Records / Manager

Posted At : June 22, 2010 9:30 AM | Posted By : Steve
Related Categories: com.sebtools

This entry has been updated as **Super Easy CRUD/File Management**.

It occurred to me recently that I haven't yet blogged about one of my favorite tools in my **com.sebtools package**. It makes the most common CRUD tasks dead simple (I think more than any other CRUD tool that I have seen). Records.cfc is one of the only components in the com.sebtools package that is used by extension rather than composition. Normally, I prefer composition but I think the benefits are worth it in this case.

Rather than try to describe what it does in generic terms, let's start with an example. We can start with a completely empty folder.

First we will add the com.sebtools package either into the folder itself or into the CustomTags directory (or into a "/com/sebtools/" mapping).

Next, load DataMgr, FileMgr, CFIMAGE (which allows image functionality with CF8 or with earlier versions of CF), and Manager.cfc. You could use any dependency injection engine for this, but I am just going to do the work directly in Application.cfc for simplicity.

```
<cfcomponent>

<cffunction name="onApplicationStart">

   <cfscript>
   Application.DataMgr = CreateObject("component","com.sebtools.DataMgr").init("TestSQL");
   Application.FileMgr = CreateObject("component","com.sebtools.FileMgr").init(ExpandPath("/files/"),"/files/");
   Application.CFIMAGE = CreateObject("component","com.sebtools.cfimagecfc").init();
   Application.Manager =
CreateObject("component","com.sebtools.Manager").init(Application.DataMgr,Application.FileMgr,Application.CFIMAGE);
   </cfscript>

</cffunction>

<cffunction name="onRequestStart">

   <cfif StructKeyExists(URL,"reinit")>
      <cfset onApplicationStart()>
   </cfif>

</cffunction>

</cfcomponent>
```

This seems like a lot of code for not having accomplished anything yet, but these four lines of code give us a lot. We get DataMgr hooked up to our "TestSQL" datasource. DataMgr will determine the database type for us. We tell FileMgr to deal with uploads in the "/files/" folder (which it will create for us if it doesn't already exist). We create CFIMAGE which will give image manipulation functionality for CF8 (or earlier if we have **Rick Root**'s **image.cfc** loaded into "com.opensourcecf"). Manager.cfc is the keystone here allowing unified access to the previous pieces of functionality (and is required for using Records.cfc).

Now that we have that done, we can create a Records component (Widgets.cfc, in this case). Records.cfc is built to work with Manager.cfc, so we must define any tables that we want to use with **Manager.cfc's XML syntax**:

```
<cfcomponent extends="com.sebtools.Records" output="no">

<cffunction name="xml" access="public" returntype="string" output="no" hint="I return the XML for the tables needed for Academic Affairs to work.">

	<cfset var result = "">

	<cfsavecontent variable="result">
	<tables>
		<table name="tblWidgets" labelField="WidgetName" labelSingular="Widget" labelPlural="Widgets" folder="widgets">
			<field name="WidgetID" type="pk:integer" />
			<field name="WidgetName" Label="Widget" type="text" Length="255" required="true" />
			<field name="ordernum" type="Sorter" />
			<field name="DateCreated" Label="Date Created" type="CreationDate" />
			<field name="DateUpdated" Label="Date Updated" type="LastUpdatedDate" />
			<field
				name="WidgetImage"
				Label="Image"
				type="image"
				Folder="images"
				MaxWidth="400"
				MaxHeight="400"
			/>
			<field
				name="WidgetThumb"
				Label="Image"
				type="thumb"
				original="WidgetImage"
				Folder="thumbs"
				MaxWidth="80"
				MaxHeight="80"
			/>
		</table>
	</tables>
	</cfsavecontent>

	<cfreturn result>
</cffunction>

</cfcomponent>
```

This may look like a lot of code at first, but we are really describing everything we need to know about the table and plenty about the application functionality as well. Let's instantiate Widgets.cfc and then discuss what we have.

```
<cffunction name="onApplicationStart">

	<cfscript>
	Application.DataMgr = CreateObject("component","com.sebtools.DataMgr").init("TestSQL");
	Application.FileMgr = CreateObject("component","com.sebtools.FileMgr").init(ExpandPath("/files/"),"/files/");
	Application.CFIMAGE = CreateObject("component","com.sebtools.cfimagecfc").init();
	Application.Manager =
CreateObject("component","com.sebtools.Manager").init(Application.DataMgr,Application.FileMgr,Application.CFIMAGE);

	Application.Widgets = CreateObject("component","Widgets").init(Application.Manager);
	</cfscript>

</cffunction>
```

Make sure to **re-initialize your application**. As you can see, Widgets.cfc has an "init" method that takes Manager.cfc. More than that, however, it has several other notable methods (the names of which are mostly driven by the "labelSingular" and "labelPlural" attributes of the table element), including the following:

- getWidget(WidgetID): Returns a recordset of a single widget
- getWidgets(): Returns a recordset of multiple widgets. Any arguments passed in act as equality filters (so, getWidgets(WidgetName="Glue") would return widgets with a "WidgetName" of "Glue").
- removeWidget(WidgetID): Deleted a widget record.
- saveWidget(): Saves a widget and returns the WidgetID of the saved widget (more on this later).
- sortWidgets(Widgets): Takes a comma-delimeted list of WidgetID values and sets the values of the "ordernum" field accordingly (because it is type="Sorter").

OK. Let's do a little CRUD work so we can see what we have got. We'll create a very simple form to add a widget:

```
<form action="action.cfm" enctype="multipart/form-data" method="post">
   <div>
      <label for="WidgetName">Widget</label><br>
      <input type="text" name="WidgetName">
   </div>
   <div>
      <label for="WidgetImage">Image</label><br>
      <input type="file" name="WidgetImage">
   </div>
   <div>
      <input type="submit" value="Submit">
   </div>
</form>
```

Here is the action page for the form:

```
<cfif StructKeyExists(Form,"WidgetImage")>
   <cfset folder = Application.Widgets.getFolder("WidgetImage")>
   <cfset dir = Application.FileMgr.getDirectory(folder)>
   <cffile action="UPLOAD" filefield="WidgetImage" destination="#dir#" nameconflict="overwrite">
   <cfset Form["WidgetImage"] = CFFILE.ServerFile>
</cfif>
<cfset Application.Widgets.saveWidget(argumentCollection=Form)>
<cflocation url="index.cfm" addtoken="no">
```

Widgets.cfc will resize the image (if needed) to fit inside 400X400 and copy it to the thumbnails folder where it will resize it to fit inside a 80X80 box.

The names of the files (not including their locations) will be stored in the appropriate fields in the table, along with an increment value for the ordernum field and the current date for both the "DateCreated" and "DateUpdated" fields (and, of course, the given WidgetName).

Now we can call getWidgets() to see our data. The resulting query will have columns for all of the defined fields plus the following fields:

- WidgetImagePath: The physical location of the image.
- WidgetImageURL: The browser path to the image.
- WidgetThumbPath: The physical location of the thumbnail.
- WidgetThumbURL: The browser path to the thumbnail.

So, we could have a table like this:

```
<cfset qWidgets = Application.Widgets.getWidgets()>

<table border="1" cellpadding="3" cellspacing="0">
<tr>
   <th>Widget</th>
   <th>Image</th>
</tr>
<cfoutput query="qWidgets">
<tr>
   <td>#WidgetName#</td>
   <td><cfif Len(WidgetThumbPath) AND FileExists(WidgetThumbPath)><a href="#WidgetImageURL#"><img
src="#WidgetThumbURL#" alt="" border="0"></a><cfelse> </cfif></td>
</tr>
</cfoutput>
</table>
```

That is a fast overview of Records.cfc. If you want to find out more, the **Records.cfc documentation** is just getting started, so feel free to let me know if you have any questions (either in comments or on my **contact form**).

Records.cfc is part of the **com.sebtools package** which is open source and free for any use.