Getting Data to Records.cfc

Posted At : October 26, 2010 10:30 AM | Posted By : Steve
Related Categories: com.sebtools

In our last Records/Manager blog entry, we dealt with XML syntax for **defining a data structure in Manager.cfc**. Although **Manager.cfc** is plenty useful by itself, I like to use it with **Records.cfc**.

Before we talk about how to get data into Records.cfc, we should cover how the two relate. Manager.cfc actually stores all of the data structure definitions and does all of the work. It manages the database (through **DataMgr**) and the files (through **FileMgr**) and image sizing (through CFIMAGE.cfc). So, why have Records.cfc at all?

Records.cfc is a table-specific service and provides a much nicer API to a single database table. If this were an OO system, Records.cfc would be the object. It is important to remember, however, that it is a table-specific service - not an object. It takes arguments and returns values. It doesn't represent an instance of data.

So when I say we need to get data into Records.cfc, what we really need to do is get data definitions into Manager.cfc and let Records.cfc know which table it is handling. There are several ways to do this, most of which I will cover here.

Records.cfc's "xml" method

Records.cfc has an "xml" method that serves that puts data definitions into Manager.cfc and tells Records.cfc which table to use. While you can define as many tables as you want in the "xml" method, Records.cfc will only automatically handle the first table in the XML.

Using our previous XML (sans the data), here is an example of a Records.cfc component using the "xml" method:

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

<cffunction name="xml" access="public" returntype="string" output="no">

   <cfset var result = "">

   <cfsavecontent variable="result">
   <tables prefix="hr">
      <table entity="Department" />
   </tables>
   </cfsavecontent>

   <cfreturn result>
</cffunction>

</cfcomponent>
```

The "xml" method can either return a string (as above) or simply output it. While outputting a string from a method may seem uncouth, nothing will actually be output to the buffer (unless, of course, you call the "xml" method directly). It exchange for a seemingly uncouth use of a method, you get a shorter, cleaner syntax. To wit:

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

<cffunction name="xml" access="public" output="yes">
<tables prefix="hr">
   <table entity="Department" />
```

```
</tables>
</cffunction>

</cfcomponent>
```

In either event, this is the entire code for your component (probably named "Departments.cfc" - though that is up to you).

Using variables.table

Assuming that the definition for the table has been sent to Manager.cfc outside of your Records.cfc component (or you don't want to worry with the order of the tables in your "xml" method), you could also simply set the value of "variables.table" to the name of the table that you want to manage. So, using the same table definition as before the following could be an entire Records.cfc:

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

<cfset variables.table = "hrDepartments">

</cfcomponent>
```

Once again, this could be all of the code in your component (though in all of these examples you might have custom code as well).

Use the "table" argument

You can also set the table by passing it to the "init" method of Records.cfc in the "table" argument. So, earlier examples would have been called like this:

```
<cfset Application.Departments = CreateObject("component","Departments").init(Application.Manager)>
```

You could do this instead:

```
<cfset Application.Departments =
CreateObject("component","com.sebtools.Records").init(Manager=Application.Manager,table="hrDepartments")>
```

In this case, you wouldn't even need to create a CFC file for that table.

This approach might not make sense at first. But depending on where this is being called it can be very advantageous. Remember that the resulting component will be the same regardless of which way Records.cfc is told which table to use.

The Resulting Records.cfc

No matter how the Records component is created, it will still do the same thing. For the example table, it would still have (among others), the following methods:

- getDepartment(DepartmentID): Returns a recordset of a single department
- getDepartments(): Returns a recordset of multiple departments. Any arguments passed in act as equality filters (so, getDepartments(DepartmentName="Glue") would return departments with a "DepartmentName" of "Glue").
- removeDepartment(DepartmentID): Deletes a department record.
- saveDepartment(): Saves a department and returns the DepartmentID of the saved department.

This is not an exhaustive list of the methods available. For example, the component would have several information methods (like **getFieldsArray**, **getFieldsStruct**, and **getMetaStruct**) available as well.

The point here, however, is not to detail the functionality provided by Records.cfc (which will be covered in later entries), but to note that Records.cfc provides the exact same API and built-in behaviour regardless of how the database table is made available to it (even if no CFC file is created to handle that table).

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