The Boolean-O-Matic: ColdFusion's Weird Relationship With Truth

Posted At : February 5, 2010 12:28 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Tips and Techniques

Hello muse readers. I apologize for my long hiatus (which means a stretch of time where I was absent - it's not a size joke). I have been swamped with closing out the old year and implementing plans for the new year. I'm afraid our little chats were put on the back burner temporarily. However, now that new year has begun I am committed to continuing our friendship. I'd like to start out with something simple. Indeed, some of you may find this to be ColdFusion 101.

This post is going to discuss Boolean values. A Boolean is one of those datatypes more defined by how it is evaluated than by what it contains. The muse definition is that if something can return a "true" or "false" in the context of a logic statement (cfif) it is a Boolean. It may be other things as well, but it has the properties of a boolean and returns one of 2 states - true or false. Interestingly, every language handles Booleans differently and many of them use the same wild west sort of approach that ColdFusion uses - where several things can be used as Booleans.

Even if you don't know it, you use Booleans every time you create a cfif statement. Still, it's surprising how many advanced developers do not fully grasp all the ways that ColdFusion has of evaluating something as *True* or *False*. And having said that I am fully aware that some smarty-pants developer will immediately inform me of some new way I haven't seen before of evaluating true or false (thank you sir, may I have another).

Anyway, I'd like to take a little journey into the world of Booleans to start off my 2010 blogging. Note: this post has a number of neat "tips and tricks" that you may have not seen before. Whether you choose to use them can depend greatly on your environment, the structure of your code and the standard you are using (especially in a team environment). I'm not advocating for or against, although I have my own preferences. I'm only putting it out there as another arrow in your quiver. So with that caveat taken care of, let's begin.

Typeless

Here's some deep background. You may have heard it said that ColdFusion is a "typeless" language. This is certainly true, although with cfparam, JavaCast and CFC argument declarations it *is* possible to specify the required "type" of a variable. What does it mean that CFML is typeless? It means that you can create variables willy nilly without specifying what "type" of data they contain (string, decimal, float, date etc). Under the hood CF does all sorts of checking to figure out what you want. For example, if you do something like:

```
<cfset x = 10>
```

ColdFusion will note the lack of quotes, evaluate the variable and create a variable x with a type of "numeric" – all hidden from you. This allows you to do something like:

```
<cfset x = x + 5/>
```

Of course this wouldn't be possible if ColdFusion did not know (or at least be able to

assert) that x was a numeric value. ColdFusion performs this same magic with dates and strings and other types of primitive values. Typelessness is what makes it possible for a developer to concatenate a number with a string or Boolean with a number or whatever. ColdFusion flips through the data types it can possibly use and figures out which one will work for the statement you have written. In some other languages this is not possible. In SQL for example if you try to concatenate a character column with a numeric column the server will complain about an "incompatible data type" and you will have to re-cast the column into the proper type.

When it comes to Boolean values ColdFusion's penchant for figuring things out for you has some crazy nuances. But let's start with a "normal" example.

```
<cfset ishomerhappy = true>

<cfif ishomerhappy>
    Whoo Hoo!
<cfelse>
    Doh!
</cfif>
```

Notice a few things. First, I'm using the word "true" without quotes. I believe ColdFusion knows this is a boolean value and it needs no "conversion" to figure out how to evaluate it in the CFIF statement (I'm assuming here). As a boolean it is either true or false. But what if we did this?

```
<cfset ishomerhappy = "false"/>
<cfif ishomerhappy>
    Whoo Hoo!
<cfelse>
    Doh!
</cfif>
```

Now ColdFusion no longer has a Boolean variable to work with. Instead it has the "string" false. But just like with the numeric conversion ColdFusion ticks through the possibilities and *knows* that the *string* "false" can stand in for the *Boolean* "false". So the code above works (indeed, it doesn't even matter what case you use for the string).

But Wait There's More

Ok... so ColdFusion can evaluate the string "false" and "true" as the real live Boolean false and true - so what. All that means is that I don't have to remember whether I put quotes around certain variables - right? Well there's more to this saga. ColdFusion can also evaluate "YES" and "NO" in the same way. Early on there were some language choices made for CFML where certain attributes of tags took "YES" or "NO" for the string value. For instance, cfqueryparam uses YES and NO for the "list" and "null" attributes.

```
<cfqueryparam
    list="Yes"
    value="1,2,3"
    cfsqltype="CF_SQL_INTEGER" />
```

I'm not sure why yes and no were chosen over true or false. My guess would be that the

origin is in some other language familiar to the CFML authors - or perhaps they felt it would be more intuitive for folks coming from HTML. Either way, the strings Yes and No may also be converted to Boolean. So the following works:

```
<cfset ishomerhappy = "YES"/>

<cfif ishomerhappy>
    Whoo Hoo!
<cfelse>
    Doh!
</cfif>
```

There are two things to note here. First, to set a yes/no it must be a string with quotes around it. If you try this:

```
<cfset myvar = NO/>
```

It will throw an error. Secondly, the strings yup, nope, nay, yay, yeah, uhuh, word, nah, nine, nada, si, oui, and fershizzle do *not* work. You will have to stick with just "yes" or "no" to use this technique.

There's Still More!

The third way that CF handles Boolean values closely mimics the world of C programming. In C any numeric value (signed integers I believe) can be evaluated as true or false. The rule is, if it is a zero it is false. Everything else is true. So, 1,2,9,-40 ... all true. Only zero is false. The exact same rule applies in CFML but CF handles *any* number (integer or not). A decimal, or whether a number is negative or positive do not matter. zero is false, everything else is true. Try it for yourself using this little snippet:

```
<cfset boolnum = 545>
    <cfset boolnumdec = 545.5>
    <cfset boolnegnum = -50>
    <cfset boolzero = 0>
    <cfif boolnum>
        Any positive number IS True<br />
    </cfif>
    <cfif boolnumdec>
        Even a decimal evaluates as true<br />
    </cfif>
    <cfif boolnegnum>
        Even a <em>negative</em> number evaluates as true<br />
    </cfif>
    <cfif boolzero not>
        Only a zero evaluates as false<br />
    </cfif>
```

How is this useful? One way is in evaluating the success of a query. Instead of doing the following:

```
<cfif myQry.recordcount GT 0>
        My query has rows!
    </cfif>
```

...which compares the recordcount against a threshold numeric value, I can now do

this...

```
<cfif myqry.recordcount>
    My query has rows!
</cfif>
```

Which simply transforms the "recordcount" into a true/false value (no rows returned is false, *any* rows returned is true). Another easy use of this principle is to combine it with the val( ) function. Let's say I have a query that returns a column called "expenses" that might contain a number or it might be blank (CF treats a null column as an empty string in a returned query). You could pretty easily do the following:

```
<cfif val(myqry.expenses)>
    ...do something with expenses...
</cfif>
```

I'm sure you can see how thinking along these lines could be helpful. Please note that there are those who will tell you this is bad form - and if you are in an environment where this is not best practice you should naturally follow the standard. In practice I will tell you that there is no perceivable performance penalty for doing it one way or the other. That makes it a judgment about what the code is to *look* like (readability, standards and practice). I *personally* find the code above to be perfectly readable and I have no questions about what is being done within the cfif statement.

It Make's Julienne Fries

Finally, there are some nuances you may not have thought of regarding how boolean values are handled in your IF/Else logic blocks. Consider this block of code.

```
<cfparam name="url.action" default="screen" />
    <cfparam name="url.someID" default="0">

    <cfif url.action IS 'print' AND url.someID IS NOT 0>
        <cfset showprintdisplay = true>
    <cfelse>
        <cfset showprintdisplay = false>
    </cfif>
```

If you have been around the block enough times you have seen this sort of code before. It is often used in a procedural manner to check some action or variable at the top of the execution sequence in order to affect some display or value later on down the line. The logic of the statement is "IF action is 'print' and SomeID IS NOT 0" then set a value called "showprintdisplay" to true, otherwise set it to false. This logic insures that A) there will always be a "showprintdisplay" variable and B) it will always be true or false.

Let's take a look at 2 properties here. First, you may *not* know that pretty much any logic block can be output to the screen like so:

```
<cfoutput>
#url.action IS 'print' AND url.someID IS NOT 0#
</cfoutput>
```

In this case it returns the string NO or YES (not true or false - weird right?). You might have noticed that I did not use my little val( ) trick. I did this to show off another interesting nuance. The *order in which the statements appear and the choice of yes/no, 0/1 or true/false can change the format of the return value*. We have shown how ColdFusion can evaluate Yes/No, True/False, and 0/not 0 as boolean. But did you

know that ColdFusion also returns all three of those type of values as Boolean to it's own statements? Check this out.

You can return a number by evaluating a true/false value with a number as in - true/false first AND number second. This example returns a *1*:

```
<cfst tnum = 1>
    <cfset tval = true>
    <cfoutput>
    #tval AND tnum#
    </cfoutput>
```

If you reverse the position and try to AND the number with the true/false value it will return a true/false. This example returns *true*.

```
<cfset tval = true>
    <cfset tnum= 1>
    <cfoutput>
    #tnum AND tval#
    </cfoutput>
```

To return YES/NO the first statement must be an evaluative statement that returns yes no. This example returns YES or NO presumably because the statement "url.action IS 'print'" takes precedence over the number returned by val(url.someid).

```
<cfoutput>
#url.action IS 'print' AND val(url.someID)#
</cfoutput>
```

Wow... I'm tripping over that. I haven't been this loopy since I played Joust for 10 hours straight in college one semester.

Final Wild West Tip

You can see there is a great deal happening under the hood when you build a logic statement. Perhaps it occurred to you, "If I can output the values of a logic statement to the screen, why can't I set it as a variable?" You absolutely can. Remember our example?

```
<cfif url.action IS 'print' AND url.someID IS NOT 0>
        <cfset showprintdisplay = true>
    <cfelse>
        <cfset showprintdisplay = false>
    </cfif>
```

It could just as easily be written:

```
<cfset showPrintDisplay = (url.action IS 'print' AND url.someid IS NOT 0)/>
```

This would create a showprintdisplay variable of .... well yes or no in this case. But even if it is 1/0 or true/false you can still treat it in your code exactly the same. Am I advocating this approach? Not really - at least not with complex multi-part statements. I think it would make the code less maintainable in the long run. Indeed I'm not a huge fan of the procedural approach above to begin with. But it's still pretty cool that it can be done.

Conclusion

Once again let me say I'm sorry for my lengthy absence. Even the muse needs a break to recharge his batteries from time to time. I hope to begin posting my long anticipated (by me mostly...) cloud computing series soon. Meanwhile, I look forward

to comments on this post from the bright folks who read my blog and always contribute richly to my way of looking at things.