

CFDEBUGGER

Tracing code execution in BlueDragon

Have you ever wanted to see what lines of code are being executed in your template? If you looked for CFTRACE to do that, you were likely disappointed. The CFDEBUGGER tag in BlueDragon, however, can be a great tool for debugging.

When CFML developers need to figure out what their code is doing, in terms of what lines of code their template is executing, they have surprisingly few options. There are other debugging tools, but they address different needs.

There's the server debugging info, but that's all after the fact, simply showing the state of variables at the end of the program (and/or in some cases their values when passed into it). And many rely on a trusty combination of CFABORT and CFDUMP to help identify control flow issues and other debugging challenges.

There's also the new CFTRACE tag introduced in CFMX and the older CFLOG tag from CF 4; each offers interesting variants on enabling you to track the value of some variables or log some string at a point in time. But neither of these can be used to simply show what lines of code have been executed in a run of a template. And the Java-based stack trace doesn't solve this problem either.

The closest we had was the interactive debugging tool of CF Studio that worked with CF 5, but is no longer supported in CFMX. Even then, many found the interactive debugger to be lacking and often not usable.

If you simply wanted to get some listing of all the lines of code that were executed in a template (what some really would call a "trace"), there's currently no solution in CFML unless you use BlueDragon. In this month's **Blueprints** column I'd like to introduce this feature



By Charlie Arehart

that could be useful to all CFML developers.

CFDEBUGGER: It's There If You Need It

<CFDEBUGGER> is a tag that works in BlueDragon only, and it simply writes an indication of each CFML line of code that's been executed to a log file. That's something many

have long wished for. There are times when this is just the ticket to solve a challenging debugging problem. Here's a simple example of its use. Let's say you have a template that just does this:

```
<CFDEBUGGER LOGFILE="trace.log">
<cfset name="bob">
```

Note that no closing CFDEBUGGER tag is needed (as will be explained later). This will create an entry in a file named trace.log (as indicated in the LOGFILE attribute) with the following info:

```
#0: CFDEBUGGER trace started @ 19/Aug/2003
15:03.19
#1: active.file=C:/inetpub/wwwroot/regression/
cfdebugger.cfm
#2: tag.end=CFDEBUGGER; L/C=(1,1); File=C:/
inetpub/wwwroot/regression/cfdebugger.cfm
#3: tag.start=CFSET; L/C=(2,1); File=C:/inetpub/
wwwroot/regression/cfdebugger.cfm
#4: tag.end=CFSET; L/C=(2,1); File=C:/inetpub/
wwwroot/regression/cfdebugger.cfm
#5: file.end=C:/inetpub/wwwroot/regression/cfde-
bugger.cfm
#6: Session Ended
```

Note that it indicates the time the template was run and the template's name, which is useful because the log could hold lots of such "trace sessions," for reasons I'll explain in a moment. More important, for each CFML tag it encounters, the trace shows its start and end lines in the given template. Unlike looking in an editor, this log shows only lines with CFML tags that were executed, not all lines in the entire file.

And while the CFML error message (in both BlueDragon and CF) reports the line of code on which an error occurs, what if the problem is a logic error that's not generating any CFML errors?

This capability could be useful for debugging a knotty problem where you can't tell which lines of code are being executed. If this was a several-hundred-line program doing loops and ifs, it could help you quickly narrow down just where the flow of control was going if you were having trouble figuring it out. Sometimes even well-placed CFOUTPUTs, CFDUMPs, and CFABORTs just don't do the trick.

I'll grant that it could produce a lot of data to analyze (and sadly it's not in a format that's easily conducive to performing automated analysis), but there will be times when it's better than nothing.

And it's something that CF has never had. Did you perhaps think that this was what CFTRACE would or should have done? It doesn't. CFTRACE (introduced in CFMX) just writes the value of a given variable or string to either a log file, the debugging output, or the screen. It's just a little more useful than CFLOG or CFDUMP. (BlueDragon currently supports CFLOG and will eventually support CFTRACE.)

So CFDEBUGGER could be very handy when you need it. Give it a try. Just be careful not to leave it on lest it create humongous log files.

Here are some additional notes about the use of CFDEBUGGER.

Where Is the Trace File Stored?

In the example I showed, I didn't name the path for the location to the log file. Where does it go then?

In the Server editions of BlueDragon, it's stored in the BlueDragon install directory. For instance, if you're using BlueDragon Server JX and it's installed in C:\Program Files\New Atlanta\BlueDragon_Server_JX_3.1, that's where you'll find the log file.

In the J2EE edition, the actual location will depend on the J2EE app server you're using. I've not found any logic to explain where the server decides to put it. It's neither the application's context root nor the install directory for the app server. For instance, in my WebLogic 8.1 test, it was found in C:\bea\user_projects\domains\mydomain. In JBoss, I found it in C:\jboss-3.2.1_tomcat-4.1.24\bin. In JRun, it was in my C:\JRun4\bin. If you have another server, just do a search within that server's directory for whatever you named the file (you're free to call it what you want), and make note of that location for future reference.

Can I Provide a Full Path for the Log File?

Rather than wondering where it may be stored, you can also just name the full path to the intended directory for placing the file. For example, using LOGFILE="c:\trace.log" works as expected, and in both the Server and J2EE edition.

What Happens If You Do a CFINCLUDE or Custom Tag Call While Tracing?

No problem. The trace log will continue to log all the activity (hopefully that's what you wanted). The log file does indicate when you switch to running a new template. At the point of the cfinclude, you'd see an entry like this:

```
#6: file.start=C:/inetpub/wwwroot/includedfile.cfm
```

where in this example my calling template included a file called includedfile.cfm. That's useful. The same is true when calling a custom tag.

When Does Tracing Begin?

The tracing begins at the point that the CFDEBUGGER tag appears. Therefore, it will *not* trace activity in the application.cfm that occurs before you start tracing. Naturally, if you place the CFDEBUGGER tag in the application.cfm, then indeed it will do the tracing from that point forward (as I just said above, the tracing does remain in effect across includes and custom tags, and application.cfm is executed like an implicit cfinclude before any template runs, so the same logic applies).

When Does the Tracing Stop?

Tracing ends at request termination (of course, if you have an onrequestend.cfm, it will trace that as well). As will be discussed, while tracing stops at the end of request execution, the log file is not emptied upon each new request.

Is There a Way to Turn Off Tracing?

In other words, if a calling/including template or the application.cfm starts the tracing, is there a way to stop it in the called/included/requested program? No, currently there is not. And again, there is currently no available closing CFDEBUGGER tag to surround code to be traced. If you don't want to do the tracing, you should just remove the CFDEBUGGER tag.

If a template is running under the influence of another that turns tracing on within the current request, you'll just need to remove the CFDEBUGGER from that controlling template if you don't want it to trace any that it includes/calls/requests.

You might wonder if you could solve the problem another way – perhaps by leaving the LOGFILE attribute value empty or leaving it off entirely, hoping it will turn the logging off. That doesn't work and will instead get an error. I suppose in Linux boxes you could redirect the tracing to the null device, though.

Again there's no way to stop tracing within a request once it's started for a given request, short of removing the CFDEBUGGER tag.

Is a New Trace File Created on Each Request?

No, and this is very important. The trace files are cumulative. They are appended to by each request that writes to them. Beware, therefore, as they can grow quite large with successive executions. Don't forget to remove the CFDEBUGGER when no longer doing tracing. There is currently no attribute to cause the tag to erase existing log entries or otherwise manage the log file size. (It does create one if it doesn't exist the first time you call it.)

What Happens When a New CFDEBUGGER Is Encountered Naming a New Log File?

In other words, what happens if one template (such as application.cfm or a calling/including template) has turned on the tracing to one file, and another template (or even later code in the same template) offers a new CFDEBUGGER naming a new log file? Tracing simply stops on the first log file and continues with the newly named one. That could be useful (or it could be a hassle in some instances, but that's the way it currently works).

Does CFDEBUGGER Trace Functions As Well? Or Just Tags?

It currently traces just tags, not references to expressions within tags. For instance, consider the following code:

```
<CFDEBUGGER LOGFILE="trace.log">
<cfoutput
this is a test
#cgi.request_method#
another line
#now()#
</cfoutput>
```

It might be nice if it reported encountering the variable and function references on the 4th and 6th lines. It does not, reporting only on the opening and closing CFOUTPUT tag. The same would apply to expression references within CFMAIL, etc., as well as statements within CFSCRIPT. These are not individually traced.


(There's a TagsOnly attribute available for the tag, designed to take a yes/no value, which was perhaps intended to solve this problem, but it's not currently implemented.) It's not really a big deal if you think about it, at least with respect to CFOUTPUT and CFMAIL. The value of detecting the flow of control within these tags would seem somewhat diminished.

One last question that some may wonder about: Is it risky for BlueDragon to add a tag like CFDEBUGGER to CFML? Does it make your code incompatible? Look at it this way: you wouldn't leave the CFDEBUGGER tag in a production CFML application. You'll only use it for debugging purposes, so there's really no risk to your using it and therefore our adding it.

Some Possible Improvements

I have alluded to some challenges of using the CFDEBUGGER tag. It's not perfect. While it's useful now, perhaps as more people take advantage of this nifty tag, New Atlanta can consider improving it. Here are some possible improvements I'd look forward to:

- Change the log output format into a CSV file or the like (at least, give us the option) so that New Atlanta could then perform analysis on the output. For instance, it would only be a short step to then be able to produce a report indicating which lines of code were executed the most.
- Write out the time taken to execute each tag. Again, this could then feed into an analysis reporting not by frequency of execution but by total time per line of code. That could be really cool.
- Offer a means to turn off tracing, in case it was enabled in a calling/including template or application.cfm and you don't want it continuing in an included/called/requested template.
- Cause it to create a new log file on each new request (rather than growing unchecked), or a way to indicate a maximum number of requests to keep or a maximum size the log file should grow, or perhaps an attribute to indicate that the log should be wiped clean before being written to for the current request.
- Have the tagonly work as it would seem it should, letting it trace the lines containing variable and function references.
- Add an attribute that indicates if it should only create its tracing output if the debugging option is turned on in the Admin. That way it could be left in place in production and not have any effect unless debugging is on, or you run it while debugging is enabled for your IP. CFTRACE works this way in CFMX, and our new CFASSERT tag that we've added to the latest release of BlueDragon works similarly (more on that in a later article). Supporting debugging tags to run only when debugging/testing is on is a good model. Then again, some may especially want it tracing while folks are running the template in production. That's why I think it should be controllable as an attribute.

But, for now, enjoy what it does offer, which is a unique way to trace the lines of code being executed in a CFML template. 

About the Author

Charlie Arehart is a Macromedia Certified Advanced ColdFusion developer and trainer. He has recently become CTO of New Atlanta Communications, makers of BlueDragon. In his new role, he will continue to support the CFML community, contributing to several CF resources, and speaking frequently at user groups throughout the country.

charlie@newatlanta.com