CFMX and the Dot Operator - Migrating From CF 5 to CFMX

Posted At : March 14, 2006 11:23 AM | Posted By : Mark Kruger
Related Categories: Coldfusion MX 7, Coldfusion Tips and Techniques

If you come from the old "Coldfusion 4-5" days (in fact many or our customers are still running CF 5) then you might remember how those earlier versions handled variables with periods in the name. If you created a variable with a period in the name CF simply treated the period as if it were part of the variable name. For example, if you did the following in CF 5:

```
<cfset var1.var2.var3 = "My Dotty Variable">
```

You would not have created anything more than a primitive variable named "var1.var2.var3". If you tried to use <cfdump ...> to dump out *var1* it would generate an error - var1 not found. If you *intended* for var1 to be a structure containing a structure var2 containing a primitive var3 then you would have to rewrite the code like this:

```
<cfset var1 = structNew()>
   <cfset var1.var2 = structNew()>
   <cfset var1.var2.var3 = "My Dotty Variable">
```

Fast forward to CFMX.

In CFMX the Dot is an Operator

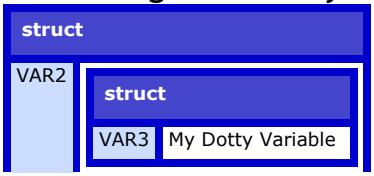
In CFMX the dot is special. It is used as an "operator". No, that's not a middle aged woman in horned rim glasses asking "how may I direct your call?" It means that the period performs a function or operation on the "operands". To give a simple example, consider this equation.

```
3 + 5 = 9
```

The 3 and the 5 are the "operands" - the inputs - and the "+" is the operator. In CFMX, the string to the left and the string to the right of the period are the operands and the dot is the operator. So the code above (var1.var2.var3 = "My Dotty Variable") is actually instructing CF to do something specific to var1 and var2 and var3. It is telling CFMX to A) create a primitive variable called "var3" and place the string "My Dotty Variable" in it, and B) create a structure called "var2" and add var3 as a key and C) create a structure called var3 and add the structure var2 to it as a key. The result is an primitive inside of a structure inside of another structure. If you run this code in CFMX:

```
<cfset var1.var2.var3 = "My Dotty Variable">
<Cfdump var="#var1#">
```

You will get this very interesting result:



As you can see, CFMX has automatically created nested structures based on your operator.

The Gotcha

There is one important caveat to this behavior. It has to do with key naming in structures. You see while it is not possible to create a primitive (non-object) variable using the syntax above, it *is possible to create a structure key that includes a dot*. For example, if you had the following code in CFMX:

```
<cfset variables["var1.var2.var3"] = "My Dotty Variable">
```

You would produce a key in the variables scope called "var1.var2.var3". In other words, the behavior would be *exactly like CF 5*. Why? Because you are not using the dot as an operator, and CFMX allows literally *anything* to be a key name for a structure. If you don't beleive me, see my post on **isDefined() vs. structKeyExists()**. Where this sometimes has implications is when items are passed in the URL or FORM scope. These are both structures, so passing a url variable like "?var.var2=blah" is the same as saying "url['var1.var2']" - in other words, no dot operator. So this code:

```
<cfoutput>#url["var1.var2"]#</cfoutput>
```

Would work, but this code:

```
<cfoutput>#url.var1.var2#</cfoutput>
```

Will complain that there is no variable named "var2" in the object named "var1". If you have old code lying around that passes variables like this but doesn't use the bracket notation (as in early versions of Fusebox) it will cause you problems.

Solution

The solution is to *avoid using dotted notation for primitive variables*. Only use dotted notation for variable naming when you *intend* to create a structure. Especially avoid passing such dottily named variables as URL or Form variables. It can be a real headache to ferret out all the places where this code exists when you are migrating code from CF 5 to CFMX.