Extending Mura CMS with Plugins - Part IV - Using Extended Attributes

Posted At : January 6, 2010 12:12 PM | Posted By : Bob Silverberg
Related Categories: ColdFusion, Mura CMS

In the previous post in this series about Extending Mura CMS (an open source ColdFusion CMS) with Plugins, we looked at using plugin settings to make our plugin configurable by a Mura administrator. In that simple example, our plugin settings were used directly in our ouput. Although plugin settings can be useful in a number of scenarios, using them in that manner has a limitation, which is that you'll get the exact same output on each and every page that uses the plugin, because the value of the setting is set via the Mura Admin and hence the plugin always uses that single value.

In this post we'll look at a different technique for making a plugin configurable, and this time we'll be able to configure the plugin on a page-by-page basis, instead of having one global configuration setting. We'll do this by assigning a value to an extended attribute of a page, and then direct our plugin to use that value. The first step, therefore, is to enable an extended attribute for a page, so we'll look at that first.

When editing a page in the Mura admin, there's a tab labelled *Extended Attributes*. When you click on that tab for a standard page, you'll see a message stating: *There are currently no extended attributes available*. In order to provide extended attributes for a page you need to use the *Class Extension Manager* to create a *Page Sub Type* that extends the built-in Page class, and then add attributes to that new Sub Type. Here is a step-by-step guide:

1. Access the *Class Extension Manager* in order to create a *Page Sub Type*:
    1. In the upper right-hand corner of the Mura Admin, point to *Site Settings* and choose *Edit Current Site*.
    2. Click on the *Class Extension Manager* link immediately under the *Site Settings* header.
2. Add a *Page Sub Type* which can then be assigned to a Page:
    1. Click on the *Add Class Extension* link.
    2. Choose a *Base Type*. In this example we are choosing *Page* because we want to add extended attributes to a Page.
    3. Enter a name for the *Sub Type*. In this example we'll enter "HelloPage" as this will be a type of page that can say hello.
    4. Click the *Add* button.
3. Add an *Attribute*:
    1. Click on the *[Edit]* link beside the word *Default*. This will allow us to add an attribute to the Default attribute set.
    2. Click the *[Add New Attribute]* link. This will bring up a form which allow us to define our new attribute.
    3. Fill in the form. You must specify at least a *Name* for the attribute. Let's name this attribute "HelloName". You can leave the rest of the fields blank. The other fields on the form are used to define how the attribute will be presented to an administrator when editing a page. These fields are nearly identical to those available when defining settings for a plugin, so you can find more information on them in theprevious post in the series that discussed plugin settings.
    4. Click the *Add* button. This will being you back to the *Manage Attributes Set* screen, and you should see your new attribute at the bottom with links to *[Edit]* and *[Delete]* beside it.

We now have a *HelloPage* Page Sub Type that can be assigned to a page, and it will allow us to specify a value for the *HelloName* extended attribute. To do that, create a new page or edit an existing page. On the *Edit Content* screen, you'll see there's a select box labelled *Type*. When you click the select box you should now see an option labelled *Page / HelloPage*. Choose that option, which tells Mura that you want this page to use the HelloPage Sub Type that we just created. Now click the*Extended Attributes* tab and you should see a text box labelled *HelloName*. Enter your own name into the text box and then publish the page. That page will now have your name assigned to the extended attribute *HelloName*. Next we have to tell our plugin to make use of that extended attribute.

As we did in the last article, we're going to add another display object to our plugin, which will make use of the extended attribute. Let's do that by editing the */plugin/config.xml* file. Here's the new version:

```
<plugin>
 <name>Hello World Plugin Step 3</name>
 <package>HelloWorldPluginStep3</package>
 <version>1.0</version>
 <provider>SilverWare Consulting</provider>
 <providerURL>http://www.silverwareconsulting.com</providerURL>
 <category>Sample Application</category>
 <settings>
  <setting>
   <name>toWhom</name>
   <label>We are speaking to</label>
   <hint>The name of the person to whom we're speaking</hint>
   <type>text</type>
   <required>true</required>
   <validation></validation>
   <regex></regex>
   <message></message>
   <defaultvalue></defaultvalue>
   <optionlist></optionlist>
   <optionlabellist></optionlabellist>
  </setting>
  <setting>
   <name>sayWhat</name>
   <label>We are going to say</label>
   <hint>This is what we are going to say to them</hint>
   <type>select</type>
   <required>true</required>
   <validation></validation>
   <regex></regex>
   <message></message>
   <defaultvalue>Hello</defaultvalue>
   <optionlist>Hello^Goodbye</optionlist>
   <optionlabellist></optionlabellist>
```

```
  </setting>
 </settings>
 <displayobjects location="global">
  <displayobject name="Hello World Message"
   displayobjectfile="displayObjects/dspHelloWorld.cfm"/>
  <displayobject name="Greeting Message"
   displayobjectfile="displayObjects/dspGreeting.cfm"/>
  <displayobject name="Hello Page Message"
   displayobjectfile="displayObjects/dspHelloPage.cfm"/>
 </displayobjects>
</plugin>
```

Once again we changed the *name* and *package* elements to reflect the fact that this is a new plugin. Then we added a new *displayobject* element, *Hello Page Message*, which will be used to display a message based on the extended attribute assigned to a page.

Next we create that new display object. Here's what dspHelloPage.cfm will look like:

```
<cfoutput>
<h1>Hello Page</h1>
<p>Hello #Event.getContentBean().getValue("HelloName")#!</p>
</cfoutput>
```

We get the *ContentBean* for the current page using the *getContentBean()* method of the *Event* object, and then ask for the value of the *HelloName* extended attribute using the *getValue()* method. Install this plugin as you've done previously and add the *Hello Page Message* display object to a content area on the page for which you entered your name into the *HelloName* extended attribute. When you browse to that page ou should now see the output of the plugin saying hello to you. To use this same plugin display object on a different page to say hello to someone else, simply provide a different value for the *HelloName* extended attribute on that page.

As with the other articles in this series, the usefulness of this particular example is questionable, but the technique of providing information to a plugin on a page-by-page basis can be quite useful. For example, I'm using this technique in a plugin that generates catalog pages for an ecommerce site. The site administrator can specify a value in an extended attribute for a catalog page, and the plugin uses that value to ask the model for data for that particular catalog page. A display object in the plugin then formats the data and it is presented on the page.

As always, the completed plugin is available as an attachment to this post.