Cast Your Java Arrays Upon the Water and They Shall Return in Due Season

Posted At : November 19, 2007 7:46 PM | Posted By : Mark Kruger
Related Categories: Coldfusion 8

If you have ever had to use a Java Widget from inside of your Coldfusion page you might know about the *Javacast()* function. Javacast allows you to bind a variable to one of Java's primitive "types" like string, long, float etc. Why is Javacast() necessary? I'm glad you asked. In the old days of Coldfusion 4 and 5, CF treated any variable as a sort of mystery meat. It didn't look too closely at it until you chose to do something with it. Only when an operation *required* a variable to have the properties of a certain type did Coldfusion complain. This is still true to a large extent today.

If you have a URL variable called "total" that is supposed to contain a number, you can populate it however you wish. You can put the string "sasquatch" in it for example. Coldfusion will ignore the type and pass the variable along right up until you try to add shipping to it, or divide it by another number or something. Only then will Coldfusion throw an error. Of course, using CFCs allows you far more control over the "type" of variables in play - but you can still treat your variables in this "weakly typed" way if you wish.

Java, on the other hand, is a strongly typed language. Variables are defined as "new" this and "new" that - where the this and that are classes or objects with methods and properties. So things written in Java don't take just anything for arguments to worry about at runtime. Javacast() was invented to convert Coldfusion variables into specific java types that Java methods are expecting. Take the following as example:

```
<cfscript>
    myVar    = 12;
    foo     = createobject("java","myclass");
    //acceptfloat takes a float
    foo2you    =    foo.acceptfloat(myvar);
</cfscript>
```

If the java function "acceptfloat()" complains that myVar is not of the right type you can change that line to be **foo.acceptfloat(javacast("float", myvar))**. This ensures that the variable is passed to the Java function containing all the properties and methods of a "float" type in Java. Note - there's a good chance that the function above might succeed without complaining - but you get the idea.

The Problem With Java Methods

Ah... but he problem with Java methods is that they often require arguments that are "typed" as other classes or (in many cases) as an array of strongly typed variables. For example, **CF Webtools** is currently working with some Java classes that run calculations against stock and derivative price data. These Java objects need arrays of floats as arguments. Up until CF 8 this was not really possible. We would have had to create some sort of wrapper class to get the result we needed. Thankfully with Coldfusion 8 this has become trivial. Now, it's possible to write code like this:

```
<cfscript>
    myVar    = arraynew(1);
    myVar[1]    =    13.00;
```

```
    myVar[2]    =    18.00;
    myVar[3]    =    21.00;
    myVar[4]    =    12.30;
    foo     = createobject("java","myclass");
    //acceptfloat takes an array float
    foo.acceptfloat(javacast("float[]", myvar))
</cfscript>
```

Coldfusion converts each index in the array into a float and passes it as an array of strongly typed float objects. In the words of Hugh Neutron - "Now you gotta admit that's pretty neat!" You should note that it is important that all the vars that exist in the array should be "castable". If you are casting to floats you should not have 1 var be "bob" and the next var be "13.00". This requires a bit more attention than Coldfusion programmers are used to paying. We are used to stuffing pretty much any type of data into a query or a structure. But if you are working with arrays of specific types then all of the data will need to comply with the rules for that type. Also note that you cannot cast an array of "nulls".

Advanced Casting

What if your Java method takes an array of *objects* or *classes* as an argument? Actually this is possible as well - although I'm struggling to find an example for you. The example given in the docs is an array of classes:

```
<cfscript>
javacast("vom.x.y.MyClass[]", myCFArr)
</cfscript>
```

That doesn't tell you much does it? What is in myCFArr? I assume that each member of the array is of type "vom.x.y.MyClass" - but that code is not present. I can imagine code like this:

```
<cfscript>
    foo     = createobject("java","myclass");
    myVar    = arraynew(1);
    // create an array of "myClass"
    myVar[1]    =    foo.returnsMyclass(1);
    myVar[2]    =    foo.returnsMyclass(2);
    myVar[3]    =    foo.returnsMyclass(3);
    myVar[4]    =    foo.returnsMyclass(4);

    foo2     = createobject("java","myotherclass");
    //acceptmyclass needs an array of myClass objects
    foo2.acceptMyClass(javacast("myclass[]", myvar))
</cfscript>
```

Presumably the "returnsMyclass()" method would send back an instance of "myClass" and the acceptMyClass() method would take as an argument an array of class objects. But I have not tested this or had the occasion to use it. In the past I have run into exactly this problem using the Paypal API and solved it with wrapper classes - so I can see how it is intended to be used. If one of the muse' readers happens to have a working example I would love to include it in this post.