Extending Mura CMS with Plug-Ins - Part II - Hello World

Posted At : August 4, 2009 5:18 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, Mura CMS

In my previous post about Extending Mura CMS (an open source ColdFusion CMS) with plugins, we looked at what plugins are and how one creates and installs a plugin. In this post we'll look at a very simple plugin, and then add new features to the plugin in future posts.

There are already a couple of sample plug-ins that can be found on the Mura site. This example is loosely based on the Hello World example found there. This simple plugin won't do anything useful (yet), it will simply display the message "Hello World" on a page. So let's get started.

You may recall from the previous post that a plugin is simply a folder of files that has been zipped up, so to start our plugin we'll create a new folder, called "HelloWorld1". Inside this folder we'll create three new folders:

- plugin
- displayObjects
- eventHandlers

The *plugin* folder will contain files that Mura needs to install your plugin, the *displayObjects* folder will contain files that generate output that your plugin provides and the *eventHandlers* folder will contain files that describe different actions that your plugin will perform when certain events fire. As mentioned in the previous article, this structure (the displayObjects and eventHandlers folders) is not mandatory, but I think it provides a good illustration of the fact that, at a basic level, a plugin is able to do two things:

1. Display output
2. Perform actions

Many plugins will be designed to display output. For example, you may want to include a summary of what is in a user's shopping cart on each page, or provide a news ticker. The plugin architecture will allow you to create a display object in your plugin, and then use Mura's Site Manager to define where that content should appear in your CMS rendered pages.

As your plugins get more complex, you will also want them to perform actions to respond to certain events. There is a long list of events that your plugin can respond to, some examples are:

- *onApplicationLoad* - You would use this if your plugin needs to initialize itself when Mura is loaded or reloaded.
- *onGlobalRequestStart* - You would use this if your plugin needs to perform some action (e.g., setting a variable) whenever a request is started.
- *onRenderStart* - You would use this if your plugin needs to perform some action just before the content is rendered by Mura. You can use this to change the way that Mura renders your page, while having access to all of the content that is about to be rendered.

There are a lot more events available, and I imagine that new ones will being added as identified. The bottom line is that there are a whole lot of "plugin points" that you as a developer can hook into, which allow you to control and change the way that Mura behaves.

In our simple HelloWorld plugin we don't need to respond to any events, so we won't be putting anything into the eventHandlers folder. All we need is a single display object, which will display our message to the world. So let's create a file in the displayObjects folder called dspHelloWorld.cfm. That file will contain the following super complicated code:

```
<h1>Hello World!</h1>
```

Next we have to let Mura know that we have a display object for it. We do that by defining a display object in our config.xml file, so let's create that now. We'll create a file called config.xml in the plugin folder, which will contain the following code:

```
<plugin>

 <name>Hello World Plugin Step 1</name>

 <package>HelloWorldPluginStep1</package>

 <version>1.0</version>

 <provider>SilverWare Consulting</provider>

 <providerURL>http://www.silverwareconsulting.com</providerURL>

 <category>Sample Application</category>

 <settings>

 </settings>

 <displayobjects location="global">

  <displayobject name="Hello World Message"

   displayobjectfile="displayObjects/dspHelloWorld.cfm"/>

 </displayobjects>

</plugin>
```

The first set of elements in the xml file (everything from name to category) define the settings for your plugin. Let's take a moment to walk through them:

- *name* is the name of your plugin. This is how it will appear in the list of installed plugins that you see via Site Settings in the Mura Administrator.
- *package* is used to name the directory where the plugin gets installed by Mura. Mura will create a new directory for each plugin that you install, using the package name and appending a number to it. This makes it easier to find your plugin's directory after it has been installed. It also means, however, that you cannot count on the plugin being installed to a specific location (because of the numeric portion). If your plugin needs to know where it is installed there are a number of ways of doing that, which I plan to cover in a future post.
- *version* is the version number of your plugin.
- *provider* is you, silly. This will appear both in the listing of installed plugins and on the Plugin Setting page in the Mura Administrator.
- *providerURL* is a URL that points to your web site. It too will appear both in the listing of installed plugins and on the Plugin Setting page in the Mura Administrator.
- *category* can be used to describe the type of plugin that you have created. From what I can tell this is only used for display purposes (again both in the listing of installed plugins and on the Plugin Setting page) and does not affect how Mura interacts with your plugin. I suppose it would make managing plugins easier if you had a large number installed.

The next section consists of the *settings* element, which would include any user-customizable settings that your plugin would provide. As we don't have any of those we'll leave that element empty for now.

The final section in this file is the *displayobjects* element, which is where you tell Mura what display objects your plugin will make available. In our example, as all we're doing is displaying a message, we have one display object, so there is one *displayobject* element inside the *displayobjects* collection. We give our display object a name, in this case "Hello World Message", and we tell Mura where in our plugin structure it can find the code that generates the output for the display object. In this case the code is in displayObjects/dspHelloWorld.cfm, which is the file that we created above.

I mentioned in the previous article that, in addition to the config.xml file we need two other files in our plugin folder; config.cfm and plugin.cfc. After speaking with Matt Levineabout the purpose of and need for these files, I must now correct myself and state that neither of those files is actually required. Here's a brief description of what purpose they serve:

config.cfm

This is an optional file that one may choose to include (via a cfinclude tag) into one's plugin if one is creating a plugin with an administrative interface. In this Hello World sample plugin there is no administrative interface, so this file is not needed. Later on in the series, when we look at creating a plugin with an administrative interface we'll look more closely at this file.

plugin.cfc

This file defines actions that are to be performed when the plugin itself is altered via the Mura Administrator. For example, if you need to perform an action, such as copying files to a specific location, or running a query to update a database table, when your plugin is installed, you'd define those actions in the *install* function of this component. Actions that should be performed whenever your plugin is updated would reside in the *update* function, and actions that should be performed when your plugin is deleted would reside in the *delete* function. Because our simple plugin does not require any of those actions to be performed we can leave this component out of our plugin package.

Note, however, that the requirement for the plugin.cfc file was dropped in a recent version of Mura (version 5.1.542 to be exact). If you are running an older version, you will still need to include a plugin.cfc file in your plugin folder, and it will need to contain the following minimum of code in it:

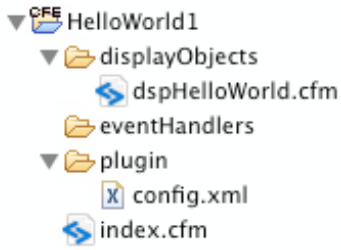
```
<cfcomponent output="false">

<cfset variables.config=""/>

<cffunction name="init" returntype="any" access="public" output="false">
 <cfargument name="config"  type="any" default="">
 <cfset variables.config = arguments.config>
</cffunction>

<cffunction name="install" returntype="void" access="public" output="false">
</cffunction>

<cffunction name="update" returntype="void" access="public" output="false">
</cffunction>

<cffunction name="delete" returntype="void" access="public" output="false">
</cffunction>

</cfcomponent>
```

The final file that we need in our plugin structure is an index.cfm file which will sit in the root of our folder structure. We'll create one with the following code:

```
This is the HelloWorld Plugin. It doesn't do much. Thanks for taking a peek.
```

This file, the index.cfm in the root, is what will be executed when an administrator clicks on the plugin's name from the list of installed plugins. You can put anything you want in here, such as a message, credits, instructions for the plugin, etc. You could even make it the entry point for an entire application (for example a Model-Glue application), if you want that application to be available to the administrator.

So that completes our simple plugin. At this point you should have a directory structure resembling the following:



To create a plugin that you can deploy into Mura, just zip up the entire contents of the HelloWorld1 folder. You can then follow the instructions in the previous post to deploy the plugin into an existing Mura installation. Once that's done you can use the display object in any of your pages. How, you ask? Read on.

Go to the Site Manager in the Mura Administrator and choose a page to edit. When you are editing the page, choose the *Content Objects* tab. You should see a select box that reads "Select Object Type". Choose *Plugins*, and the select box should populate with display objects from any plugin that is currently installed in your site. You should see a heading (an optgroup, in fact) in the select box for Hello World Plugin Step 1, and below that you should see an item for Hello World Message. Choose that and use one of the buttons to place it into one of your content areas and then click the Publish button. If you now view the page that you just edited you should see the Hello World message in the content area that you specified.

And that's pretty much all there is to it. Of course, this sample plugin doesn't do anything interesting yet, but we'll build upon it in future posts, exploring some of the ways that we can make plugins that are a lot more useful.

If you want to try out this plugin without having to create folders and type code, a complete copy of the plugin zip file is attached to this post.

I just want to close by saying that I think that Mura really is a very well designed product. I've been building a plugin that will be replacing an entire e-commerce site and I've had a number of pretty complicated requirements to fulfill. I have yet to come across a use case that I cannot address with Mura and its plugin architecture.