Super Easy CRUD/File Management

Posted At : October 12, 2010 10:30 AM | Posted By : Steve
Related Categories: com.sebtools

A few **months ago**, I wrote about using Records.cfc and Manager.cfc to manage records and associated files. I just released a new build of the **com.sebtools** package that contains these components and it makes a really easy process even easier. I am really excited about how easy this stuff is now.

The worst part is up front. We need to load up the components that will be used in this (and subsequent) examples. Here is a sample Application.cfc to load up the needed files:

```
<cfcomponent>

<cffunction name="onApplicationStart">

<cfscript>
//just needed once per site Application.Manager =
CreateObject("component","com.sebtools.Manager").init(datasource="TestSQL",UploadPath=ExpandPath("/files/"),UploadURL="/file

//One entry per data set Application.Widgets = CreateObject("component","Widgets").init(Application.Manager);
</cfscript>

</cffunction>

<cffunction name="onRequestStart">

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

</cffunction>

</cfcomponent>
```

The Manager.cfc component is part of com.sebtools and will need to be present with a number of other com.sebtools components (namely DataMgr.cfc, CFIMAGE.cfc, FileMgr.cfc, Records.cfc). You can load them up using **ColdSpring** or **LightWire** or however you normally load Application-scoped components. The Widgets.cfc component is just for our example.

Here is all of the code for Widgets.cfc:

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

<cffunction name="xml" access="public" returntype="void" output="yes">
<tables prefix="tbl">
   <table entity="Widget" labelLength="255" Specials="Sorter,CreationDate,LastUpdatedDate" folder="widgets">
      <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>
</cffunction>

</cfcomponent>
```

This will create a table in your database name "tblWidgets". The table name is derived from the "name" attribute of the "table" element. If no "name" attribute is provided (as in this example), then the "entity" attribute is made plural and prepended with the "prefix" attribute if one is available. The "prefix" attribute defaults to the value of the "prefix" attribute of the "tables" element.

So, this represents a lot of short cuts. In fact, many of the fields are created by similar short cuts. For example, each item in the "Specials" attribute of the "table" element represents a field. So, this syntax provides a lot of common functionality for very little work.

WidgetID	incrementing integer primary key field
WidgetName	varchar(255)
WidgetImage	varchar(180)
WidgetThumb	varchar(180)
ordernum	integer
DateCreated	date
DateUpdated	date

The "DateCreated" field is a **DataMgr Special="CreationDate"**, so it will always hold the date that the record was created. The "DateUpdated" is a DataMgr **Special="LastUpdatedDate"** so it will always hold the date the record was last added or modified.

The Application.Widgets component also gets several methods for free based on the "methodSingular" and "methodPlural" attributes of the "table" element. Those values are created (indirectly) from the "entity" attribute. Once again, a lot of stuff is provided for free.

Here are some of the methods that are automatically created in Application.Widgets:

- 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-delimited list of WidgetID values and sets the values of the "ordernum" field accordingly (because it is type="Sorter").

So, we have seen that very little code has created a database table and a handful of handy methods. Let's see how easy those methods are to use. First up, let's save a widget using a form:

```
<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:

```
<cfset Application.Widgets.saveWidget(argumentCollection=Form)>
```

```
<cflocation url="index.cfm" addtoken="no">
```

How is that for easy? The Records component is smart enough to recognize a file upload and handle it appropriately.

Whether the file field comes in as a file upload or just as the name of an already uploaded file, it will automatically be re-sized to fit the specified dimensions. The thumbnail will also automatically be created from the main image (based on the "original" attribute of that field) and be re-sized appropriately.

The files themselves will go in /files/widgets/images/ and /files/widgets/thumbs/. This location is determined by the arguments passed into the "init" of **FileMgr** and the "folder" attribute of the table and field elements.

That is it, no SQL to write, no objects to create and populate, no file uploads to manage by hand. The code executes immediately, so no need to worry about caching or execution times or sessions.

Now let's output our data:

```
<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>
```

The data returned from our "getWigets" and "getWidget" methods are ColdFusion queries, plain and simple.

The WidgetThumbPath and WidgetThumbURL variables are columns that are automatically added into the recordset. They hold the file path and URL to the image respectively. The columns are created for any field in the recordset that holds a file reference.

OK. I think we have covered quite a bit about how easy Records and Manager make CRUD and file work. The last thing I will cover for this entry is a bit about arguments that the getWidgets method can take. It can take an argument for any field name in the table. For example, if you only wanted to get widgets named "Bob", you could use the following code:

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

The method can also accept any argument for **DataMgr.getRecords()**. So, if you wanted to get records 11-20, you could use the following code:

```
<cfset qWidgets = Application.Widgets.getWidgets(offset=10,MaxRows=10)>
```

That code, by the way, would execute in the database for any database that supports it or in ColdFusion for databases that don't support it.

Records and Manager can do much more than I am showing here, but this seems like enough to whet an appetite without making for an overly long entry. I do have more entries planned covering some more complicated scenarios, but I would love suggestions on what else you would like to see as well - especially any worries that you might have about this approach.

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