Taking a First Look at FusionDebug

The first interactive debugger for ColdFusion MX

By Jeffry Houser

I remember a particularly long weekend sitting in a computer lab for 12 hours and trying to write an assembler program on a VAX machine that would read and write files. (A VAX is a big archaic mainframe computer.)

It took many days, but I was finally able to provide successful results. Sometimes, I think we take for granted that ColdFusion makes our lives extremely easy. (Even a novice CF developer should be able to figure out how to read and write files in under five minutes.) One tool that was integral to my success in the project was the use of an interactive debugger. In my professional career, I've used one for writing Pascal and Lotus Notes, but, moving to the Web world, debuggers were noticeably vacant. That is until now. The folks who brought us Fusion Reactor have now brought us FusionDebug, the first interactive debugger for ColdFusion MX.

Why Do You Need an Interactive Debugger?

What is an interactive debugger? It is a tool that allows you to view the results of your code, line by line. You can see the value of variables, queries, CFC Instances, and change the values of variables on the fly. If you slip back into the old days of your memory, you probably remember a product called ColdFusion Studio, right? Before CFEclipse, ColdFusion Studio was the tool that (almost) everyone used for creating ColdFusion applications. A little known (and even lesser used) feature in CF Studio was the availability of an interactive debugger. Unfortunately, the product was hard to configure and even harder to use, so it never became a staple of the ColdFusion developer's toolbox. Today, the landscape of ColdFusion development is much different than it was in the CF Studio days. In the old days, most templates did multiple things and business logic code was implemented right next to display logic. There was little thought of code encapsulation, and templates were written the same way they were processed (start at the top and work your way down). Although CFML custom tags and User-Defined Functions provided facilities for encapsulating code, it wasn't until ColdFusion Components were introduced that people started to apply advanced programming principles to their ColdFusion applications. Business logic is now put inside of CFCs. Frameworks such as Model-Glue and Mach-II help us separate business logic and presentation code. When a single ColdFusion page loads, it may be performing actions where the code is located across multiple files. It isn't always easy to find the root of your errors. FusionDebug is a tool that will help you find and correct your troublesome code.

I thought a good place to start might be to offer a definition of debugging concepts and definitions:

- ***Breakpoint:*** A breakpoint is a spot in your code where you want the debugger to stop, and can be any line with CFML code (tags, variables, and so on). After starting a debug session in FusionDebug, you'll load the page you want to debug in the browser and FusionDebug will intercept it at your first breakpoint. (It can also intercept requests from pages and CFCs called from Flex, AJAX, Flash Remoting, and Web services.) If there are no breakpoints, the page will run as normal. Breakpoints do not have to be in the page you request, they can be located in CFCs, custom tags, includes, or UDFs.
- ***Variables:*** You already know what a variable is and how to use them. (If not, read "Creating Variables in CFML" from CFDJ, Vol. 6, issue 2 [http://coldfusion.sys-con.com/read/43790.htm]). The variable list of FusionDebug shows all of the variables currently available to the template you are in, at the point where you're debugging. It will display local variables (the variables scope), URL, Form, request, CGI, and cookie variables, as well as shared-scope variables like session and application, and more.
- ***Expressions:*** In the past I've always referred to these as "watch variables." Expressions, in its simplistic form, are variables that you want to keep tabs on. However, the expression pane of FusionDebug will support any ColdFusion expression. Perhaps you want to keep tabs on what the first item of a list is, or the last element in an array? You can do that with the Expression tab.
- ***Debug:*** The debug command means to starts a debugging session. In other software I've seen this described as "run" or "execute."
- ***Step Into:*** A step is how you move from one line of code to the next. There are three variations of the Step depending on what you want to do. Step Into is probably what you'll use

most of the time. It means to step over this line of code, but jump into any functions, custom tags, or CFC method calls that are called from the line of code you're on. FusionDebug will open the file that you step into and stop at the first line of CFML code.

- ***Step Over:*** Step Over is the exact opposite of Step Into. It will execute the code in any function, custom tag, or CFC Method without opening it. It moves right onto the next line. It's a good idea to favor Step Over even on tags and expressions that wouldn't open a file, for reasons I'll explain later.
- ***Step Return:*** Step Return is used when you're inside a file that you stepped into. It will execute the rest of the file, and put you back to the place where the file was initially stepped into
- ***Resume:*** Resume means to continue execution of code until you run out of code, or find another breakpoint.
- ***Terminate:*** Terminate will end the debugging session without completing your code's execution. Usually I try to avoid using this.

These definitions should give you a good understanding of what features the debugger offers. I'll move on to the FusionDebug install, and then an example.

Installing FusionDebug

FusionDebug is created as an eclipse plug in. and you'll need Eclipse to install it. If you're already using CFEclipse or FlexBuilder, you must already have Eclipse installed. If not, you can download Eclipse from http://www.eclipse.org/. It's as simple as extracting a zip file and running the Eclipse command. There's no installer.

Speaking of CFEclipse, which is another plug-in for editing CFML code, while you don't need to use it with FusionDebug, if you want to try it, I strongly recommend reading the ACME (Apache, ColdFusion, MySQL, Eclipse) guide for detailed instructions (http://www.stephencollins.org/acme/) on the install. Even if you don't use Apache or MySQL look at just the final chapter on CFEclipse. It's the best place to find detailed installation instructions on CFEclipse.

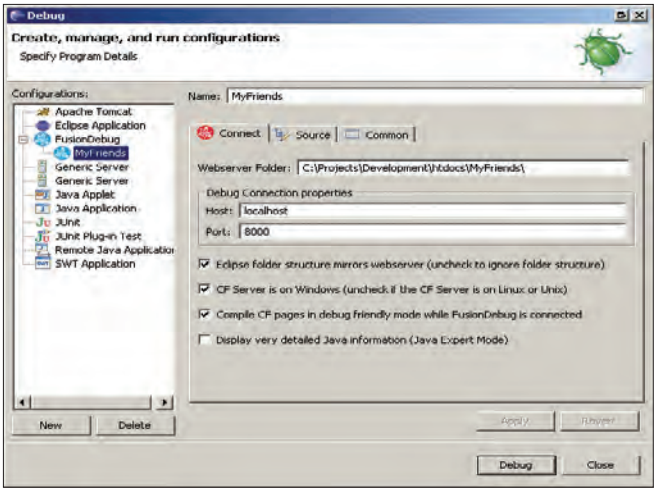

Figure 1: FusionDebug Configuration Window

There are two ways to install FusionDebug: using a manual process or an automated one. The automated process uses the "Find and Install Updates" feature built into Eclipse. This is documented in the FusionDebug user guide, which (at the time of this writing) is available at http://www.fusion-reactor.com/fusiondebug/helpDocs/FusionDebug_User_Guide.pdf. I'm going to step you through the process of the manual install, so while you're at it you may as well download the files for a manual install. The zip archive includes the PDF documentation at "plugins\com.intergral.fusionreactor.debug.core_1.0.0\FusionDebug – User manual.pdf". No matter which way you install, I recommend reading through the guide.

Plugins for Eclipse are located in a subdirectory of your Eclipse installation named plugins. You can unzip the download to the plugins directory if you wish. Restart Eclipse so the plugin is found. If you don't wish to extract into your Eclipse directory, unzip somewhere else, and copy the directories manually. There are two directories you need to put in "eclipseinstall\plugins": "com.intergral.fusionreactor.debug.core_1.0.0" and "com.intergral.fusionreactor.debug.ui_1.0.0". Restart Eclipse and you should be good to go. Yes, the install really is as easy as copying files.

Configuring ColdFusion and FusionDebug

Your next step is to set up ColdFusion to allow for debugging requests. In reality, you're setting up the Java server, which underlies your ColdFusion installation, and not making any direct changes to ColdFusion settings. You need to change the arguments that are used when launching the JVM. In most ColdFusion installations, you'll be using JRun, and the JVM settings are located in a jvm.config file. For Windows installations, this file is located in the CF install directory under "runtime/bin". For Unix, it's just in "bin" under your CF install directory. If you are using CF in any sort of multiserver configuration, your location may vary slightly. Before editing the file, you may want to save a copy of the current file, in case anything goes wrong with the one-line edit I'm about to describe.

Find the "java.args" line of the config file and remove this argument:

```
-XX:+UseParallelGC
```

Then add these options:

```
-Djava.compiler=NONE
-Xnoagent
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

Make sure that all the options are on the same line. For ease, you can copy these lines out of the FusionDebug documentation. Note the port 8000 that's used. If you know that port to be used is already in your system, choose a number that's not in use. I'll explain its use shortly. Save the file and restart your ColdFusion services. If it fails to start, revert to the saved copy from above and study your changes to make sure you followed the instructions carefully.

The final step in getting yourself set up is to create an

instance of FusionDebug that points to the server you set up for debugging:

1. Load up Eclipse, and select a project you want to debug with. (If you don't have any Eclipse projects, see the FusionDebug User Guide for more information on creating one for the first time.) Select "Debug" from the Run menu.
2. Select FusionDebug from the menu and Click New to create a new FusionDebug instance. This will bring up a dialog similar to Figure 1.
3. Enter a name for the FusionDebug instance. This can be anything you want it to be. Specify the Web server folder. I used the same folder as the project root for my Eclipse Project.
4. Specify the host and port. Note that the hostname can be localhost or indeed any server you have access to, which has been configured as per the steps above. Yes, FusionDebug can debug remote servers. The port is not your Web server port, but the port you specified in the Java arguments. It is 8000 by default. If you followed my instructions above, that didn't change.
5. Select your other options. In my case, my CF Server is on Windows, I want CF to compile pages in debug friendly mode, and my Eclipse folder mirrors the Web server. I have not yet experimented with the "Detailed Java Information" option, soI left that unchecked. These are explained in the User Guide.

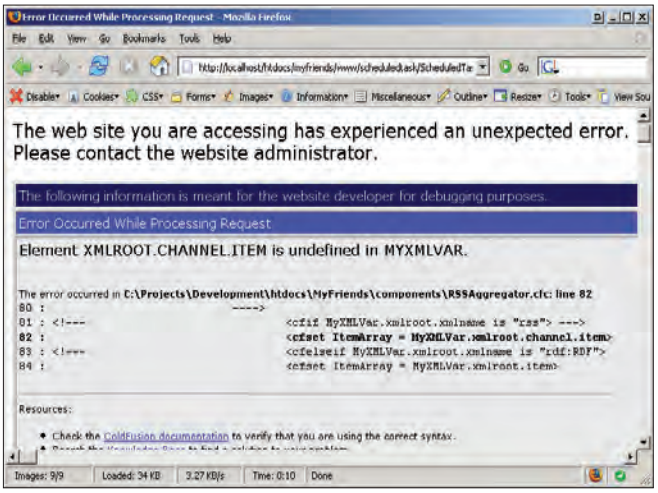

Figure 2: MyFriend error

Now you are good to go, ready to debug.

You're First Debugging Session

If you've read my two articles on creating an RSS Aggregator, you probably noticed that I found a bug between part one and part two. The code I wrote in part 1 (http://coldfusion.sys-con.com/read/235976.htm) supported only RSS 2. It turns out that the Macromedia XML News Aggregator (http://weblogs.macromedia.com/mxna/) was only providing an RSS 1.0 feed. The XML was different in each feed. In part 2 (http://coldfusion.sys-con.com/read/264745.htm), I described the error and explained the fix. I thought I'd step you through the process I used to diagnose and fix the problem (without FusionDebug) and then step you through the process using FusionDebug.

In my test database for the MyFriend RSS aggregator, I have two RSS URLs. One points to my blog and the other to MXNA. When trying to process the MXNA URL (by manually loading the scheduled task), I was seeing an error, as shown in Figure 2. I wasn't sure why I was receiving the error, since all RSS feeds should contain an item as part of the channel (right?). The first thing I did was add a cfdump tag to see why "xmlroot.channel.item" did not exist. I reloaded and got nothing [insert two head scratches and one look of utter confusion here]. Why wasn't I seeing any debug output? It turns out the CFFUNCTION had its OUTPUT attribute set to false. No output means no output and the cfdumps weren't being sent back to the browser. I changed that attribute to true and reloaded. Okay, now with the cfdump I could drill down into the XML and see the problem. Items (AKA Blog Posts) are stored differently in RSS 1 vs RSS 2. And the root elements are different.

Where would the debugger have helped? To debug this code inside a component, I'm changing code. It's one thing to add a variable output or cfdumps so you can view variables. Those are easy enough to comment out later. It's quite another to have to change code, such as the output parameter from false to true. I don't want to have different code for "production" versus "development." That defeats the purpose and allows too much margin for error. How many of these output attributes have I forgotten to change back? I have no idea. The problem gets worse when you are dealing with nested components, because you have trickle back up to the top of the tree and set all outputs to true. Still another situation occurs with code within CFSILENT, which I would have to find and remove to see any debugging output. These are situations where an interactive debugger offers benefit. To get to the root of the problem I don't have to change any code.

This is how I'll debug this in FusionDebug:

1. First, I'm going to open up the project in Eclipse, and switch to the debug perspective. You can switch perspectives using "Window --> Open Perspective and selecting "Debug." You should see something similar to Figure 3. (Again, all this is described very well in the User Guide, for those new to Eclipse.)
2. Next, start a debugging session. You can re-open the debug window (see Figure 2), select your Debug instance, and select "Debug," or you can start the debug session by clicking the "debug" button from the toolbar. You should see the session started up in the debug window (top left).
3. Open up a page in the project that you want to debug, and set a breakpoint. (Note that if you open a file from the file system rather than from a project, the debugger won't enable you to step through code.) With MyFriend, I am opening up the scheduled task file (schedultedtask/scheduledtask.cfm) and putting a breakpoint on the first CF line of the page. You can add a breakpoint by selecting the line

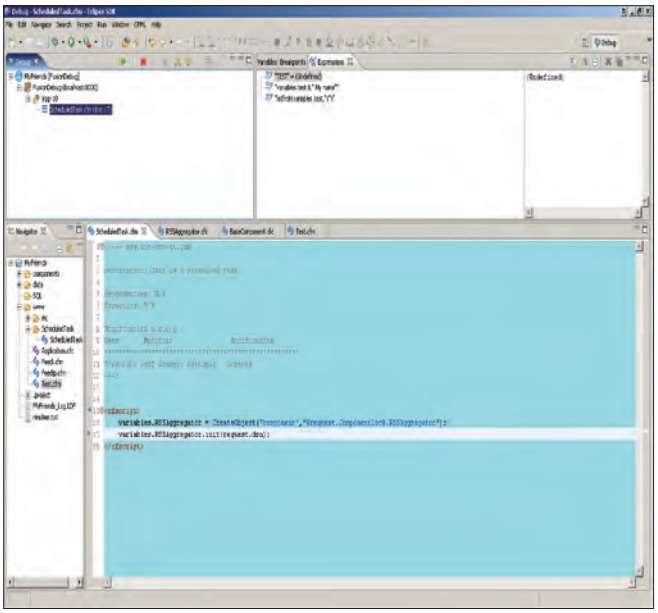

Figure 3: FusionDebug Perspective

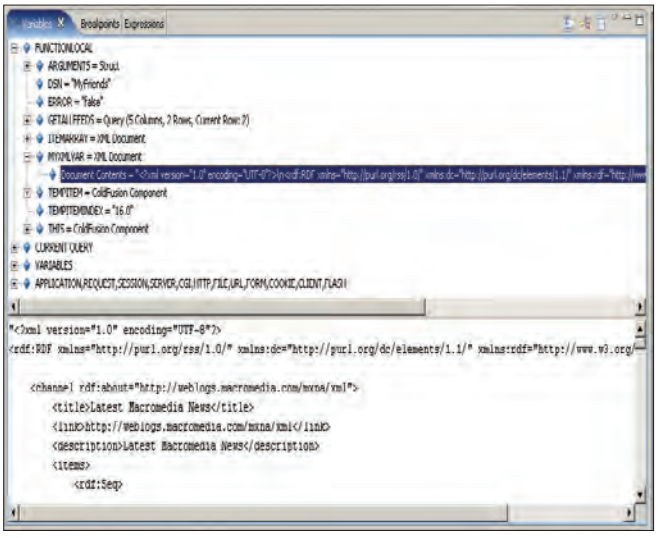

Figure 4: Variables Panel

and choosing "Window --> Toggle Line Breakpoint" or using Control + Shift + B, or right-click the line and choose "Toggle Breakpoint." You'll now see the breakpoint was set because of the blue dot in the grey bar to the left of the line number. (You can see the dot on line 15 of Figure 3).

4. Open your favorite browser and launch the page you want to debug. FusionDebug will intercept the page at the first breakpoint. Again, note that you're not opening the page "in" the debugger but in whatever browser you want. There is a blue arrow in the same grey bar to show you which template you are currently running. (This is called the "Current Instruction Pointer.")
5. Click Step Over and you can step through the initialization code to watch the component being created. Continue to click Step Over until you reach the init method of the RSSAggregator component. Around line 75 you should see code like this:

```
<cfset MyXMLVar = xmlparse(cfhttp.filecontent)>
```

6. Click past the line that parses the XML content. Now take a look at the variables tab shown in Figure 4. It should be in the upper-right corner of the screen. You can view all the variables available to the component. The one we care about most is the local function variable that holds the XML. The first time through the loop, you'll see that the XML is in proper RSS 2 format. The second time through you'll notice it uses "RDF" instead of XML. Drilling down, you'll realize that the items are different in the RDF format than they are in the RSS format. And that is the root of the problem.

The bug was easily solved by adding an if statement (explained in the previous article) to decide which RSS format to use, and how to access the item array. Both debugging methods can bring you to the same end result, but I personally find the use of FusionDebug to be much more elegant.

I mentioned near the start to favor Step Over versus Step Into, even on tags or function calls that don't open a file. The reason is that being a Java debugger, Eclipse will try to step through the underlying Java code. FusionDebug is configured to hide that by default, but the execution of the tag/function will take longer than if you'd used "Step Over."

What Next

The folks who created FusionDebug created some Captivate videos to help demonstrate what FusionDebug can do and some uses of it. It is one thing to read about how to do things, it is another to see it done, so I suggest checking them out at http://www.fusion-reactor.com/fusiondebug/gettingStarted.html. Charlie Arehart wrote a tips and tricks article for this edition of ColdFusion Developer's Journal. I've had a chance to preview it and it contains a lot of information beyond the scope of this article.

The product is available as a free 20-day trial. Pricing starts at US$299, with an 10% discount currently available with the code CFCOMMUNITY. Volume discounts are also available. The company offers free support, at support@fusion-reactor.com.

The RSSAggregator code is downloadable from my blog at www.jeffryhouser.com. I appreciate all the feedback I've received from it. Thanks for reading, and let me know what you think. I'll see you next month.

About the Author

Jeffry Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company and has authored three separate books on ColdFusion, most recently ColdFusion MX: The Complete Reference (McGraw-Hill Osborne Media).

jeff@instantcoldfusion.com