the same as the name of the class. If you were designing only for PHP 5 you could replace this with the name of `__constructor` instead.

The remaining methods will take on the name of "events" that are triggered 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 Joomla! application sets itself up for work. For more information on when some events are triggered, see the [API Execution Order](#) page on the [Documentation Wiki](#).

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 Joomla! Framework will auto-register all the methods in the class for you.

That's the basics of the plugin PHP file. Its location, name and methods will depend on what you want to use the plugin for.

Joomla 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 Joomla! PHP, you can include any triggered event in a system plugin.

CONTROLLERS

The events triggered in Joomla! are:

Authentication

- onAuthenticate

Content

- onPrepareContent
- onAfterDisplayTitle
- onBeforeDisplayContent
- onBeforeContentSave (new in 1.5.4)
- onAfterContentSave (new in 1.5.4)

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

CONTROLLERS

- onAfterStoreUser
- onBeforeDeleteUser
- onAfterDeleteUser

XML-RPC

- onGetWebServices

For more detailed information on how to create specific plugins, visit the [Plugins Category](#) on the Joomla! Documentation Wiki.

Component Events

The following are events that are triggered from within their respective components:

Groups

- onGroupAreas
- onGroup
- onGroupNew
- onGroupDeleteCount
- onGroupDelete

Members

- onMembersAreas
- onMember

Tools

- onBeforeSessionInvoke
- onAfterSessionInvoke
- onBeforeSessionStart
- onAfterSessionStart
- onBeforeSessionStop
- onAfterSessionStop

Resources

- onResourcesAreas
- onResources

Support

- getReportedItem

CONTROLLERS

- deleteReportedItem

Tags

- onTagAreas
- onTagView

Usage

- onUsageAreas
- onUsageDisplay

What's New

- onWhatsnewAreas
- onWhatsnew

XMessage

- onTakeAction
- onSendMessage
- onMessageMethods
- onMessage

XSearch

- onXSearchAreas
- onXSearch