# FusionDebug Explained: Interactive Step Debugging for CFML

*By Charlie Arehart*

Many CFML developers really wish their arsenal of tools included an interactive step debugger. While most developers may assume that using CFOUTPUT and CFDUMP, among other techniques, can do everything such a tool would do, they have probably run into limitations with these techniques.. I have good news for both camps.

Interactive step debugging is now available to CFML developers in the recent release of FusionDebug, from the same folks who brought us the FusionReactor monitoring tool. Available at http://www.fusiondebug.com, the tool can help solve many problems where CFOUTPUT and CFDUMP are not options, such as debugging CFCs exposed as web services, or within CFCs with the output="no" option. Still other output-challenged spots exist within CFSILENT, or when generating XML (in which case a CFDUMP could be inappropriate).

Sure, CFTRACE and CFLOG can write their output to a file instead, but all that can be a hassle, and what if you simply aren't authorized to edit the template you want to debug? Besides web services, debugging challenges can crop up when debugging CFCs and CFML code that are called from Flex, Flash Remoting, or Ajax requests. FusionDebug can be used for all of these, as well as for debugging code running on a remote server. These are just a few of the tool's many powerful features.

In this article, I'd like to introduce the concept and the tool, then show what it can do for you, and finally explain why developers, even those who would dismiss debuggers, should give it serious consideration. Those already familiar with debuggers can skip to that last section. Then if I've piqued your interest, you can return to the beginning for a discussion of the tool's features, which may exceed expectations, based on your experiences with other such tools.

## What is Interactive (Step) Debugging?

Interactive or step debugging has got nothing to do with the debugging output at the bottom of your CFML page. Have you ever wished you could watch as your program executes from line to line? That's exactly what a step debugger does for you. More than that, you can watch the values of expressions and variables—and even change variables on the fly.

Such tools are common in other languages such as Java, .NET, JavaScript, Flex and Flash. CFML developers who have not used these may not even have noticed that we've lacked a debugger. Some may know that there was indeed interactive debugging in CFML in ColdFusion 4 and 5, by way of ColdFusion Studio (now HomeSite+). But Adobe (then Macromedia) chose not to carry that feature forward into CFMX, so FusionDebug represents the first and only way to do step debugging in CFML for ColdFusion MX 6.1 and 7. Those who have a bad taste left from their ColdFusion 4/5 days can take heart since this is a commercial product. There is a company behind FusionDebug to help support, improve, and evangelize it.

## About FusionDebug

FusionDebug is a commercial plug-in on top of the free Eclipse framework, like FlexBuilder. Priced at US$ 299, there's currently an available 10% discount code (CFCOMMUNITY) as well as volume discounts and a 20-day free trial. Better still, just announced at the end of

September is a discount of 20% as well as a new "Community Edition" also priced at $99. See the website, http://www.fusiondebug.com, for more.

Even so, your first response may be that you don't think you should have to pay for such a product, but it's really a rather small price to pay if you will benefit from debugging.

Your second response, if you don't use Eclipse currently, may be to worry about having to use it. First, note that *you don't need to give up your favorite CFML editor, whether it's DreamWeaver MX, ColdFusion Studio, HomeSite+, CFEclipse, or whatever. Further, you need to know only a minimal amount of Eclipse functionality*—which is very easy to learn—to use FusionDebug.

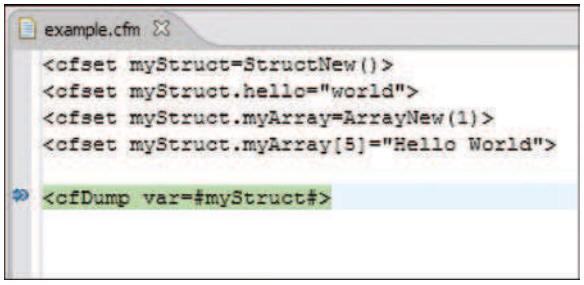
## Getting Started

You will need to download Eclipse, which is free. You may already have it installed if you have FlexBuilder or CFEclipse. It's very easy to get and install it if you don't and the process is explained in the FusionDebug User Guide, available online at http://www.fusion-reactor.com/fusiondebug/helpDocs/FusionDebug_User_Guide.pdf.

FusionDebug also requires just one minor single-line change in the jvm.config file for the CFML server you'll be debugging against, which is also easy to do and well-documented in the User Guide. There you indicate a port on which the debugger will listen, and then you do a minor setup in Eclipse to enable debugging against that server. All of this is well-documented and simple. I'd like to skip those details and focus on how to use the tool.

## First Stop: Setting a Breakpoint

After opening a file in the FusionDebug environment, you can begin debugging by telling the tool that you want to stop execution on a given line of CFML code. This is called setting a "breakpoint". You just right-click on the line of code in the Eclipse editor, and choose **Toggle Line Breakpoint**.



```
<cfset myStruct=StructNew()>
<cfset myStruct.hello="world">
<cfset myStruct.myArray=ArrayNew(1)>
<cfset myStruct.myArray[5]="Hello World">

<cfDump var=#myStruct#>
```

When that CFML template is requested and that line of code is about to be executed, the program will halt and the FusionDebug interface will reflect that execution has halted. It will open the file (if it's not already open) and show the line of code on which execution has stopped (*See figure at left*).

The blue dot to the left of the line shows where a breakpoint has been set. The blue arrow and the shading on the line indicate that control has halted there. (The browser window in which the page had been requested will generally look as though the request is just taking a long time.)

Indeed, a really cool thing about the way FusionDebug works is that it can intercept a request from any user, not just the user who has initiated the debugging session.

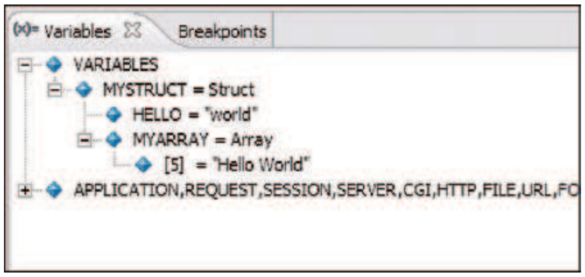
- This means that you can use it to intercept a request other than one you yourself initiated. How often have you tried to understand why a user is having a problem in a shared server environment (such as test, staging, or even production) that you couldn't recreate locally?
- It can also intercept a request via web services, Flex, Flash, Flash Remoting, or Ajax. More on all that later.

The problem, of course, is that it also means that anyone on the server being debugged who also runs that request will also be affected when you set a breakpoint. So caution is certainly advised in setting breakpoints in production. With power comes responsibility. Still, it's nifty that you can.

## Observing Program State Information (Variables)

Being able to stop the program in its tracks may seem only mildly interesting, but you can learn a lot about what was going on in the program at the time. For instance, you can see the values of all the variables that may have been set in the program or perhaps in other templates before this one executed.

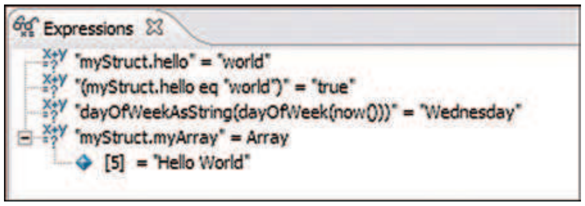
FusionDebug provides a "*Variables*" view, which in the case of the code above would show the following (*See image below*).



```
VARIABLES
  MYSTRUCT = Struct
    HELLO = "world"
    MYARRAY = Array
      [5] = "Hello World"
APPLICATION,REQUEST,SESSION,SERVER,CGI,HTTP,FILE,URL,FO
```

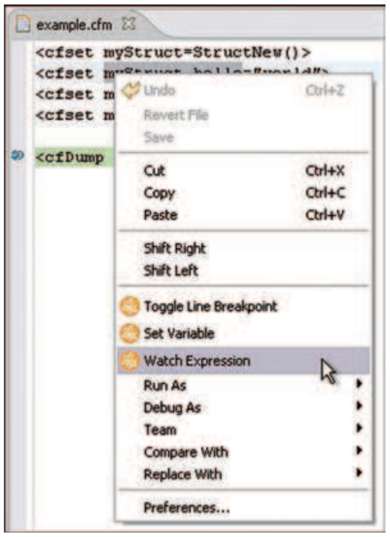
You can see that a structure with a key and an array has been created. Note how you could expand the local "variables" scope as well as any application, session, server, and other scopes. Indeed, if we were stopped within a method (CFFUNCTION or CFSCRIPT function), we could also see the local ("var") and, in a CFC method, the "this" scopes. Isn't that a whole lot easier than putting in CFDUMPS, and CFOUTPUT, and having to remember to remove them?

If you had a large number of variables that would make it tedious to explore this *Variables* view, another available option is "watched expressions". This is even more like using old-style outputs, except they never send output to the browser. Instead, the results are shown inside the debugger. With this "Expressions" panel, you can choose to watch any variable or expression. (An expression can be anything you might put on the right side of the "=" of a CFSET, or inside a CFIF condition, including variables, functions, and so on.) *(See Expression image to the left)*



```
"myStruct.hello" = "world"
"(myStruct.hello eq "world")" = "true"
"dayOfWeekAsString(dayOfWeek(now()))" = "Wednesday"
"myStruct.myArray" = Array
  [5] = "Hello World"
```

You can enter expressions by right-clicking in the Expressions View, selecting **Add Watch Expression**, and typing in the expression manually. You can also highlight an expression in the code editor, then right-click and select **Watch Expression** (*See firgure to the right*).



```
<cfset myStruct=StructNew()>
<cfset m
<cfset m
<cfset m

<cfDump
```

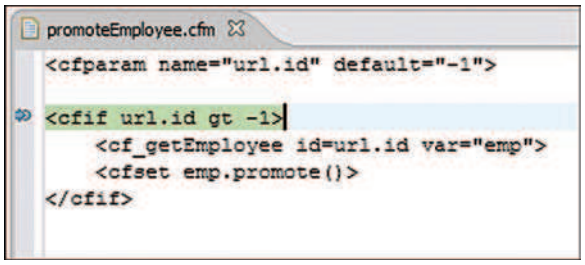
## Setting Variables on the Fly

Perhaps one of the most compelling things about step debuggers is that they permit you to do something you simply can't do otherwise: you can alter the value of any variable in your code while the program is running. All you need to do is right-click on the variable in the code editor (while at a breakpoint) and choose *Set* Variable. You then indicate the value, and that will take effect for the remainder of the request. Nifty!

## Stepping Through Lines of Code

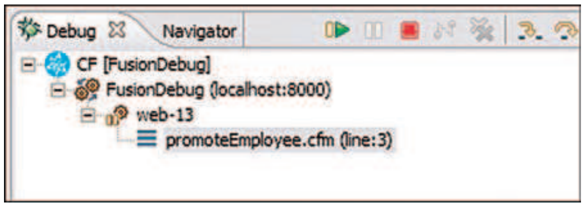
It's useful to stop at one point in the program and see all this, but another important fundamental feature remains — the ability to step through your code.

Consider the following code, stopped at the indicated breakpoint:



```
<cfparam name="url.id" default="-1">

<cfif url.id gt -1>
    <cf_getEmployee id=url.id var="emp">
    <cfset emp.promote()>
</cfif>
```

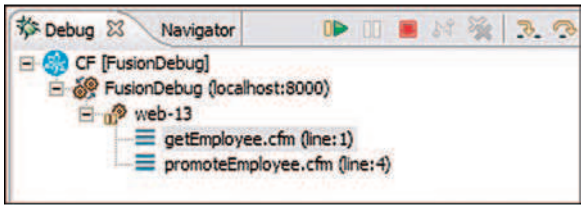


```
CF [FusionDebug]
  FusionDebug (localhost:8000)
    web-13
      promoteEmployee.cfm (line:3)
```

How do you tell it to proceed? Another pane in the FusionDebug interface, the Stack Trace pane, looks like figure to the left:

The icons at the top of the pane let you control execution. Perhaps the most commonly used icon is the one on the far right in the screenshot to the left. This is called **Step Over**, and it simply executes the next line of code. Other options exist outside of what's visible in the screenshot. All of the things you can do at a breakpoint apply while stepping through code as well.

Also available are options to indicate whether you'd want to follow the flow of execution into another file, such as a custom tag, CFC or included file. Note that the screenshot showing the code and breakpoint above indicates that the next line of code is a custom tag (cf_getEmployee). If we were to use **Step Into**, the icon to the left of Step Over, then FusionDebug would open the getemployee.cfm custom tag and stop at the first line of CFML within that. If the first line of CFML in that custom tag appeared on line 1, the stack trace would look like the following (which shows it stopped on line 1):



```
CF [FusionDebug]
  FusionDebug (localhost:8000)
    web-13
      getEmployee.cfm (line:1)
      promoteEmployee.cfm (line:4)
```

If you didn't want to step any more within the nested file, you could use an available **Step Return** button. And if you didn't want to step any more in any of the files in the request, you could use the left-most of those icons (what looks like a green arrow) which is called **Resume**. It would let the request run to completion (unless it hit another breakpoint).

## And Still Much More…

There's a lot more to show, but this should be enough to whet the whistle of those who haven't explored a debugger before. There are several learning resources on the FusionDebug web site, in addition to the User Guide mentioned above:

Information on additional features: http://www.fusion-reactor.com/fusiondebug/features.html

Captivate videos demonstrating their use:
http://www.fusion-reactor.com/fusiondebug/gettingStarted.html

Additional screenshots: http://www.fusion-reactor.com/fusiondebug/screenshots.html

## Why Use FusionDebug When You Can Just Use CFDUMP? A Dozen Reasons

Some will ask, "Why should I bother with or care about step debugging? I can get all I need with CFDUMP and CFOUTPUT." Some will go so far as to proclaim that they've coded for several years without a debugger, thank you very much, and just are not interested.

Still others may have prior experience with a debugger that has left preconceived notions of what they think FusionDebug can—or can't—do.

I'd like to conclude this article by showing the many ways (twelve, to start) that interactive step debugging can indeed be a valuable tool in the arsenal of a CFML developer, to solve problems that might otherwise be very difficult.

## You Can't Always Do a CFOUTPUT/CFDUMP

So why not rely solely on judicious (or indeed, copious) CFOUTPUTS and CFDUMPS? Well, there are simply places where you can't create output!

For example, when run within a CFC or method where OUTPUT="no" has been set, CFDUMP and CFOUTPUT create no output. You may think, "But I can just set it to yes." But there are times when doing so will introduce errors or otherwise unexpected results due to other code you've written.

Another situation in which you can't create output is within CFSILENT. No output is rendered to the browser. CFSILENT is commonly used, especially in complex apps and in most frameworks, and you will have to disable it by finding and editing the file that sets it so that you can debug. Again, there may be a negative impact by disabling the CFSILENT when the code after it executes, generating perhaps unexpected whitespace or output.

You could use CFTRACE or CFLOG to write output to a log file, true, but it's certainly not as simple as dumping output to the screen.

CFABORT is another common tag used when inserting CFOUTPUT and CFDUMP, but there are also times when doing CFABORT can have unexpected consequences.

Finally, you'll also need to remember to set all those changes back. More on that later. All these are reasons that sometimes you can't or may want to think twice about adding in code to do CFOUTPUT/CFDUMP.

## You Don't Always Have a Browser to Output to (Flex, Ajax, Web Services)

What if you're writing CFML code that is called by a Flex or Ajax client, or is a web service? You can't then always easily add debugging output—and you certainly can't do a CFDUMP, if the client is expecting XML or some other datatype, since CFDUMP typically creates a big HTML table.

*FusionDebug, on the other hand, can indeed be used for debugging requests from Flex, Flash Remoting, Ajax and web services apps*. In fact, it can intercept a call from any kind of client that requests a CFML page or CFC. Note as well that if you are using the debugger in FlexBuilder (which is also an Eclipse-based debugger), the Eclipse debugging "perspectives" will switch automatically between debugging Flex and CFML code.

## You Can Intercept and Debug a Request From Any User, Not Just the Developer

This is a very important point, and actually is an extension of the point above. Unlike the ColdFusion 4/5 debugger, FusionDebug is NOT limited to debugging only what you as a developer can browse yourself. It can intercept and show to the debugging developer the step-by-step execution of any template, CFC, custom tag or included file run by **anyone in any manner**. So you can use it to debug when someone else is running the request, where writing output to the browser would go to someone other than the developer. I know that may sound a warning bell to some. Hold that thought for a moment, and through the next couple of points.

## You Can Debug a Remote Machine, and Against Shared Servers

This may be the most powerful feature. FusionDebug can do debugging against any CFMX server for which it's been configured. It doesn't need to be just a CF server on the same machine that's running Eclipse. How often have you found something strange happening in production that you can't recreate in development? With FusionDebug, a real end user can run a request that causes a problem while you watch and debug it.

Of course, the above three points represent a two-edged sword. You can't currently limit the debugging to take place only for a given user, so if indeed you are using it on a server being used by multiple users, it will impact anyone who makes a request for that file while you have debugging enabled for it.

But let me be clear: for this to happen, a developer would have to a) open Eclipse/FusionDebug, b) turn on debugging against that application/project, c) enable a breakpoint for a given template, and the user would need to d) request that page and e) run through that code that would f) hit that breakpoint.

The user would see the page appear to hang until the developer who enabled debugging responded to let the page finish loading. It's just a responsibility to be aware of, which comes with the power of the tool. (There's also a modest performance impact in configuring the server's JVM config to support debugging, so you may want to think carefully about leaving it enabled all the time in production.)

## There's No Need to Change Code to Get Debugging Info

It may not always be desirable or possible to edit a file to add debugging code. For instance, it's clearly less palatable to just throw some CFOUTPUT/CFDUMP tags in production. Or perhaps you're debugging some code that someone has ruled should not be edited, or it's in a directory you don't have access to. And again, if you DO edit the code to add debugging tags, you have to remember to remove them when done. How often do we see code still showing debug output? FusionDebug does not require that you edit the code.

## You Don't Need to Be Able to Enable ColdFusion's Debugging Output

It's worth noting that by using FusionDebug, you don't need to enable ColdFusion's debugging output. Whether in production or simply on a server where you can't get debugging turned on, this can be a valuable benefit.

## You Can Change the Value of Variables on the Fly During Execution

FusionDebug also lets you change the value of variables within the code being debugged, on the fly, to temporarily alter the variable until the end of the running request. This is easier than editing the code to change the value for debugging purposes.

## Sometimes a Simple CFOUTPUT/CFDUMP Will Not Suffice to Solve a Problem

While you're stopped at a breakpoint, you can view the value of ALL variables in ALL scopes. FusionDebug presents a very easy to use tree view to traverse and view the scopes. This also includes query resultsets and more. And since you can view them all, you may be able to see information or make connections that you might not have thought to dump or output with traditional approaches.

## You Can Use the Debugger to Understand the Flow of Execution of the Request

Still another truly unique feature of a step debugger is that it gives you a clearly visual representation of the flow of a request. If you're ever wondering whether the code went into one IF statement or loop, or if it included a file or called a custom tag or called a method in a CFC, these are all things that you can readily see in FusionDebug. Of course, this is also a great way to introduce a new developer to your code or CFML in general.

## You Can Debug in Situations Where You Don't Even Know Where in a Complex App to Try to do CFOUTPUT or CFDUMP

Recall that FusionDebug opens files as you step through them. When following the flow of execution, this can be especially valuable in very complex applications. Let's consider (just as one example) a Model-Glue application. When I run the modelgluesamples/legacysamples/contactmanager/index.cfm and view the files in the traditional debugging output (showing all the files that were used to render the request, including CFM and CFC files and methods), there are nearly 150 shown. If something's amiss, you may be hard pressed to think where to begin. FusionDebug shows the flow readily.

One thing that FusionDebug does lack (which the ColdFusion 4/5 debugger had) that would help, especially in this last case, is a "wildcard" or conditional breakpoint: where it would stop when a given condition was met. That's SO valuable in a complex application. You can use it to find out when some condition arises, such as when a variable obtains or exceeds some value, or is set at all, or when some query exceeds a given execution time. I have requested this feature and hope it will make it into the product in the future.

## You Can View the Stack Trace During Execution

Following onto the previous point, while the traditional CF debugging output shows (at the end of the page request) what files were called to run the entire request, it doesn't really help you to know which were opened just to get to a particular point within a given template or CFC method. In FusionDebug, while you're sitting at a breakpoint, the stack trace pane will show all the files that were opened and the line of code executed in each to get to that point.

## You Can View the Java Classes Called to Execute Your Code

One last feature, especially for the bit fiddlers out there, is the "Java detail mode" in FusionDebug. If enabled, it will show in the stack trace pane all the Java classes and methods called to get to a point of code. If you double-click on a Java class, it will even show you any variables that were created from that object. For more information, including a screenshot with quick and easy explanations, see the FusionDebug site:
http://www.fusion-reactor.com/fusiondebug/featureFocus-javaDetailMode.html.

## Conclusion

That's a long list of reasons and benefits, and it's not even complete. I haven't yet mentioned the high-quality—and free—support (just drop a note to support@fusion-reactor.com). I do hope this introduction to step debugging and to FusionDebug's interface and features will help you get started. There are some gotchas worth noting. I elaborate on these and offer many more points in a series of blog entries I'm doing on FusionDebug, at http://carehart.org/blog/client/index.cfm/fusiondebug.

---

A veteran CFML developer since 1997, Charlie Arehart is a longtime contributor to the community and recently became a member of the Adobe Community Expert 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.