IsDefined() Vs. StructKeyExists() - The Nuances of CFMX Structures

Posted At : September 8, 2005 6:26 PM | Posted By : Mark Kruger
Related Categories: Coldfusion MX 7, Coldfusion Tips and Techniques

There's a very interesting post and discussion on Brian Kotek's blog today dealing with the use of the function "isDefined( )" vs "structKeyExists( )". If you've been programming since the days of CF 4.x and "parameterexists()" you'll know that using isDefined() can be a delicate experience at times. It becomes especially tricky in CFMX where the way variables are initiated has changed. In the old days you could have a variable in the variables scope that included a period in the name, and unless you specifically called "structNew()" it would stand as primitive variable in the variables scope. Take this Example:

CF 5 Example 1

```
<cfset foo = 1>
   <cfset foo.bar = 2>
```

This creates 2 variables in the variables scope - "foo" and "foo.bar". Don't let the period fool you! It's a regular primitive variable in the variables scope - no structure exists called "foo". To put it another way, you have created "variables['foo']" and "variables['foo.bar']" (even though we know the variables *structure* wasn't introduced till cfmx).

Try the same code on CFMX and what happens? You get an error of course. Why? Because you have taken a primitive variable "foo" that was already initiated and treated it as if it were a structure. That's how CFMX treats the period in the set statement - an implicit call to *structNew()*.

To illustrate, try this code on CFMX:

CFMX Example

```
<cfset foo.var1.var2 = 'test'>

<Cfdump var="#foo#">
```

Notice that CMFX builds a structure "foo" with a key "var1" containing another structure with a key "var2" containing the string "test". On a CF 5 server the code would error out on the cfdump complaining that the variable "foo" doesn't exist.

Structure Fun

This behavior on CFMX can lead to some interesting consequences. I'll leave it to you to read Brian's post and the insightful comments, but let me summarize (or as my dad the Pentecostal Preacher used to say "as I continue to close"). Using structures in CFMX you can get around the limitations on variable naming. As you know, variable names in CFMX must start with a letter - for example, Code like:

```
<Cfset 33Foo = 'test'>
```

...would throw an error and complain that "33Foo" is not a valid identifier. However, CF is not really picky at all about the "key" identifiers in a structure. Pretty much anything you want to throw in there will work - numbers, spaces, special characters etc. are all fair game. Consider the following on CFMX:

This Code Errors Out

```
<Cfset 33Foo = 'test'>
<Cfset Foo* = 'test'>
<Cfset *Foo = 'test'>
```

This Code Works fine

```
<cfset variables['33Test'] = 'hello world'>
<cfset variables['Test*'] = 'hello world'>
<cfset variables['foo Test'] = 'hello world'>
```

So now you have a variable placed in the "variables" scope whose name doesn't conform to the stated standard for variable names. You can still use them as long as you scope them with bracket notation. Now using non-compliant names is a very bad idea, but because CF won't complain about it, you could introduce bugs accidentally. If you are like me, you have a great deal of code that builds structures out of "unknown" or "runtime" key names. This makes the rules for writing that kind of code easier - but harder to debug. Oh, and just in case you thought I was kidding about CF taking "everything" as a key name, check this out:

Bad Code that Works

```
<!--- special characters --->
<cfset variables['adflkjlekruoiauofiuoidufaelkruios sa()%$*@$*'] = 'hello world'>
<!--- how about line feeds?? --->
<Cfset lFeed = chr(10) & chr(13)>
<cfset variables['*test' & lFeed] = 'hello world'>
```

As you can see, pretty much everything is fair game.

Using isDefined()

I'm not complaining about how everything is a structure. I like the fact that virtually everything is accessible as an object with validation functions and the like. However, this "feature" of CFMX has an impact on the use of the "isDefined()" function. If you allow non standard variable names as keys in your structure, and then use "isDefined()" to try and test against 1 key or another you may get an unexpected result. This is the point made in Brian's blog so I won't belabor it here except to give 1 example.

```
<cfset foo = "blue">
<Cfset variables['33foo'] = "Red">

<cfoutput>

   #isDefined('variables.foo')#   - returns yes<br>
   #isDefined('variables.33foo')# - throws an error<br>
   #isDefined("variables['33foo']")# - Throws an error<br>
</cfoutput>
```

The point to emphasize here (and the point emphasized by the comments on Brian's blog) is that isDefined() checks not just if a variable exists, but if it is "syntactically correct". That's why, when dealing with structures (and almost everything is a structure) you should avoid isDefined() in favor of "structKeyExists()".

In the case of our test above:

```
<cfset foo = "blue">
<Cfset variables['33foo'] = "Red">

<cfoutput>

   #structKeyExists(variables,'foo')#   - returns yes<br>
   #structKeyExists(variables, '33foo')# - returns yes<br>
</cfoutput>
```

Now we are able to get the "proper" result from our bad code (ha).

To recap - isDefined() is an inferior choice to structKeyExists() when dealing with "runtime" variables (where the name is not pre-determined) because it will throw an error even when the code to set the variable into the structure does not. But the larger lesson is, strive to ensure that your variable names conform to the stated standard.