CHAPTER 45

Using the Debugger

IN THIS CHAPTER

ColdFusion Builder offers a powerful new developer tool in its debugger, an interactive step debugger that allows CFML developers to walk through the execution of their code, observing what lines of code it executes, what other CFML files (include files, custom tags, or CFCs) it executes, the value of all variables (optionally in all scopes) at a given point during execution, and much more. You can even change the value of variables on the fly within the debugger, such as to alter the runtime flow of execution of your code.

More than just watching the code that you run yourself, the debugger can watch the execution of any request (including requests from other users running code on your server), and also any CFML however it's requested, including calls from Ajax or Flex clients or from other servers issuing Web service requests or CFHTTP equivalents.

There are also other kinds of requests that can happen within ColdFusion through no traditional browser: scheduled tasks, requests made via the ColdFusion gateways, and events in the `Application.cfc` file, such as `onSessionEnd` and `onApplicationEnd`. With all these kinds of requests, there often is no browser to which you can easily send back traditional debugging statements.

And even with traditional browser requests, there are still times when CFML code can't display any output, such as when output has been disabled in a CFC or method, or within a `CFSILENT` tag pair, and so on.

These are just some of the scenarios in which a debugger becomes so valuable, where you use a tool to watch the execution of some code while it runs live, regardless of how it was requested by a user. Of course, there is all the requisite security to ensure that the debugger is used only by authorized developers.

The debugger is built into ColdFusion Builder, which has been introduced in other chapters in this series, especially Chapter 3, "Introducing ColdFusion Builder," in *Adobe ColdFusion 9 Web Application Construction Kit, Volume 1: Getting Started*. Whether you're new to debugging or feel lukewarm about the concept, this chapter will help you feel comfortable and confident using the debugger as a

tool in your CFML developer toolkit. It will introduce debugging and the debugger, how to install and configure it, and how to use it.

The initial discussion of understanding, installing, and configuring the debugger (and ColdFusion) may seem a bit tedious, but it's generally a one-time effort. Once you configure the debugger, you will be able to use it quite easily and quickly.

Overview

We start with an overview of traditional and step debugging.

Traditional Forms of Debugging

As developers, we spend most of our day debugging code: whether it's new code we're writing, old code we're updating, or someone else's code we've inherited. When trying to resolve a problem in a given CFML template or CFC, you often need to understand the flow and function of the code, and to do that you have a couple choices.

First, you can try to just statically review the source code, relying on your skills at reading and interpreting the code to determine what it should be doing, eye-balling potentially many lines of code, and guessing at or tracking manually what the values of various variables will be at any given point. This can be a complicated effort, but some developers are especially well suited to the task.

More commonly we instead let ColdFusion just run the page and then we use either of two techniques to have it report what's going on in the execution of the code. The ColdFusion debugging output, which is displayed at the end of a page request (if enabled in the ColdFusion Administrator), can show high-level information about that request, including how long it took to run, what files it called (includes, custom tags, CFCs), what queries it executed (their duration, record count, SQL, and such), the variables and values in many variable scopes, and much more. This feature was discussed in Chapter 17, "Debugging and Troubleshooting," in Volume 1.

Still, there are times when either that level of detail is not enough or debugging output is not enabled (and/or cannot be). In that case most developers will resort to placing `CFOUTPUT` and `CFDUMP` tags strategically in the program code both to depict the flow of control ("What lines of code am I running?") and to determine the value of variables at a given point. This is a tried and true method, and for many developers they've known no other alternative.

Introducing Step Debugging

Developers who come from some other programming environments, however, are often surprised to find CFML developers having only these techniques at their disposal. Such developers have often had the advantage of using a step debugging tool to help them in this very sort of situation. With a step debugger, a developer can actually observe the lines of code as they are executed, watching each line begin and end, as well as watching the value of any and all variables in play at that point in the code.

Consider the code fragment in Figure 45.1 (from `getfilms.cfm` in the source code directory for this chapter), which depicts some code about to be executed. As we observe the code, we can wonder

whether the first "if" test will be true, and we may wonder what goes on in the `listFilms` method that would be called. If we knew the value of the variables being tested, then of course we'd know which test would be true.

Figure 45.1
Sample CFML code being debugged.

A step debugger could tell us all that information, and more. We could tell the debugger to stop execution at any given line of code, and while there we could view all the currently defined variables (in any of the scopes we've configured the debugger to show). More important, we could ask the debugger to step through the code, running whatever the next line would be (line 14), and then step again, in which case we'd run either line 15 or 16, depending on the condition evaluated. No more guessing!

Beyond that, while observing this code (without a debugger) we may wonder other things: how did we get to this point in the code? Was this file called as an include file, custom tag, or CFC method? Did we run an `Application.cfm` or `Application.cfc` file before getting here?

And of course we may even wonder about the `listFilms` function itself: What does it do, where is it defined, and so on. The great news is that we could ask the debugger to step into that method, and since it's a CFC method in another file, the debugger would find the file, open it, and stop on the first line of code inside that file. The same would be true if we stepped into included files, calls to custom tags, and so on.

All this and more can be discovered with the ColdFusion Builder debugger. Let's look next at how configure it; then we'll see how to use the debugger to answer the questions just posed.

Configuring ColdFusion and the Debugger

To use the debugger against a given implementation of ColdFusion (8 or 9), you must configure that ColdFusion server to permit debugging, and then you must configure ColdFusion Builder itself. We'll cover each of these topics in this section.

NOTE

The debugger works only with ColdFusion 8 or 9. If you're interested in debugging ColdFusion 6 or 7 servers, look into FusionDebug, a commercial debugger that functions similarly to the ColdFusion Builder debugger and can debug ColdFusion 6 or higher. For more information, see `www.fusiondebug.com`.

Configuring the ColdFusion Administrator

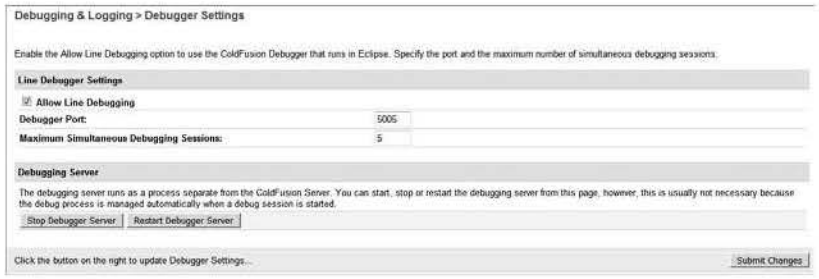
You enable a server to permit debugging by way of settings in the ColdFusion Administrator. The setting for the ColdFusion debugger feature is separate from the traditional debugging output feature, which is also enabled in the Administrator. There are also important settings on other Administrator screens.

Enable Debugger Settings

The first and most important settings to enable for debugging are on a page in the Debugging & Logging section of the Administrator, in a page called Debugger Settings (Figure 45.2).

There, you must select the Allow Line Debugging option and click Submit Changes to enable debugging against this server. Note that you must restart the server upon enabling (or disabling) this setting for the it to take effect. (More on this is offered later, in the section "Special Considerations for Multiserver Deployment.")

Figure 45.2
ColdFusion Administrator Debugger Settings page.



While on that Debugger Settings page, notice the available Debugger Port setting, which you need not change unless the port is one that's unavailable on your server.

WARNING

If you're debugging a server that is remote to you, beware of the following issue. The debugging feature on the server actually listens for commands (from ColdFusion Builder) on a port separate from the one specified in the aforementioned setting. For the sake of security, ColdFusion will by default cause the debugger feature to use a randomly available port. This would be a problem if ColdFusion is behind a firewall and you're trying to debug it from outside that firewall, since the firewall will block that random port.

One solution is to modify ColdFusion's Java configuration arguments to specify a fixed debugger server port number (different from that mentioned earlier), and then modify the firewall to allow access to this port from your (or any authorized developer's) machine.

To set a fixed debugger port number, specify the following JVM argument on the Java And JVM page of the ColdFusion Administrator (in ColdFusion Standard or Server deployments) or in `jvm.config` for ColdFusion Multiserver deployments:

```
-DDEBUGGER_SERVER_PORT=portNumber
```

Replace `portNumber` with the port that you wish to use, and restart ColdFusion. Be careful when modifying the Java configuration arguments, as incorrect values could prevent ColdFusion from starting. Be sure to make a copy of the `jvm.config` file before making these changes.

The next setting on the Debugger Settings page is Maximum Simultaneous Debugging Sessions, which need not be changed unless you expect to have more than the default number of developers (five) debugging against a single shared server.

I'll explain the Start Debugger Server button at the end of the chapter, in the section "Stopping the Debugger." For now, just know that you don't need to click the button to start debugging. The first debugging request will cause debugging to start on the server. (And note that when debugging has started, the button will change to Stop Debugger, and a Restart Debugger button will appear next to it.)

Configuring RDS to Secure the Server

Once the Allow Debugging option is enabled, which permits a server to be debugged, the next significant step is controlling who may debug code on the server. Adobe chose to build debugger security on top of the long-standing Remote Developer Services (RDS) feature, to provide secured access for developers using editors (including Dreamweaver and ColdFusion Builder) trying to access information from the ColdFusion server.

For a ColdFusion server to be debugged, the developer enabling debugging from ColdFusion Builder will need to provide credentials as indicated in the RDS settings of the server. The section "Configuring ColdFusion Builder" later in this chapter explains how to use these RDS credentials in ColdFusion Builder.

The ColdFusion server administrator controls whether RDS is enabled (it can be disabled at installation) and also chooses the credentials (password, if any) that will be required by developers, as configured on the Administrator's Security > RDS page.

Note that ColdFusion Enterprise and Developer editions (since ColdFusion 8) add a powerful RDS feature, allowing multiple user accounts, so that different developers can each be given their own RDS username/password pair authorizing them to access the server from development tools such as ColdFusion Builder and the debugger. RDS security is discussed further in Chapter 56, "Security in Shared and Hosted Environments," in *Adobe ColdFusion 9 Web Application Construction Kit, Volume 3: Advanced Application Development*.

Increase Maximum Request Timeouts

While debuggers are powerful in enabling you to stop on a line of code and inspect variables and so on, there is a downside. ColdFusion is often configured to forcefully timeout requests that take too long, as configured in the Administrator's Server Settings > Settings page. If the Timeout Requests after (seconds) option is checked, then ColdFusion will try to terminate requests lasting longer than the specified time limit. When debugging code on a server using the step debugger, you will need to either increase the time limit or disable this feature entirely (in which case requests are free to run as long as they need to, which may be acceptable on a developer workstation).

Special Considerations for Multiserver Deployment

A final discussion regarding ColdFusion server configuration for the debugger revolves around a difference between installing ColdFusion in the stand-alone (Server) mode versus the Multiserver (or multi-instance) or J2EE deployment mode. In the former, when you enable the aforementioned setting, Allow Line Debugging, ColdFusion will automatically modify the underlying `jvm.config` file used to control the server, placing there the appropriate information needed to enable debugging.

In the Multiserver or J2EE modes, however, you must manually place the following information into the `jvm.config` file on the existing `java.args` line provided there:

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=#portNum#
```

Here, `#portnum#` is the port number entered in the Debugger Port setting, discussed earlier. That information (and the rest of the `java.args` line) must be entered on a single line, without line breaks.

WARNING

Be sure to make a copy of the `jvm.config` file before editing it. Any mistake in the file could prevent ColdFusion restarting.

Yet another concern when you run the Multiserver edition of ColdFusion is that by default all the instances share a single `jvm.config` file. This is a problem when using the debugger, as only one instance at a time can use the debugger port specified above. You may find that at times when an instance goes down, the lock on the debugger port remains tied to one or the other instances. In this case, you should explore giving each instance its own `jvm.config` file, with its own designation of the debugger port. This can be done most easily by starting a ColdFusion instance from the command line, which may be especially suitable in a development environment. You can also configure an instance that is started as a Microsoft Windows service so that different instances use different configuration files. Each approach is discussed in the Adobe technote at `http://www.adobe.com/go/tn_18206`.

Upon configuration of ColdFusion, you're ready to proceed to configuring ColdFusion Builder.

Configuring ColdFusion Builder

To connect a developer's ColdFusion Builder installation to a ColdFusion server, you need to configure a few settings in the ColdFusion Builder environment, including settings to define a project, define a server connection, configure an RDS connection, and possibly add debug mappings and debugger settings, and then you need to define a ColdFusion Builder debugging configuration for ColdFusion.

NOTE

As discussed in Chapter 3, ColdFusion Builder can be installed both in a stand-alone mode and as a plug-in on an existing Eclipse installation. While debugging can be performed in either, this chapter describes steps and shows screenshots for the stand-alone approach.

Define a ColdFusion Builder Project

For you to debug a CFML template, it must exist within a ColdFusion Builder project. If you're new to Eclipse-based IDEs like ColdFusion Builder, you may not be familiar with or may not yet have defined a project. Project creation is discussed in Chapter 3, as well as in the ColdFusion Builder Help section "Managing Projects."

Define a Server Connection

The template needs to exist within a project, and in addition that project must be defined with a server connection to the ColdFusion server. You may be using projects without having defined a server connection yet. How to create a server connection is discussed in Chapter 3, as well as in the ColdFusion Builder Help section "Adding ColdFusion Servers," which discusses how to create a connection to both local and remote servers.

The server connection is also where you will specify the RDS connection and authentication information as discussed in the first section of this chapter. For additional help on configuring and testing an RDS connection, see the ColdFusion Builder Help section "ColdFusion Builder Development Perspective."

Configure Debug Mappings If Needed

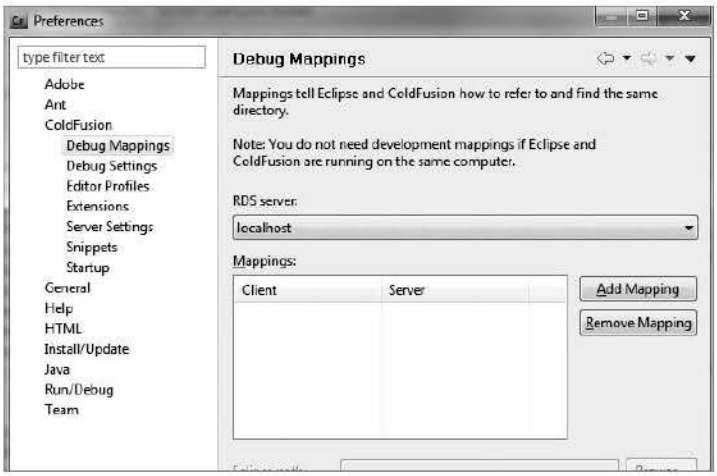
If ColdFusion and ColdFusion Builder are running on the same computer, you can skip the Mappings setting (Window > Preferences > ColdFusion > Debug Mappings). Otherwise, use this setting to map a connection between the path to files as ColdFusion Builder will see them (on your local machine typically) and the path to the same files as they will be found by the ColdFusion server. For instance, if you edit and test files locally in `C:\inetpub\wwwroot\someapp\` but later upload them to a remote or central server where you'd like to debug them, and they're identified on that server as being in `D:\webs\someapp`, then these would be the two values you'd specify on this page for the client and server mappings.

NOTE

Both paths are case-sensitive. There will be no error if the incorrect case is used here, but when you later attempt to debug against the given remote server, breakpoints simply won't fire.

To add a new set of mappings, first choose the server against which you will be debugging (in the RDS Server list, populated per the previous steps described above), then click Add Mapping and follow the instructions to fill in the Eclipse and ColdFusion paths, and click Apply to save the settings. See Figure 45.3. (Again, though, if you're running ColdFusion Builder and ColdFusion on the same computer, you don't need to add a debug mapping.)

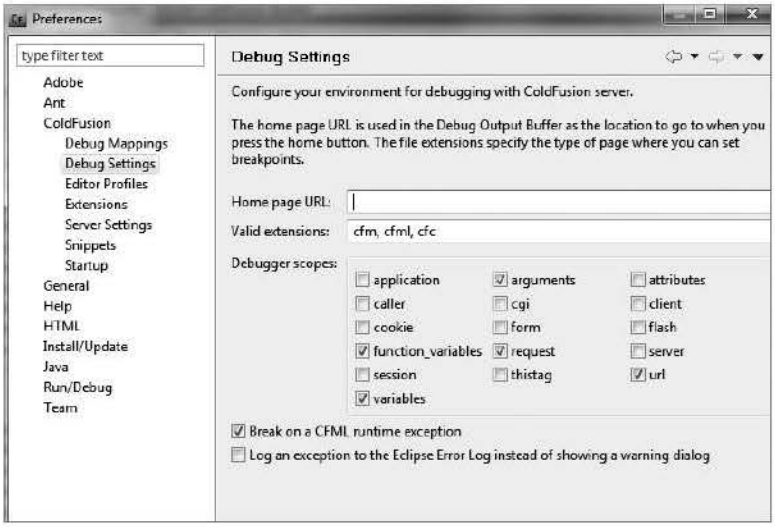
Figure 45.3
ColdFusion Builder Debug Mappings page.



Configure ColdFusion Builder Debug Settings

You specify the next segment of ColdFusion Builder debug settings on the Debug Settings page at Window > Preferences > ColdFusion > Debug Settings. There are four settings configurable on the page, with the final two being most important (Figure 45.4).

Figure 45.4
ColdFusion Builder Debug Settings page.



First, you can specify (if desired) a home page URL, which will be used to point to the page that appears in the Browser pane of the Debug Output Buffer of the debugger (a feature discussed later) when you click the Home button there (not something you'll likely do that often). Also, you can modify the extensions of the types of files that you want to debug.

More important for most is the Debugger Scopes setting, where you can specify which variable scopes you want the debugger to display when you're displaying variables (discussed later). Though it's tempting to select many scopes to see all possible information within the debugger, each will add more overhead to (and slow execution of) the debugger.

TIP

Here's an important tip: you can always view any variable in any scope using the Add Watch Expression feature, discussed later.

The fourth setting on this page is Break on a CFML Runtime Exception. With this feature enabled, the debugger will intercept an error occurring in any template within a project you are actively debugging (see the next section, "Manage Debug Configurations"). The error message will appear in an alert window, and the debugger will stop at the line that caused the error. This is a powerful feature since it precludes your needing to anticipate where code in a program might cause an error.

Remember also that because the debugger can intercept any request, you could be notified about errors happening to other users executing code in a project you are debugging. Note also that once the debugger has intercepted a request, it will not intercept another until you have completed step debugging of the first request.

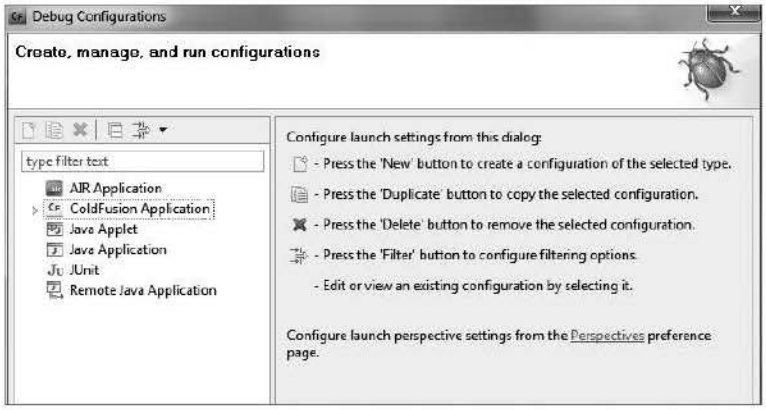
The final option is Log an Exception to the Eclipse Error Log Instead of Showing a Warning Dialog. With this option enabled, the Break on Exception feature will not display the pop-up message, although it will still intercept the request triggering the error. Instead, the error will be logged to the Eclipse Error Log (located in the `.metadata\.log` file in your Eclipse workspace).

Manage Debug Configurations

The final step in being able to debug ColdFusion code is to create a new ColdFusion Builder debugging configuration for ColdFusion, which simply indicates for a given project the ColdFusion server to which you want to connect (as defined in the "Define a Server Connection" section earlier in this chapter). In ColdFusion Builder, choose the menu command Run > Debug Configurations, which opens a window for creating, managing, and running debugging configurations (Figure 45.5).

Figure 45.5

ColdFusion Builder Debug Configurations page.

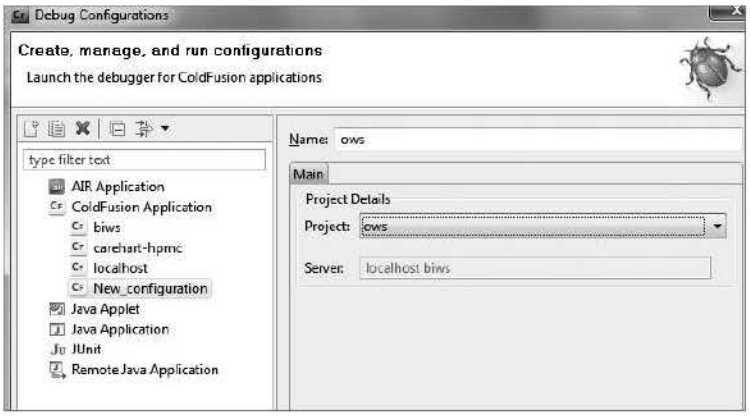


Here you could define debug configurations for different ColdFusion applications and projects. To create a new ColdFusion configuration, double-click the ColdFusion Application option, or single-click the ColdFusion Application option and, as directed on the screen, click the New button. The icon to click is depicted in the instructions shown on the screen as in Figure 45.5.

This step creates a new configuration, as shown in Figure 45.6.

Figure 45.6

Adding a new ColdFusion debug configuration.



Here you can specify a name for the debugging configuration, which can be anything you like, such as `ows`, as I have used here since I created an `ows` project for which I'll want to perform debugging. The name is simply a designation you'll use later when launching a debugging session.

Then select the project that you want to debug, using the Project drop-down list of options. These are projects that you have created previously, as discussed briefly earlier, in the section "Define a ColdFusion Builder Project." The Server field will be populated automatically with the name of whatever server configuration you defined for the project.

NOTE

If you can't select any project in this New Configuration page, that's an indication that you've already created a debug configuration for that project. You cannot create another debug configuration for the same project.

Having selected a project (and server) name, you could click Debug to cause ColdFusion Builder to attempt to start a ColdFusion debugging session for the selected project or server, but let's leave that for the next section. For now, just click Apply and then Close to save these settings.

Okay—that really was a lot of introduction to get to debugging. You may be wondering why the process is so complicated, but there simply are a lot of moving parts. Again, most of the steps to this point are one-time configuration settings. Once set, you'll never configure them again.

Exploring the Debugger

We're now finally ready to debug a CFML page (`.cfm` or `.cfc` file). In this section we'll learn how to set breakpoints (places where control should stop while debugging), start a debugging session, step through code, and observe (or change) variables, among other things.

To recap, for a CFML page to be debugged, three things are necessary (each discussed previously):

- The ColdFusion server on which the code is running must be configured to permit debugging.
- A developer must have ColdFusion Builder open with the debugger settings configured: a project created with server, RDS, and debugger configuration created for the server against which debugging will take place.
- The debug configuration for that project and server must be started (typically with a breakpoint set for the file in question, though Break on a CFML Runtime Exception can stop on a line even without a breakpoint being set.)

Switching to the Debugging Perspective

When you initially open ColdFusion Builder, you may notice that the top-right corner shows an icon labeled "ColdFusion." This icon represents what ColdFusion Builder calls a perspective. A perspective defines a preconfigured layout of the screen, showing tabs and window panes that are appropriate for a given kind of development work.

In our case, we want to switch to what ColdFusion Builder calls the ColdFusion Debugging perspective. You can switch perspectives in a number of ways, including by choosing Window > Open Perspective > Other, or by clicking the Open Perspective icon shown to the left of the current perspective name in the top-right corner of the screen or by clicking the double-arrow icon to the right of that (if you've already opened another perspective). There are also some situations in which the debug perspective will open automatically, such as when the Break on Exception feature discussed previously is used.

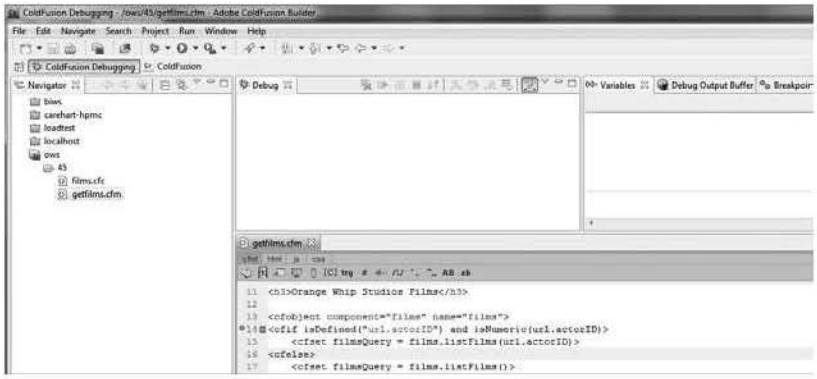
NOTE

You may find that your perspective names don't appear in the top right of ColdFusion Builder, but instead appear in the top left, under the menu. If you want to switch the location between these choices, right-click the perspective name and choose Dock On and then select Top Left or Top Right.

You'll notice that the interface changes (Figure 45.7) and you now see panes that are related to doing debugging. I'll explain these panes and their use in the remainder of this chapter.

Figure 45.7

The ColdFusion Builder Debug perspective.



Opening a File

When you switch to the Debug perspective, the ColdFusion Builder Navigator remains displayed on the screen, so if you want to debug a page, you need to have opened the page from a project (again, you can debug only files in projects). Drill down in the project directory tree to find the file and double-click to open it. The file will appear in an editor like that shown in Figure 45.7.

Notice that Figure 45.7 shows numbers next to each line. These line numbers are very helpful, especially for debugging. To enable the display of line numbers, you can choose Window > Preferences > General > Editors > Text Editors and select the option Show Line Numbers, or right-click in the gutter (where the numbers are shown in the figure) and choose Show Line Numbers.

Setting a Breakpoint

With a file open, you can identify a line of code on which you want the debugger to stop when it reaches that line during execution of the request. This is called setting a breakpoint. You can right-click in the gutter, that area to the left of the line (where the line number appears, or the gray area left of that) and choose Toggle Breakpoint. You can also set a breakpoint by using the key combination Ctrl+Shift+B or by choosing using Run > Toggle Breakpoint. Figure 45.7 shows that I had already set a breakpoint on line 14.

When you do that, a blue dot will appear to the left of the line (and line number), depicting that the breakpoint has been set. Note that it only makes sense to set a breakpoint on a line with CFML (tags or expressions), though ColdFusion Builder won't stop you from trying.

The breakpoint(s) you set will also be depicted in the breakpoints tab (click the tab shown at the top right of Figure 45.7), where they are displayed with their filename and line number.

Note that breakpoints remain enabled across editing sessions. In other words, if you close ColdFusion Builder and reopen it, the breakpoints you set previously will remain enabled. That said, breakpoints are specific to your editor, so if another developer opens the file, they will not see breakpoints you have set.

This may lead to a question: If breakpoints remain enabled across editing sessions, does that mean that a request will always be intercepted once a breakpoint is set? Not quite. Setting a breakpoint is the first step. The far more important step is to start a debugging session.

Starting a Debugging Session

Even with breakpoints set, the debugger doesn't do anything until you begin a debugging session. To begin a session, open a file within a project and choose Run > Debug or press the F11 keyboard shortcut. This will cause ColdFusion Builder to start a debugging session for the project associated with the file that's open.

There are still other ways to start debugging: Choose Run > Debug As > ColdFusion Application, or click the bug icon just above the file Navigator, or right-click in the editor and choose Debug As > ColdFusion Application.

Technically, you don't need to open the Debug perspective before using any of the options to start a debugging session. If the perspective isn't open, ColdFusion Builder will prompt you to confirm the perspective switch, asking if you want to switch to the debugging perspective, with a Remember My Decision option for future debugging sessions.

Assuming that all is configured properly, this should initiate a debugging session between your editor and the ColdFusion server.

WARNING

If you do not have your cursor focus within the editor of a currently opened file, then using any of the options mentioned here to start the debugger will restart whatever was the last-executed debugger configuration. This could be very confusing as it may not be the debug configuration for the files you intend to debug. Be sure to note the debug configuration that is started, as shown in the debugging window discussed next in the section "Understanding the Debug Status Window."

If you're debugging a local server, it will also cause ColdFusion Builder to open your default browser to start executing the URL for the file you selected, again assuming that you selected an open file and the cursor focus is within that file at the time you started the debugger.

Even if a browser window isn't opened automatically, you can certainly still open a browser: any browser, whether an external one or one of the available internal browser tabs within ColdFusion Builder. This topic is discussed further in the upcoming section, "Browsing a Page to Be Debugged."

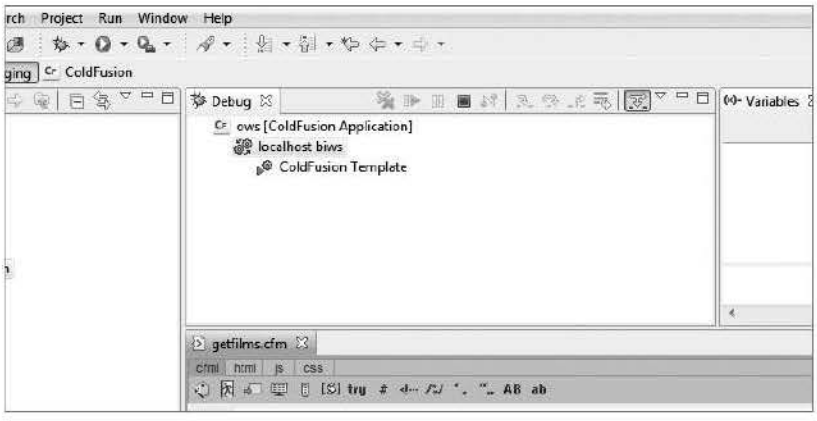
Understanding the Debug Status Window

After a debugging session is started, the debugger is now ready for you (or someone) to execute a request to be intercepted and debugged.

Figure 45.8 shows the kind of information that might be displayed in the debug window, which shows the debugger ready to accept requests for files in the selected debugging configuration. (I've selected to start my debugging configuration named `ows`. Technically, the value shown on the first line, with the CF icon to its left, is the name of the selected debugging configuration, and the next line below it is the name of the server used for the project in that configuration.)

Figure 45.8

A successfully started ColdFusion Builder debugging session.



The approach just described starts a debugging session for the specific file that was open. You can also start a debugging session for any existing debugging configuration through a couple of other approaches in ColdFusion Builder.

First, you can choose Run > Debug Configuration, which will open the same interface that we saw in Figure 45.4, where we defined debug configurations. You can select a configuration and click the Debug button in the lower-right corner.

Second, you can click the icon (just above the file navigator) that looks like a bug. If you've not previously used any of your debugging configurations, you'll be presented with the same dialog box as shown earlier to select one. And when you've selected more than one debugging session, you will be able to select one of them quickly by clicking the down arrow icon just to the left of the bug icon, which will display a list of previously selected debugging configurations. Choosing one will start that debugging configuration. Note that the menu also offers a Debug option, which also opens the interface shown in Figure 45.4.

When Starting a Debug Session Fails

Your attempt to start a debugging session can fail for a number of reasons. What are some of the things that could go wrong?

- **ColdFusion isn't started:** You'll get a pop-up saying "Connection timed out: connect" followed by another starting with "Unable to connect to the RDS server" and continuing with another message, "Connection timed out: connect." Start the ColdFusion server and try again.
- **RDS wasn't enabled in the ColdFusion server:** You'll get a pop-up starting with "Error Executing RDS command. Status code: 404" and then another reporting "Unable to connect to the RDS server." Again, the ColdFusion Builder debugger requires an RDS connection, so the server being debugged must have RDS enabled. You can learn how to enable it at `http://www.adobe.com/go/tn_17276`. After making the change, restart the server and try again.
- **The RDS authentication information provided was invalid:** You'll get a pop-up starting with "Unable to connect to the RDS server" and continuing with the error message "Unable to authenticate using the current username and password information." Correct the RDS connection information and try again. The RDS connection information can be edited by modifying the server definition, which you can do by right-clicking the server in the Servers view (Window > Show View > Servers). Be sure to select the server defined to be used for the project whose file you're trying to debug.
- **Allow Line Debugging wasn't enabled in the ColdFusion Administrator:** You'll get a pop-up starting with "Unable to connect to the RDS server" and continuing with the error message "Error Executing RDS command. Status code: 404." Return to the ColdFusion Administrator, as discussed in the section "Enable Debugger Settings," and ensure that Allow Line Debugging is checked. You must restart ColdFusion for this change to take effect. Try starting the debug session again.

- **The debugger port is not opened on the server:** You'll get a pop-up starting with "Unable to connect to the RDS server" and continuing with the message "Connection timed out: connect." You must open a hole in your firewall exposing the debugger port for access from the machine on which you're running the debugger, but see the important discussion of how ColdFusion uses a random debugger port and how you can resolve this in the earlier section, "Enable Debugger Settings." Then try starting the debug session again.

Browsing a Page to Be Debugged

With a debugging session enabled, it's now finally time to run your page to experience debugging. This step can actually be as confusing for some developers as any of the configuration features discussed to this point. While most debuggers may have some configuration that must be performed, as discussed earlier, with most debuggers the next step is simply to run the desired program in the IDE or in some sort of special debugging window. That's not necessary with the ColdFusion Builder debugger.

Instead, you simply run the CFML page request just as you normally would, whether from a browser, or via a Flex or Ajax client, or as a Web service call from some other environment, and so on. There's no special debugging window in which you must run your request. (Internal browsers provided by ColdFusion Builder are available; you don't have to use those, though you can.)

TIP

Since the debugger intercepts any request for a CFML page, it can even intercept some requests that have no browser or client at all, such as requests if you debugged the `OnSessionEnd` method of `Application.cfc` (which would be executed when a session ended), or requests in a ColdFusion event gateway like the DirectoryWatcher (which would be executed when there was a change in the directory it was observing).

If you have a ColdFusion Builder debugging session enabled against a server, with a breakpoint set in a file, then when that CFML code is executed, whether by you or anyone else, the debugger will stop on that line of code.

You may be surprised that the debugger intercepts any request for a CFML page, regardless of whether you request it or someone else does. This means that in an environment with multiple users making requests against a given server that you're debugging, you could intercept someone else's request. That could be either desirable or unexpected. Just be aware.

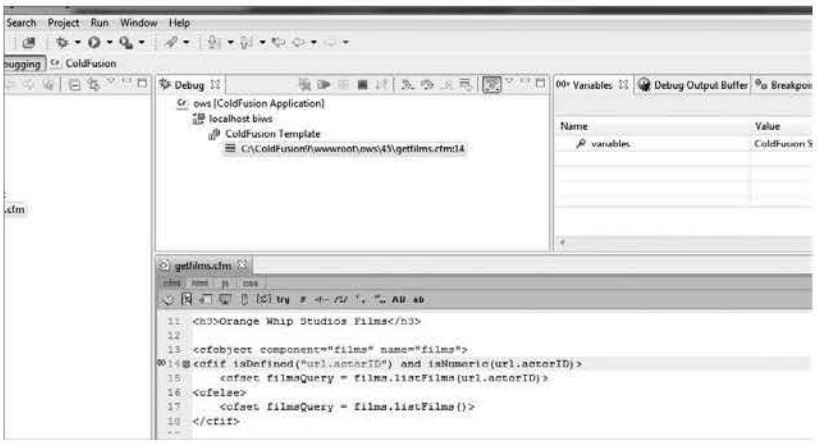
What You, the Developer, Will See

Assume that you or someone else has issued a request and it's been intercepted by the debugger at a desired breakpoint; Figure 45.9 shows how the ColdFusion Builder environment will reflect that you're stopped at that breakpoint.

First, notice that the Debug window at the top now reflects more information than we saw in Figure 45.8. It shows the complete path to the file being debugged and the line number at which execution has stopped. This complete path can help if you ever wonder what file you're viewing or debugging.

Figure 45.9

Execution stopped at a breakpoint.



Note also that in this case we see other lines there, which reflect what some may call the stack trace—or how the code we're stopped on was called by other code. For instance, if we were stopped in a CFC, include file, or custom tag, it would show the file (and line number) of the code that called this file.

Second, in the code window, you can see that where Figure 45.7 had showed only a blue dot next to the line on which a breakpoint had been set, now we see also a blue arrow overlaid atop it. (This may be too small to observe in the screenshot.) The arrow is a current instruction pointer, which reflects where the debugger has stopped execution. It shows the line that is about to execute. We'll learn about stepping through that and other code shortly.

What if the breakpoint doesn't fire? You may find that when the request is executed, the breakpoint does not fire. The first step is to ensure that all previous configurations are correct (you've opened a file in a project and properly defined a server and RDS server for that project) and that you've set a breakpoint and started a debug session for the project in which the desired file is located.

Even if all these settings are correct, occasionally the debugger doesn't pick up the breakpoint. You may even see an icon where the blue dot should have appeared (indicating that a breakpoint had been set on a line) showing instead a small question mark over the dot. In either case, try making some change to the file (while in the editor) and save it. You may even want to terminate and restart the debug session, as discussed later, in the section "Stopping the Debugger."

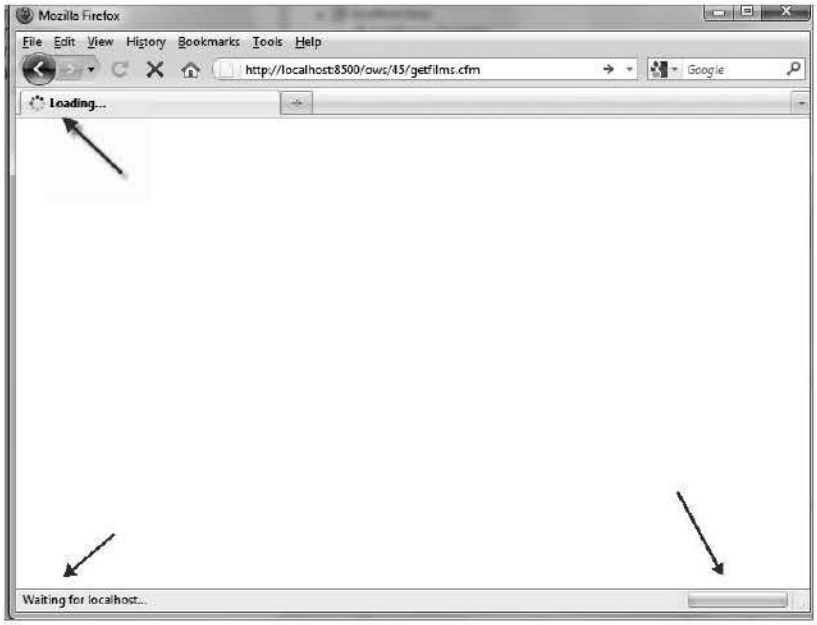
What the User Sees

While we sit here looking at the debugger, observing the fact that the debugger has intercepted the request, allowing us to look at the information above (and more to be discussed soon), you may—and should—wonder what things look like to the user making the request. If it's a regular browser user, what they'll see will appear as if their request is hung, as if the server is not responding, and in fact it's not. We've held up the request's processing so there is not yet a response for the user to see.

Just as when a user's request is waiting for any response from the server, they will see various indicators that they are waiting, which will vary depending on the browser being used. See Figure 45.10 for an example in Firefox, where I've highlighted with arrows the three ways that it shows that a page is waiting to complete loading. This is the page that triggered the debugging session in Figure 45.7.

Figure 45.10

The user waits for page execution to finish.



You may wonder, if the page had started to generate output before the breakpoint, whether that output would at least be sent to the browser. It is *not*. As with a normal ColdFusion request, the page output is buffered internally and sent only when the page completes (unless a `CFFLUSH` causes it to be sent sooner or an internal buffer size is reached). If you want to see what partial page output has been created in the output buffer, such a feature is available in the debugger, discussed later in this chapter, in the section "Observing the Debug Output Buffer."

Of course, this discussion has assumed we were referring to a normal browser request calling the page. If the page had been an Ajax or Flex client request, those two would show the request waiting, typically using some sort of prompt to the user (which might flash by in a split second under normal circumstances).

Beware Page Timeouts

It's possible that you could leave the browser waiting so long for a response that it would consider the page abandoned and could show a message indicating that it stopped waiting.

We mentioned previously how ColdFusion itself could report that the request exceeded a configured timeout. You would receive a pop-up message in the debugger reporting, "coldfusion.runtime.RequestTimedOutException: The request has exceeded the allowable time limit." In

this case, you should configure ColdFusion to allow CFML pages to run longer than normal, by adjusting the ColdFusion Administrator's Timeout Requests feature, discussed earlier in the section "Increase Maximum Request Timeouts."

If for some reason you can't get the Administrator setting changed, there's yet another option to prevent timeouts, at least on a page-by-page basis. When the normal execution of a page may be legitimately expected to exceed the server timeout, you can use the `CFSETTING` tag and its `RequestTimeout` attribute to specify a number of seconds that the page should be permitted to run before timing out.

Stepping Through Code

Now let's continue with the use of the debugger.

We've stopped the debugger on the line of code set as a breakpoint. There are many things we can do at the point, as we'll see in upcoming sections, but often the sole reason for using the debugger is to observe the flow of execution through the code (and among the files that the code may call upon). There are many beneficial reasons to step through the code.

Perhaps you're trying to understand why certain code is (or is not) being executed. Or maybe this is new code that you've been asked to maintain (or debug), and you want to use the debugger to become more familiar with its operation (versus statically reading the source code by hand).

A powerful feature of the debugger is that it opens files that are called upon as you step into lines that point to other files (such as includes or CFC methods). Thus it's also an excellent tool for learning where your code is going and what files are being used. Recall from Figure 45.8 that it shows you the full path to the file being debugged.

Finally, you may be new to ColdFusion in general, and the debugger is an excellent tool to help you become familiar with how CFML works.

You can step through the code on a line-by-line basis, or you can run until a next breakpoint is reached. And if you've stepped into some other file, you can have the debugger complete execution of code in that file and stop at the next line in the calling file after the line which called the other file.

Stepping Line by Line: Step Over, Step Into

The most common situation is that you want to have the debugger just proceed to execute the next line in the flow of execution. Here you have two choices.

First, you can choose to step over the line, which means simply that you want to execute the code and proceed to the next line (in the current file) that would have followed it (unless there are no more lines to execute, in which case the request will complete). This can be accomplished by choosing Run > Step Over, pressing the F6 key, or click the Step Over icon listed among the icons atop the debug window.

Your second option is to step into the next line of code (choose Run > Step Into, or press the F5 key, or click the Step Into icon). This option makes sense if the line of code about to be executed would call another file (such as a `CFINCLUDE`, custom tag call, or CFC method invocation) and you want to use the debugger to open that file and stop on the first line of CFML code in the newly opened file.

Let's return to our code example and follow as it steps into the call to a CFC method.

NOTE

If the tag you're interested in stepping into (which would open a file) happens to have an opening and closing tag (such as a `CFINVOKE`) with intervening CFML tags within it, the Step Into process won't happen until you reach the closing tag.

Notice that you don't need to tell the debugger where the file is. That's one of the very powerful features of the debugger: it figures out the file that ColdFusion would use. This is a great feature when you have any doubt about what file is being opened. (The file may not always be obvious to you due to ColdFusion mappings and other mechanisms by which ColdFusion may find and open a named file; if the file can't be found, you may need to take an extra step, discussed at the end of this section.)

TIP

Here's a bonus ColdFusion Builder feature. You don't need to use the debugger to have it open the file associated with a tag or function that would point to another file. ColdFusion Builder has a handy feature that lets you hold down the Ctrl key (Windows) or Mac key (Macs) while using the mouse to hover over such a filename. An underline will appear, indicating that the mouse is pointing to a file, and clicking will open that file. This feature works with both CFML and HTML.

What if you don't want to follow the flow of execution into the file that would be opened (by the line of code about to be executed)? You don't have to open it. You can instead use the Step Over option, and the code in the current file will be executed (just as during normal execution of the page) and the current instruction pointer will be set at the next line in this page (unless there are no more lines to execute, in which case the process will complete, or if you've stepped into a file, the debugger will return to the calling file).

If you do choose to use Step Into on a tag that opens a new file, note that once you're inside that new file, you have the same options as discussed earlier, so you can use Step Over and Step Into, as well as a new option, Step Return, discussed next.

CAUTION

Be careful not to uses Step Into when you don't need to (don't use it on tags, functions, or expressions that would not otherwise have any reason to open a new file). For one thing, it can cause the request to take longer than it would with a Step Over.

More important, on some tags and functions, using Step Into will actually try to open an underlying file unexpectedly. Some CFML tags, such as `CFDUMP`, `CFSAVECONENT`, and a few more, as well as some functions, such as `writedump()`, are actually executed as CFML pages by ColdFusion. ColdFusion Builder will try to open the respective file, and you'll get an error in ColdFusion Builder reporting "source not found" and offering an Edit Source Lookup Path button. But the path used, such as `E:\cf9_final\cfusion\wwwroot\WEB-INF\cftags\dump.cfm`, does not really exist on your ColdFusion server. The path corresponds to a location where the file existed when ColdFusion itself was compiled.

When considering the idea of stepping into files, it's important to remember that for the debugger to locate a file, the debugger must be able to find the file, and it expects to find it in the project.

This creates a dilemma that often stumps new debugger users: What if the file being opened is not defined in the project (which perhaps is pointing to a single Web site's Web root)? What if the file is instead in some directory that's shared among multiple Web sites? An example may be a CFC or custom tag stored either in the default ColdFusion custom tags directory or in a directory pointed to by a custom tag mapping in the ColdFusion Administrator. While ColdFusion can certainly find that file when it needs to, how can the debugger be told?

The magic is in the definition of the project (and for remote servers, the debug mappings mentioned previously). The solution is to add a link folder definition in the project's properties. This can be done in the project's properties (right-click a project name and choose Properties); then in the list of options, select ColdFusion Project and under Additional Resource, select Add to point to the location. For more information, see the ColdFusion help topic "Link to resources outside the workspace."

Step Return and Resume

If you do step into a new file, and you decide you no longer want to step line by line through the code, you don't need to laboriously make your way through the remaining lines of code. That's what Step Return is for (Run > Step Return, or F7, or the appropriate icon). That option completes execution of the file that's been stepped into (unless there's another breakpoint later in the file) and returns control to the line of code that follows whatever tag or function called the file. All the remaining code in the file is executed, of course. You just don't step through it.

Similarly, if at any point while debugging a request you decide that you no longer want to step through the code (whether you've stepped into a file or not), you can have the debugger proceed to execute the entire remainder of the request using Resume (Run > Resume, or F8, or using the appropriate icon). Unless there's another breakpoint in the flow of execution, the debugger will finish and the completed page will be shown to the browser (or whatever client made the request for the CFML page).

The Terminate feature is discussed in the last section of this chapter, on stopping the debugger.

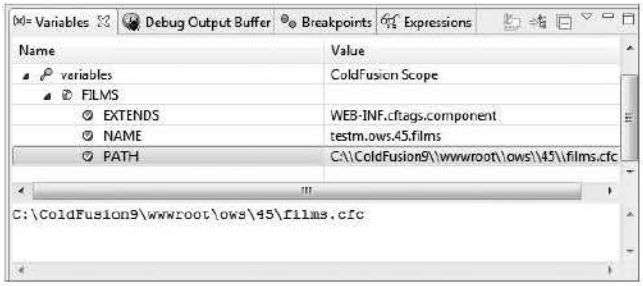
Observing Variables and Expressions

For some developers, just being able to have a granular, line-by-line depiction of the flow of execution through their pages is miraculous enough. But there's more. While you're at a breakpoint, you can also observe the value of any variable, in any of ColdFusion's scopes. This includes all the usual scopes (`FORM`, `URL`, `SESSION`, `REQUEST`, and so on), as well as things like the `ARGUMENTS` scope when in a function, the `THIS` and `LOCAL` scope as created by a CFC, and so on.

The Variables Pane

The first place to look is the Variables pane, which appears to the left of the Breakpoints pane as shown in Figure 45.9. If you select it, you can traverse the tree of available scopes, expanding them to see the name and value of any variables in the selected scope. See Figure 45.11, where I've selected the VARIABLES scope. Since this code just created a CFC instance that was stored by default in the VARIABLES scope (as films), , I've expanded that variable, and you can see the various internal variables that ColdFusion tracks (as structure keys) for a CFC instance, such as the actual path where it was found.

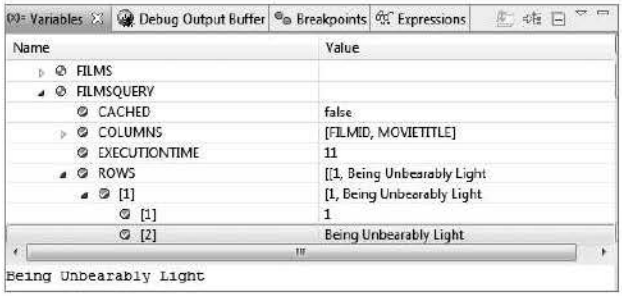
Figure 45.11
Display of VARIABLES scope.



The depiction of variables in this pane is of course not limited to the simple variable values shown above. If the code had executed a query, we could expand the query to see all its related variables (`recordcount`, `columnlist`, etc., and even each row and column in the result set). If we stepped through the next line of code, which will call the CFC that returns a query, we can view the query result set. Figure 45.12 shows the query result set expanded, showing the first row and its two column values (and the column names are shown in the `columns` variable also displayed for the query).

Here are a couple of tips that could make working with the Variables pane more effective. First, if the value of the variable you wish to observe is too long to be displayed in the Value column provided, while you could select the column divider to make it wider, a simpler solution is that if you select the variable, you'll notice (as shown in Figure 45.12) that the selected value is also displayed in a Details window at the bottom of the Variables pane. This can be more easily scrolled to view the full value.

Figure 45.12
Display of a query.



Also, while in the Variables pane, you can scroll up and down in this list to see all the available variables in the selected scope. But note that you can also use an available Find feature, which can search through all the scopes to find a variable of a given name. Right-click the variables pane and choose Find, or left-click on it and use Ctrl-F.

Remember, too, that the list of scopes shown is limited to a default subset shown in the Debug Settings preferences (or those you chose during setup), as discussed previously in the section of the same name. Recall from that discussion that you should be careful about adding too many scopes, as that can slow the execution of the debugger. Another option is that you can instead ask the debugger to show you some specific variable using the Expressions pane, discussed next.

The Expressions Pane

You may want to observe a variable value to watch how it changes over lines of code as you step through it, or perhaps over multiple requests for the page. While you could traverse the Variables pane to find and display your desired variable each time, you could instead use the available Expressions pane. This has the benefit that you choose to name what variable you want to watch (including one in a scope that is not being displayed in the Variables pane because of Debug Settings preferences).

To add a new expression to the pane, right-click it, choose Add Watch Expression, and then provide the name of the variable. You don't need to surround the variable in pound signs (though it's okay if you do). The variables you name will be displayed in the Expressions pane (in the order in which you add any). When you're stopped at a breakpoint, its value will be shown—assuming the variable is defined at that point in the code. If it's not, the variable will be shown in red. (If you ever find that a variable which should be defined is not displaying, right-click it and choose Reevaluate Watch Expression. You can also right-click an expression and choose Edit Watch Expression.)

Note that you're not limited to naming variables with simple values. You can even name an entire scope and all its values will be displayed. You can also create a new Watch Expression by right-clicking on a variable in the Variables pane and choosing Watch.

Another benefit of the Expressions pane is that, like breakpoints, the items you create here remain across debugging sessions and editor restarts. And like the variables pane, you can use the Find command to search among them.

Changing Variables on the Fly

Here's a feature that will amaze some readers. Did you know that you can change the value of a variable on the fly, while the program is being debugged? Yes, you can. While at a breakpoint, you can right-click on a variable name in either the Variables or Expressions pane and choose Change Value. It's really that simple. The value will be changed for the remainder of the execution of the request (or until you or some other code changes it again).

Observing the Debug Output Buffer

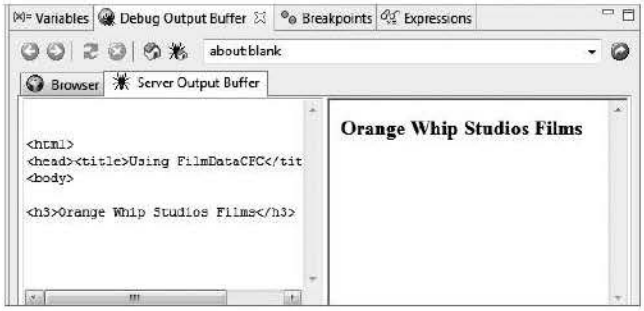
Finally, the last feature to discuss is the Debug Output Buffer pane. I mentioned it previously as a way that you can view the generated output of the CFML code (whether you're generating HTML, XML, JSON, or whatever else you may be outputting as the page response).

The Debug Output Buffer pane is what enables that processing. The pane appears to the right of the Variables pane in the top right of the Debugging perspective. Within that pane, select the Server Output Buffer tab (I'll discuss the Browser tab in a moment).

Server Output Buffer Tab

The Debug Output Buffer tab consists of two panes (Figure 45.13).

Figure 45.13
Server output buffer pane.



The left pane shows the raw generated browser code (HTML, in this case) that the CFML template has generated to this point. You could scroll down and see that it's not a complete page, since the request is in the middle of generating the content. Further testament of that fact is in the right pane, which shows an attempt to render that HTML in a built-in browser.

WARNING

If you open the Debug Output Buffer pane after having already started to debug a request, you may not see it displaying any of the request's output. Try completing and restarting the request to see output presented in this pane.

As you step through the code, you'll be able to observe the building of the page as it might appear in the real browser. This is just a simple rendering of the HTML in a temporary file created by the editor. It's not really executing in the context of the URL of the server, so some things (like the image shown) may not render correctly.

Also, don't be misled by the URL displayed atop the pane. That's not the URL of the page being debugged (unless you typed a URL there). That address bar is really there for the sake of the other window in this pane, the Browser tab, discussed next.

And to be clear, the server output buffer is populated regardless of the browser used to launch the request being debugged (again, it could be launched by a user other than you).

Browser Tab

I had mentioned previously that to trigger the debugger you could make a request from any browser, and I'd mentioned that there were some internal browsers available within ColdFusion Builder. This is one of them. If you wanted to, you could make your page requests using this address bar and the result will be shown in the Browser tab of this pane, not in the Debug Output Buffer pane. Indeed, it wouldn't really make sense to use that address bar while you're viewing the result of a page being debugged.)

It's a real browser. In fact, the URL that's shown (unless you entered one yourself) is the very value of the Home Page URL that was mentioned briefly in the discussion of Debug Settings, Figure 45.4. (So that's what that's for!) If you click the home page icon atop that Browser pane, that URL will be loaded.

That said, I should clarify that some might not care for this browser tab because the window is so small. But here's a tip: As with all the panes discussed here, you can resize this one and even detach it so that it floats as a window over top the ColdFusion Builder interface.

Stopping the Debugger

Finally, when you're finished debugging pages, you may wonder how you stop debugging. There are several ways, each with slightly different purposes.

Terminating Debugging Within the Debugger

First, if you close ColdFusion Builder, that will terminate your debugging session, which means fortunately that you can't leave it running by mistake (at least not by closing it without remembering to stop it).

You can also stop the debugging session while you remain in the editor. There are a few ways. You can use the Terminate button. That stops the debugging session and lets the request run to completion. You can also choose Run > Terminate.

You can also right-click in the Debug pane (where the file name and line number of what's being debugged is shown), and there choose Terminate. That will stop the debug session and leave an indication in the Debug pane that you had used that debug configuration. That way you could right-click and choose Relaunch as another way to start a session. You can also right-click and choose Terminate and Relaunch if, for instance, you made changes in the debug configuration. Finally, you'll see another option, Terminate and Remove, that will remove the debugging session from the debug pane.

Forcing Termination from the Server

I mentioned above that closing the editor would terminate any debugging sessions. But what if you instead mistakenly left the Editor open and a debug session running while you left for lunch—or for the day? This could be a real problem. Users might run a request for a page you're debugging,

and at least the first of those would have their request hang waiting for a debugging activity that might not happen for hours.

Here's where it's important to note that there's one final way to stop debugging that can be done from the server rather than from the editor. Recall the discussion of the Admin console page for Debugging & Logging > Debugger Settings. I mentioned the available Start Debugger Server button, which once debugging begins becomes a Stop Debugger button, and an associated Restart Debugger Button appears.

This is the situation where you would use those buttons. You could either use Stop Debugger (and then Start Debugger) or just use Restart Debugger. That will terminate all current debugging sessions, thus freeing up the requests that were pending waiting for a debugging session to complete. Even if you'd used Stop Debugger, the first request for a debugging session from a developer would restart the debugger server anyway.

Think of the Stop or Restart as like a forced termination of debugging from the server. All developers who were performing debugging will have their debug sessions terminated, so it's a bit brute force, but they can easily restart them. The good news is that they do need to manually restart them, so the person who left their editor (and debugger) running when they left will no longer be the cause for intercepted requests.

NOTE

The Restart option also can be used if, for any reason, ColdFusion Builder crashes during a debugging session. In that case, on restarting and launching a new debug session, you may get a breakpoint error reporting "Another session has already set a breakpoint there." Clicking the Restart button should resolve the problem.

If you wonder how to stop all debugging from happening any more on the server, clearly that's not what the Stop Debugger Server does (since the next request for a debug session will start it up). Instead, turn off (uncheck) the option on that Admin page for Allow Line Debugging. That takes effect immediately, and ColdFusion would no longer permit debugging requests against that server.

Other Features

While we've covered a broad range of features and troubleshooting topics, there are still more features that cannot be elaborated upon due to space constraints.

For instance, you can copy variables and expressions by right-clicking them in the Variables or Expressions pane. You can also export and import breakpoints (to share them with other developers) by right-clicking in the Breakpoints pane.

There are also options to temporarily disable one or more breakpoints or to skip all breakpoints (both via an icon on the Breakpoint pane's toolbar, or via Run > Skip All Breakpoints.)

That said, there are also some features on the Run menu that do not work in ColdFusion Builder (but instead are remnants of the underlying Eclipse Java debugger), such as method breakpoints, watchpoints, and step filters.

Also, when you're done debugging and want to return the display to the more typical development (rather than debugging) perspective, you can either choose Window > Open Perspective > Other > ColdFusion or click the icon for the ColdFusion perspective, which again (as discussed earlier) appears either in the top left or top right of the interface.

Finally, be aware that the ColdFusion 9 manual, *Developing CFML Applications*, has a section "Using the ColdFusion Debugger," but there are some differences between what this section discusses and what you'll experience in ColdFusion Builder. This manual was created prior to the release of ColdFusion Builder and discussed an older plug-in approach to debugging ColdFusion.

It's great to see how in so many ways the Adobe Engineering team has thought ahead to help not only provide a debugger but one that addresses some common challenges that CFML developers would face. For many, the biggest challenge of using a debugger is simply remembering that it's there. I hope this discussion has motivated you to consider using it.