Watch out for Coldfusion Mappings When Using onReqest( )

Posted At : January 27, 2006 5:19 PM | Posted By : Mark Kruger
Related Categories: Coldfusion MX 7, Coldfusion Tips and Techniques

I love using the application.cfc file instead of application.cfm. The cfc approach encapsulates several events inside of automatically fired functions that formerly required "hand coding". For example, I used to check to see if application vars existed and set them if they did not. This required thinking about locking and testing one or more variables for existence (isDefined() or structKeyExists()). Using Application.cfc means this job is handled by the onApplicationStart( ) function. One function that belongs to Application.cfc deserves a bit more attention - the *onRequest( )* function. Here's an example from a webtop application that forces login.

```
<cffunction name="onRequest">
   <cfargument name="targetPage" type="String" required=true/>

   <!--- include some local functions --->
   <cfinclude template="udfs.cfm">
   <!--- include local vars --->
   <cfinclude template="localVars.cfm">
      <!---check to see if they are logged in... --->
   <cfinclude template="includes/control.cfm">
   <cfinclude template="#Arguments.targetPage#">

</cffunction>
```

UDFs and variables

You might notice that I'm using includes from some UDF's and some local variables. If, like me, you have a library of functions and some preset variables that *used* to go in a script block in application.cfm, you will have to put them here. Put them anywhere else and they will not be scoped correctly. Why? Because from this point forward the entire request is *inside this function* - so the onRequest( ) function's "this" scope is the "variables" scope.

Controlling Access

The third include in the code above is a "control" script. It checks the user's credentials and displays the login page if the user fails to pass muster (or pass the mustard as my 10 year old son Matthew informed me - "Dad, the teacher said we failed to pass the mustard - what's that mean?"). Again, it must go here in order to set things that we intend to "display". Plus, in this case, we need session and application variables that are fired by the other functions in order to make the control script work.

The target Page

The "targetpage" parameter is passed automatically to the function. It is (as far as I can tell) the value of *Cgi.Script_Name*. This is important because it includes the slashes. In otherwords, if the script name is "/dirA/dirB/myScript.cfm" Then the cfinclude will look like this:

```
<cfinclude template="/dirA/dirB/myScripts.cfm">
```

Why doesn't it just use "myScript.cfm" instead? Because the Application.cfc page might govern an application with subfolders - so naturally it needs the full relative path. It

makes an assumption about the path in which *it itself* is running and uses that assumption in the include.

Mapping Uh-Oh

"So what" you say... If I'm running the application.cfc file from "/dirA/dirB/" then the include will work fine, right? So no matter how deep my application is in the directory structure it will be able to "find" the correct included file. Ah... actually the answer to that is the same answer given by Bob Dole when asked during the presidential campaign if he wore boxers or briefs - "depends".

What if I have a Coldfusion mapping called *dirA*. In that case Coldfusion will look in whatever directory I specified in the mapping for the file ("dirB/myScript.cfm"), and if it fails it will issue a "file not found" error. This, dear friends, will drive you crazy. You will look in the directory structure and see a file called "myScript.cfm", but when you type it into the browser it will simply say not found. You will check the *trusted cache* setting. You will jump to the conclusion that you should have been a .NET programmer, but until you remember to check mappings you are going to be quite frustrated.

This happened to us recently. We had an application with an acronym for a name. We used the acronym as a Coldfusion mapping so we could reliably point our createObject( ) code to the right components. What we *didn't* realize was that this acronym would be a fairly typical choice as a sub-folder for folks installing our application. The result - when they went to the webtop they got the "file not found" error.

To put it another way, you cannot have both a mapping and a subfolder (off of the root) with the same name but different paths and expect to use onRequest( ) in it's most typical fashion. You will have to either

- Rename the mapping
- Rename the sub-folder
- Manipulate the "targetPage" variable

Otherwise you will get a file not found error.