Farcry and Extensibility

Posted At : November 25, 2006 4:28 PM | Posted By : Mark Kruger
Related Categories: farcry

One of the most outstanding features of Farcry is it's extensibility. First, let me say that I am still learning the full extent of Farcry's power and extensibility. When I went to install my own tool into the webtop I must confess I was prepared for an afternoon of trial and error. I had my comfort food, soothing music and serenity prayer standing by just in case. As it turns out, I did not need any of those things.

You see, virtually everything in Farcry is extensible in some way. Farcry developers have managed to create a framework with a *mirror like* structure that allows you to manipulate the core Farcry installation files *from your project folder*, without ever needing to change the core files. Why is this important? Because it allows you to upgrade and patch Farcry with less fuss. Years ago I did a good deal of custom work on an e-commerce application that was not written to be extensible. When a new version came out it took as long to *integrate my changes* into the new version as it did to build them in the first place.

But with Farcry you can keep your changes separate within the project folder. The Farcry framework takes advantage of the *Extends* feature of cfcomponent. To dive into how it works you need to know that all the objects you work with in Farcry are "Types" that share the same base component (or super class if you prefer) called "types.cfc". You can take a look at types.cfc in /farcry_core/packages/. This component dictates that all types in Farcry share common properties (like objectid, status and last updated).

Inside this packages/types folder you will see an individual cfc corresponding to each of the individual Farcry types. Let's use a specific example. A navigation node in Farcry is a of the *type* dmNavigation. If you look into the /packages/types/ folder you will see a dmNavigation.cfc file. When you create or view a "navigation node" in Farcry you are manipulating this "type".

Extending a "type"

Here's the neat thing. When Farcry first initializes it checks your *project folder* to see if there is a packages/types/*typename*.cfc file there. If there is one of these files, Farcry will automatically use it in lieu of the core type component. Sounds dangerous doesn't it? Actually, it's as easy as extending the *core type*.

Let's say you wanted to add an additional property to the *dmNavigation* type. To accomplish this task you would need to create a file in **project*/packages/types* called dmNavigation.cfc. At the top of this new file put the following:

```
<cfcomponent name="dmNavigation"
extends="farcry.farcry_core.packages.types.dmNavigation">
...
</cfcomponent>
```

Then add updateapp=1 to the url to refresh the application scope. If you check the debug you should see that this file is being called instead of **farcry_core/packages/types/dmNavigation.cfc*.

So far you have "extended" the dmNavigation type but it's a mere copy of the core file

right? What's next? The next step is to add a new property.

```
<cfproperty
    name="listStyle"
    type="string"
    hint="Add style to dmNavigation elements">
```

Refresh the app scope again and something magical happens. Not only is there now a new property added to dmNavigation, there is a new column added to the database. That's right - the fourq system (a brokering system that allows for the support of various db platforms through a single interface) picks up the new property and alters your DB schema to include the new column. The *type="string"* attribute of the property dictates the data type. Farcry supports the following types through fourq.

- boolean
- date - coorelates to datetime
- Numeric
- string - length of 255
- nString - length of 255
- uuid - length of 50
- variablename - length of 64
- color - length of 20
- email - length of 255
- longchar - text field

In other words, if you enter a property with a type of "string" the system will create a column in your dmNavigation table named after your property with a character type and a length of 255 chars. So, if (using MSSQL) you examined the dmNavigation table after adding the property above you would see a new column called "listStyle" set to varchar(255).

Next Steps

So adding a property preps the database and ensures that the property will be present in all your calls to that object. But how do you get it populated? After all the form doesn't automatically contain a text box for the property - does it? For that you will need to use your extended component and override the *edit* function. Here are the steps.

- Copy the _dmNavigation folder from *farcry_core*/packages/types to *project*/packages/system (at least that is where we put it although I can't remember why).
- Copy the "Edit" function from *farcry_core*/packages/types/dmNavigation.cfc into your new dmNavigation.cfc file. It will look like this:

```
<cffunction name="edit"
    access="public" output="true">

    <cfargument name="objectid" required="yes" type="UUID">

    <!--- getData for object edit --->
    <cfset stObj = this.getData(arguments.objectid)>

    <cfinclude template="_dmnavigation/edit.cfm">
</cffunction>
```

- Edit the "cfinclude" line to point to your new folder. Your function now looks like

this:

```
<cffunction name="edit"
    access="public" output="true">

        <cfargument name="objectid" required="yes" type="UUID">

        <!--- getData for object edit --->
        <cfset stObj = this.getData(arguments.objectid)>

        <cfinclude template="../system/_dmnavigation/edit.cfm">

    </cffunction>
```

- Open up the edit.cfm file found in your system/_dmnavigation folder and edit it to support the new property.
    - Add <cfparam name="*property name*" default=""< to the block of cfparams.
    - Add stProperties.*propertyname* = form.*propertyname* to the block of cfsets handling the stProperties object.
    - Find the "cfset" statements immediately prior to the form loading that set up properties in the form and add a new cfset:

```
<cfset title = stObj.title>
    <cfset externalLink = stObj.externalLink>
    <cfset lNavIDAlias = stObj.lNavIDAlias>
    <cfset fu = stObj.fu>
    <cfset *propertyname* = stObj.*propertyname*>
```

    - Add a new form element into the form to handle the new property. Make sure and populate it with your property variable like so:

```
<label for="*propertyname*"><b>*property name label*:</b>
        <textarea name="*propertyname*" cols="42">#*propertyname*#</textarea>
    </label>
```

Now, when someone goes to edit a dmNavigation item from this project, your new edit form is called instead of the default edit form. The update and create functions automatically take care of themselves. NOTE: You may need to override other items in your type component. For example, we created new includes for renderOverview and renderObjectOverview, and overrode those 2 functions in our dmNavigation.cfc.

Wrap Up

You might be wondering why we would go through the trouble to do this. In our case we were using dmNavigation to support a non-typical navigation tree and behavior. We need to be able to specify the "width" of a particular LI tag. It seemed easiest to us to simply add that property to the dmNavigation item and use it inside of our navigation custom tag, then include how to adjust for width in our training.

Please note that I'm not yet an expert on extending Farcry. If there is something I overlooked or if there is an easier way to do it - please add to our collective knowledge and comment below. As always, the muse appreciates any help he can get.