Basics of Manager.cfc XML

Posted At : October 19, 2010 12:45 PM | Posted By : Steve
Related Categories: com.sebtools

In my **last entry on Manager.cfc**, we covered a fairly basic example of managing widgets with images and thumbnails. I want to step back this time and start with a more simple example to allow us to really dive in and understand some of what Manager.cfc provides. This example will be expanded in subsequent entries to provide more complexity to cover some of the challenges that you might run into in the real world.

For our example, we have a client that has asked us to build an HR application. One part of that is that the company has 5 departments:

- Executive
- I.T.
- Legal
- Production
- Sales

When asked if these will ever change, our client says "I don't think so." (which, of course, means "probably"). So, let's set up the XML for this table:

```
<tables prefix="hr">
   <table entity="Department">
      <data>
         <row DepartmentName="Executive" />
         <row DepartmentName="I.T." />
         <row DepartmentName="Legal" />
         <row DepartmentName="Production" />
         <row DepartmentName="Sales" />
      </data>
   </table>
</tables>
```

This is our final code that we will pass to Manager.cfc via the "loadXml" method. Every other code example will be illustrative, but the one above is the only thing we will actually be writing ourselves.

The database table that would be created by Manager.cfc (if it doesn't already exist) would be named "hrDepartments" and would look like this (any missing columns would be added):

DepartmentID	DepartmentName
1	Executive
2	I.T.
3	Legal
4	Production
5	Sales

Data

So, the "data" portion will be creating that data when it creates the table. Otherwise, it won't do anything. So, let's remove that from our example just to make things simpler. If we didn't need any data in our table at the time it is created (i.e. - it would all be added later) then our code could look

like this:

```
<tables prefix="hr">
   <table entity="Department" />
</tables>
```

Table Attributes

makeCompName

The first thing we need is a table name. In order to understand this, we need to first cover an internal method named "makeCompName". The "makeCompName" function basically removes any invalid (for CFCs and database tables) characters from a string (including spaces) and camel-cases the string based on where spaces had been.

"name" attribute

The "name" attribute of the table determines the name of the database table. It is required if the "entity" attribute is not present. If the "entity" attribute is present, then the "name" attribute is the makeCompName() of the plural of the "entity" preprended with the "prefix" attribute of the table element (which itself defaults to the "prefix" attribute of the "tables" element - which defaults to an empty string).

So, the "name" attribute in our example is "hrDepartments". If the "entity" attribute had been "Distribution Center" then the "name" attribute would have been "hrDistributionCenter". If it had been "First-Born Child", then the "name" would be "hrFirstBornChildren".

Our *effective* XML for this:

```
<tables prefix="hr">
   <table prefix="hr" entity="Department" name="hrDepartments" />
</tables>
```

Label and Method Attributes

Assuming the "entity" attribute is provided, it is also used for setting default values for attributes "labelSingular" and "labelPlural". These can be used for any UI code that refers to the data set. As might be expected, "labelSingular" defaults to the value of the "entity" attribute, while the "labelPlural" defaults to the plural of the "labelSingular" attribute. If the "entity" attribute is not present, these attributes are required.

The table will also have a "methodSingular" and "methodPlural" attribute, which will default to the values of "labelSingular" and "labelPlural" respectively. These attribute are used by Records.cfc when creating methods and could be used for similar purpose by any other component.

Our *effective* XML for this:

```
<tables prefix="hr">
   <table entity="Department" labelSingular="Department" labelPlural="Departments" methodSingular="Department"
methodPlural="Departments" />
</tables>
```

If the "entity" attribute is provided then a default value is created for the "folder" attribute as well. The default a lowercase of makeCompName of the plural of the "entity" attribute with the "prefix" prepended as a list item. So, our example would default the folder to "hr,departments". That means

that if any file fields are in that table they would be placed inside the folder for that file, which would be "#FileMgr.UploadPath#/hr/departments/".

Our *effective* XML for this:

```
<tables prefix="hr">
   <table entity="Department" folder="hr,departments" />
</tables>
```

So, here is our *effective* XML so far:

```
<tables prefix="hr">
   <table prefix="hr" entity="Department" name="hrDepartments" labelSingular="Department" labelPlural="Departments"
methodSingular="Department" methodPlural="Departments" folder="hr,departments" />
</tables>
```

We'll leave off these extra attribute for our next examples, however.

Primary Key Field

The logic for determining the primary key field may, at first, seem somewhat more convulated than for other things, but I think it is pretty flexible.

In this example, an "entity" attribute is provided and no "pkfield" attribute is present nor is any primary key field. As such, the "pkfield" attribute is set to "DepartmentID" (using makeCompName() of entity plus "ID").

If the "pkfield" attribute is set and a primary key field exists in the table definition of a different name then Manager.cfc will throw an exception. If the table has a simple (one field) primary key but no "pkfield" attribute then the "pkfield" attribute will be set to the name of that field. If the "pkfield" attribute exists, but no primary key field exists, then a field will be created with a "type" of "pk:integer" (an incrementing integer primary key field).

In this example, that would give the following *effective* XML:

```
<tables prefix="hr">
   <table entity="Department" pkfield="DepartmentID">
      <field name="DepartmentID" type="pk:integer" />
   </table>
</tables>
```

Label Field

The label field is the field used to identify a record in the data set. For example, the "Title" field might be the label field for a page. It is determined similarly to the primary key field.

The "labelField" attribute of the "table" element is required. If the "entity" attribute is provided then it is makeCompName() of entity plus "Name". In our example, that would be "DepartmentName". If the "labelField" refers to a field that is not in the XML then it will be created as a text field with a length equal to the "labelLength" attribute (which defaults to 120) and a "label" equal to the "labelSingular" attribute of the table.

Here is our *effective* XML for the label field:

```
<tables prefix="hr">
   <table entity="Department" labelField="DepartmentName">
```

```
      <field name="DepartmentName" label="Department" type="text" Length="120" required="true" />
   </table>
</tables>
```

Special Fields

One thing our example doesn't have but is worth covering here is a "Specials" attribute. The "Specials" attribute is a comma delimited list of "Special" fields that should automatically be included in the table.

Here are the "Special" types:

- CreationDate (default name: "DateCreated", default label="Date Created"): Holds the date/time in which the record was created.
- LastUpdatedDate (default name: "DateUpdated", default label="Date Updated"): Holds the date/time in which the record was created or last updated.
- Sorter (default name: "ordernum"): An integer field for manually sorting records. If present, Manager will return records in this order if no other order is provided.
- DeletionDate (default name: "WhenDeleted"): A date field indicating when a record was deleted. Manager will not return a deleted record unless this field is in the criteria passed to getRecords.

For example, if we added Specials="CreationDate,LastUpdatedDate,Sorter,DeletionDate" then here would be our *effective* XML (including the primary key and label fields):

```
<tables prefix="hr">
   <table entity="Department" labelField="DepartmentName">
      <field name="DepartmentID" type="pk:integer" />
      <field name="DepartmentName" type="text" label="Department" Length="120" required="true" />
      <field name="DateCreated" type="CreationDate" label="Date Created" />
      <field name="DateUpdated" type="LastUpdatedDate" label="Date Updated" />
      <field name="ordernum" type="Sorter" />
      <field name="WhenDeleted" type="DeletionDate" />
   </table>
</tables>
```

I don't think this is the best explanation of the XML for Manager.cfc, but I wanted to get something down for a basis for future examples. Hopefully it will all become clearer as I work through the rest of the example. In the meantime, I would love to hear any questions.

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