Tipical Charlie

(Tips and Resources from a Veteran CFML Developer)

by Charlie Arehart

How can I call thee (CFC)? Let me count the ways[1]

You may be surprised by the number of ways you can call a CFC and its methods. You should understand the differences, as each approach has its place. There are three ways to instantiate a CFC and/or call a method, and then there are also three ways of passing in arguments to a method.

The three approaches for instantiating an object and invoking its methods are either the CFINVOKE or CFOBJECT tags, or the function-based equivalent to CFOBJECT, CreateObject(). The ways of passing arguments in depend on the calling approach and include approaches for naming arguments in the call, passing in a structure, naming them positionally, or naming them in subtags.

Ways to Invoke Methods

CFINVOKE: creates an instance and invokes a single method, one time.

Many developers desiring to "invoke" a CFC method will naturally gravitate toward the CFINVOKE tag. It works very simply:

```
<cfinvoke component="CFCname" method="methodname"
returnvariable="returnvarname">
```

You name a CFC (without the .cfc extension) and a desired method (and optionally arguments, discussed later), and you get back its result in the variable named in *ReturnVariable*. It certainly works, but it has a couple of drawbacks. You can only invoke one method in the CFC per request. If you intend to call another method, you have to issue another invoke with the overhead that entails, because there is no stored reference to the object. Further, without a stored reference, you can't place that instance in a shared scope variable for later reuse.

Both of these problems are solved by CFOBJECT or its sibling, CreateObject(), which will also afford a new option for using CFINVOKE, as discussed at the end of the next section.

CFOBJECT/CreateObject(): creates an instance of the object as a ColdFusion variable with a specific name (named instance) and optionally invokes a method at the same time, or allows you to invoke a method later.

Developers who come from a more object-based background may feel more comfortable using CFOBJECT or CreateObject(). These alternatives feel more like you are creating an instance of an object, for those familiar with that concept from other languages:

```
<cfobject name="instancename" component="CFCname">
```

As with CFINVOKE, you do not specify .cfc in the CFCname.

---

[1] Of course, I'm making a play on the classic line from an Elizabeth Barrett Browning sonnet that starts, "How do I love thee, let me count the ways."

ColdFusion and BlueDragon both search for the file in a range of predefined locations, including the current directory, the webroot, the customtags directory, and mappings directories, but that is not the subject of this article.

You could then invoke a method on the CFC any number of ways, including a simple CFSET tag:

```
<cfset returnvariable = instancename.methodname()>
```

The equivalent syntax when using the function, CREATEOBJECT(), might be the following:

```
<cfscript>
    instancename = createobject("component","cfcname");
    returnvariable = instancename.methodname();
</cfscript>
```

For those not familiar with CFSCRIPT, note that you MUST use semicolons to end each statement. But you don't need to use this function only in CFSCRIPT. In fact, you don't need to instantiate the CFC and then call the method. You can do both at once. Here's an example of doing it all in a CFSET:

```
<cfset  returnvariable = createobject("component","cfcname").methodname()>
```

Note, though, that this approach of combining the instantiation and method invocation again leaves you with no stored reference to reuse for later purposes, just like CFINVOKE.

However, note that if you do create an instance as discussed in the first three samples of this section, you could also use that instance name to invoke a component method using CFINVOKE, replacing the component="CFCNAME" with component="#instancename#". For example:

```
<cfinvoke component="#instancename#" method="methodname">
```

Ways to Pass Arguments

There are also several ways to pass in arguments, depending on which of the two approaches you are using (CFINVOKE vs. CFOBJECT/CREATEOBJECT).

Passing Arguments on CFINVOKE

For CFINVOKE, you can pass in arguments as additional attributes to the CFINVOKE tag. Extending the example above, I could pass in FirstName="Charlie":

```
<cfinvoke component="CFCname" method="methodname"  FirstName="charlie"
returnvariable="returnvarname">
```

There's no significance to the order of the attributes, and you can name as many as you want. Of course, you can also provide a variable for the value of one of the attributes. The type of data passed in is controlled by the argument's type (if any) as defined inside the CFC method. Any valid CFML datatype can be passed in this way.

If you do end up passing more than a few arguments, you may want to reconsider placing them all in the tag itself. All of these extra attributes can get difficult to read and maintain. Fortunately, CFML offers two alternatives, one of which is shared with the CFOBJECT/CREATEOBJECT approach.

The first is the ArgumentCollection, which can point to a structure, whose keys are the names of the arguments to be passed. If I wanted to pass in firstname and lastname to a CFC method, I could create the structure as:

```
<cfset myargs=structnew()>
<cfset myargs.firstname="charlie">
<cfset myargs.lastname="arehart">
```

You can call the structure anything you want, and it can have any number of elements. The order is not significant, and their values can be variables. I could pass the structure to my method in the CFINVOKE using the special ARGUMENTCOLLECTION attribute. It's a keyword in CFML.

```
<cfinvoke component="CFCname" method="methodname"  argumentcollection="#myargs#"
  returnvariable="returnvarname">
```

Yet another approach, specific to CFINVOKE, is to use the CFINVOKEARGUMENT subtag, where you include one for each desired argument to be passed to the method. An example would be:

```
<cfinvoke component="CFCname" method="methodname"  returnvariable="returnvarname">
  <cfinvokeargument name="firstname" value="charlie">
  <cfinvokeargument name="lastname" value="arehart">
</cfinvoke>
```

Again, you can have as many as you want, the order is not significant, and the values can be variables.

Both of these approaches give you more programmatic control over what arguments and values are passed to the method. You can use *IF* statements or other logic to decide what arguments to include, both surrounding the CFINVOKEARGUMENT tags and when building the structure for the ARGUMENTCOLLECTION.

Passing Arguments to cfobject/CreateObject-based Methods

When calling a method in a CFC that's been instantiated with CFOBJECT or CREATEOBJECT(), you have similar but slightly different options. First, as with CFINVOKE, you can choose to name the intended arguments in the method call as argument=value pairs, but you do this within the method invocation, like so:

```
<cfset returnvariable = instancename.
methodname(firstname="Charlie",lastname="arehart")>
```

As with passing arguments in CFINVOKE, their order doesn't matter, you can have as many as you need, and the values can be variables.

There's yet another approach which may appeal to some developers. You can also just provide the values without naming the arguments:

```
<cfset returnvariable = instancename.methodname("Charlie","arehart")>
```

Now, this may scare some of you. How does the system know which argument is which? Well, they are passed in positionally, meaning that if any CFARGUMENT tags are defined in the CFC

method, their values are filled in the order that the argument values are presented. If there are no CFARGUMENT tags, or if more values are presented than such tags exist, then the values are placed sequentially into the arguments array within the method, and can be accessed by referring to the element number in the array.

Finally, as with CFINVOKE, you can also pass in arguments using the ARGUMENTCOLLECTION. Assuming we had defined the MYARGS structure as in the previous section, we could call the method as:

```
<cfset returnvariable = instancename.methodname(argumentcollection=myargs)>
```

All these approaches apply equally to invocation of web services as well as CFCs, though there are some quirks: optional arguments in a web service must be specified unless the new CFMX 7 Omit="yes" attribute is used on the CFINVOKEARGUMENT. Also, if arguments named username or password are passed to the invocation of a web service method, ColdFusion presumes they are used for passing authentication to the Web Service.

With all the variety of options available, it's incumbent upon the CFML developer to know the alternatives and use the best choice for a given job. I hope this brief introduction will help you in that decision.

A veteran CFML developer since 1997, Charlie Arehart is a longtime contributor to the community and recently became a member of the Adobe ACE program. Many know he served as tech editor of the CFDJ until 2003 and was co-author of the *CFMX Bible*. A certified Advanced CF Developer and Instructor for CF 4/5/MX, he's frequently invited to speak to developer conferences and user groups worldwide. Formerly CTO of New Atlanta (BlueDragon), he is now an independent contractor and still lives in Alpharetta, GA, where he is president of the Atlanta CFUG.