

1. Home	2
1.1 Introduction & Concepts	3
1.2 Setting up Java for Debugging (ColdFusion, Railo & Lucee)	4
1.3 Getting Started - Install FusionDebug	6
1.3.1 Download and Run Eclipse	7
1.3.2 Eclipse - Views and Perspectives	7
1.3.3 Installation of FusionDebug via the Eclipse Software Install Manager	8
1.3.4 Installing and Activating a License	10
1.3.5 Configuring a Project in Eclipse	12
1.3.6 FusionDebug Perspective - Basic Introduction	15
1.3.7 Getting Started - Configuration Example - ColdFusion 2016, Lucee 4.5.3	17
1.3.8 Uninstall FusionDebug via the Eclipse Software Install Manager	23
1.4 FusionDebug Concepts - Quick Tour	24
1.4.1 Exploring FusionDebug	24
1.4.2 FusionDebug Configuration	27
1.4.3 Debug Target	28
1.4.4 Threads	28
1.4.5 Stack Frames and the Stack	29
1.4.6 Variables	30
1.4.6.1 Change Value Dialog	30
1.4.7 Expressions	31
1.4.8 Breakpoints and the Current Instruction Pointer	31
1.4.9 Stepping	32
1.4.9.1 Step Over	32
1.4.9.2 Step Into	32
1.4.9.3 Step Return	32
1.4.9.4 Resume	32
1.4.9.5 Auto-stepping	32
1.4.10 Terminate	34
1.4.11 Source Code Lookup	34
1.5 The Source Code Lookup Tab	34
1.5.1 What is a Lookup?	35
1.5.2 How to find the Source Code Lookup Tab	36
1.5.3 How to use the Source Code Lookup Tab	38
1.5.4 Alternative Webserver Setup	39
1.5.5 CF Mappings Setup	40
1.5.6 Linux and UNIX Setup	41
1.5.7 Multiple Project Setup	42
1.5.8 Remote Debugging Setup	43
1.5.9 Single Machine Setup	43
1.6 FusionDebug - Configuration Examples	44
1.6.1 Creating a FusionDebug Configuration	44
1.6.2 Working Example - Remote Debugging Setup	47
1.7 FusionDebug Feature Reference	52
1.7.1 Debug View	52
1.7.2 Breakpoints View	54
1.7.3 Expressions View	55
1.7.4 Eclipse Editor Context Menu	56
1.7.4.1 Setting Variables	57
1.7.4.2 Run to Line Operations	58
1.7.5 Conditional Breakpoints	58
1.7.5.1 How to use Conditional Breakpoints	59
1.7.5.2 Overheads	60
1.7.6 Hitcounts	60
1.7.7 Break on Runtime Exceptions	61
1.7.7.1 How to Setup Break on Runtime Exceptions	61
1.7.8 Non-Intrusive Debugging	62
1.7.9 Configuration Dialog	63
1.7.9.1 Connect Tab	64
1.7.9.2 Source Code Lookup Tab	65
1.7.9.3 Common Tab	66
1.8 Troubleshooting / FAQ	66
1.8.1 [FDS-128] Using FusionDebug to debug Flash Remoting and Gateway Messaging Requests	69
1.8.2 [FDS-130] FusionDebug: Feature Focus - Non-Intrusive Debugging	74
1.8.3 [FDS-118] OnDemand Seminar: FusionDebug Introduction (with Railo)	75
1.8.3.1 [FDS-125] Configure FusionDebug 3 for use with Railo 3.1 on Tomcat	76
1.8.3.1.1 [FDS-121] FusionDebug: Feature Focus - Breakpoint Conditions & Hitcounts	77

Home

What's new in 5.0

We are excited to bring the latest FusionDebug update - FusionDebug 5.0.0. Thank you all for your feedback, we always appreciate it! Here's a rundown of what's new in 5.0.0.

Support for Eclipse Neon

Since our last release, Eclipse has released a new version; Eclipse Neon (4.6.x).

Data from SQL queries

After a number of requests, the latest version of FusionDebug is able to reveal data from SQL queries.

Support for Lucee and Railo

FusionDebug now allows you to continue debugging your code that is hosted on Lucee and Railo services. **Lucee 5.1.0.034** version and above is **WORKING** as expected with the latest FusionDebug release. Lucee **4.5.5.006** version is **WORKING** as expected with the latest FusionDebug release.

Please note that **Lucee 4.5.4.017** will not work with the latest update. At the moment, the engineering team is working closely with the Lucee developers in order to fix that issue. More information about the issue mentioned above can be found in the release notes [here](#).

What's new in 4.0

Support for ColdFusion 2016

Another update from Adobe which has just been released - ColdFusion 2016 - is now fully supported with FusionDebug.

Support for Eclipse 4.4 and 4.5

Since our last release of FusionDebug, Eclipse have released two new updated versions of the Eclipse IDE;

- Eclipse Mars 4.5
- Eclipse Luna 4.4

We have ensured that FusionDebug works alongside the latest versions of Eclipse.

Support for ColdFusion Builder 2016

We now offer support for Adobe ColdFusion Builder 2016. A update from ColdFusion Builder 3 IDE which "is the only professional IDE which allows you to build and deploy web and mobile applications".

New Support for Lucee and Railo

FusionDebug now allows you to continue debugging your code that is hosted on Lucee and Railo services.

Support for Java 1.8

FusionDebug now supports Java 1.8

What's new in 3.7

We're excited to bring you another update to FusionDebug - FusionDebug 3.7. Thank you for all the feedback from the previous release, we always appreciate it! Here's a rundown of what's new in 3.7.

Support for ColdFusion Builder 3

We now offer extensive support for Adobe ColdFusion Builder 3. An all-new IDE from Adobe which "is the only professional IDE which allows you to build and deploy web and mobile applications".

Support for ColdFusion 11

Another update from Adobe which has just been released - ColdFusion 11 - is now fully supported with FusionDebug. Including member functions and new CF tags.

What's new in 3.6

We're excited to be able to bring you FusionDebug 3.6, and we'd like to thank our users for the many great suggestions and bug reports you sent us -- we truly appreciate it! Here's a quick rundown of what's new in 3.6.

Support for the latest Servers

We now offer extensive support for ColdFusion 9, 10 and Railo 4. This includes full support for Closures, so if your servers are using them - good; you can continue to do so and debug your pages with the latest FusionDebug software. Of course, we still offer full support for CF 6,7,8 and Railo 3.

As well as this, the latest Eclipse (Eclipse Kepler) is also fully supported, along with previous versions (3.1 to 4.3).

Integrated help with Eclipse

FusionDebug now offers help from within the Eclipse environment. Simply navigate to the help contents of Eclipse ("Help" --> "Help Contents") and click on FusionDebug. These help pages offer full help on Setting up a connection to a ColdFusion or Railo server, and debugging a page to get you started.

Browse for your Servers

We've made a slight improvement to the "Source Code Lookup" tab to make it easier to find specific CFML Servers. If your files are located locally, simply browse for a folder by clicking the "Browse" button. Once you have chosen the folder and clicked "open", the path will populate the dropdown box and will be added to the CFML Folder Table. If your folders are located remotely you will not be able to scan your local machine so the browse button can not be used. You can still manually enter the path to your CFML Server folder if you would prefer.

Bug Fixes

We've also worked hard to eliminate the bugs which you reported to us from our 3.5.x release. Here's a quick rundown of the major bug fixes.

Handshake Error

The main one reported by you is now fixed - you can now debug against a 1.7 JVM without receiving any handshaking errors.

Introduction & Concepts

Introduction

Welcome to **FusionDebug**, the interactive debugger for ColdFusion applications. This document introduces the concepts you will need to start debugging your own applications, guides you through the configuration process and gives you some helpful hints on using FusionDebug to it's full potential.

FusionDebug is **delivered as a plugin** for the Eclipse Platform. Eclipse is a universal framework for applications – not just debuggers – and provides many features that make developing interactive applications simpler. We selected Eclipse because it already has a good tradition of being a comprehensive, well-supported, stable platform, and a proven architecture for debuggers.

Many other plugins are available for Eclipse – **CFEclipse**, for example, is a leading environment for writing ColdFusion scripts. **If you have CFEclipse installed in your environment, FusionDebug will integrate with that environment when it needs to open an editor.** If not, don't worry – FusionDebug will also use the standard Eclipse text editor.

The sections below will guide you through the process of downloading, installing and using FusionDebug.

Why do I need a ColdFusion Debugger?

Debugging ColdFusion pages usually requires using lots of `<cfoutput>` or `<cfdump>` statements, or writing to log files. But as applications grow more sophisticated with the addition of object oriented components, and the integration of Flex, it's becoming increasingly difficult to solve complex problems.

FusionDebug solves these problems by allowing you to suspend any page or component **during its execution** and see all the variables used by that page. You can add specific **watch expressions** or even **set variable** values while the page is suspended. Pages can then be single-stepped or resumed.

You may think you do not need a ColdFusion debugger because you've never used one. FusionDebug will change the way you work.

What do I need to get started?

The first thing you have to do in order to get started with FusionDebug is to access our official website Download page and **get a copy of the latest FusionDebug version**. The next step will be to access the **installation instructions** listed here and **install** FusionDebug with the Eclipse environment or with the ColdFusion Builder environment.

Additionally, in the installation instructions page you will be able to find analytical instructions on how to install the CFEclipse package.

Video Tutorials

Another way of getting familiar with FusionDebug is by watching the **quick** video series on our website, FusionDebug Videos.

Below, you will be able to find an introductory video about FusionDebug and how to get started.

Setting up Java for Debugging (ColdFusion, Railo & Lucee)

Setting up ColdFusion for Debugging

This procedure adds the **required debugging parameters** to a standard ColdFusion (JRun) installation.

1. **Stop** all running ColdFusion services.
2. **Locate** the JVM arguments file **jvm.config**. Below you can find the most common directories the jvm arguments file is located.
 - Coldfusion 8: {CF8_HOME}\runtime\bin\
 - Coldfusion 9: {CF9_HOME}\runtime\bin\
 - Coldfusion 10: {CF10_HOME}\cfusion\bin\
 - Coldfusion 11: {CF11_HOME}\cfusion\bin\
 - Coldfusion 2016: {CF2016_HOME}\cfusion\bin\
3. **Make a copy** of this file, and store the copy somewhere on your disk. If any issues will arise, you can **restore** the copy to this location.
4. **Open** this file with a text editor of your preference.
5. **Locate** the line beginning with "**java.args**". These are the settings used to start the JVM, in which ColdFusion runs.
6. **Remove** the following options if they exist:
 - `-XX:+UseParallelGC`
 - `-Xdebug`
 - `-Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=5005`
7. **Add** these options (Ensure they are on the same line - at the end):
 - `-Xdebug -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000`

If you have any applications listening/running on port 8000, please change the port to another one which is not in use.

8. Start all ColdFusion services.

Below you will find an example of the "java.args" arguments.

```
# Arguments to VM
java.args=-server -Xms256m -Xmx512m -XX:MaxPermSize=192m -XX:
+UseParallelGC -Xbatch -Dcoldfusion.home={application.home} -Djava.
awt.headless=true -Dcoldfusion.rootDir={application.home} -Djava.
security.policy={application.home}/lib/coldfusion.policy -Djava.
security.auth.policy={application.home}/lib/neo_jaas.policy -
Dcoldfusion.classPath={application.home}/lib/updates,{application.
home}/lib,{application.home}/lib/axis2,{application.home}/gateway
/lib/,{application.home}/wwwroot/WEB-INF/cfform/jars,{application.
home}/wwwroot/WEB-INF/flex/jars,{application.home}/lib/oosdk/lib,
{application.home}/lib/oosdk/classes
-Dcoldfusion.libPath={application.home}/lib -Dorg.apache.coyote.
USE_CUSTOM_STATUS_MSG_IN_HEADER=true -Dcoldfusion.jsafe.
defaultalgo=FIPS186Random -Xdebug -Xrunjdw:transport=dt_socket,
server=y,suspend=n,address=8000
```

Setting up Railo and Lucee for Debugging

There are two different types of Railo servers; an **EXPRESS installation with Jetty** and a **Railo Server with Tomcat**.

Please note that the installation process for the Railo application server is the same as the Lucee application server.

Please note that the latest version of **FusionDebug 5.0.0 supports** the following versions of the Lucee application server; **Lucee 4.5.2, Lucee 4.5.3** and any **older versions** of the Lucee application server. The reason why Lucee 4.5.4 and Lucee 5.x is not working with FusionDebug is because the latest Lucee versions do not generating a fully qualified file path in the .class files and as a results, the user will not be able to fire any breakpoints.

Our development team is currently working with the Lucee developers in order to find a solution of this issue. When the fix will be available, you will be notified accordingly.

Railo Express Installation (Using the start.bat)

1. **Stop** any Railo running instances.
2. **Locate** the **start.bat** file in the installation directory of the application server.
3. **Make a copy** of the **start.bat** file and store it in a temporary location in case of errors during editing.
4. **Replace** the **start.bat** file, not the copy, with a replacement batch file, based on your operating system.
5. If you have any other applications running on port 8000, change the port number to a free port.
6. **Start** the Railo services

The JVM arguments file should look similar to the following.

```
"jre/bin/java" -Djetty.port=8888 -DSTOP.PORT=8887 -DSTOP.KEY=railo -
Xms256M -Xmx512M -Xss1m -Dcom.sun.management.jmxremote -Djava.
compiler=NONE -Xnoagent -Xdebug -Xrunjdw:transport=dt_socket,
server=y,suspend=n,address=8000 -jar start.jar
```

Railo / Lucee Server with Tomcat

AS A WINDOWS SERVICE

- **Stop** the Railo or Lucee services.
- **Locate** the "**Railow**" or the "**Luceew**" application. Generally located in the following directory.
 - {Railo installation directory}\tomcat\bin\
 - {Lucee installation directory}\tomcat\bin\
- **Access** the "**Java**" tab and **insert** the JVM arguments below:
 - -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000

If you have any other applications running on port 8000, change the port number to a free port.

- **Start** the Railo or Lucee services.

AS A SCRIPT (CATALINA.BAT)

- **Stop** the Railo or Lucee services.
- **Locate** the and **open** the "**Catalina.bat**" file. Generally located in the following directory.
 - {Railo installation directory}\tomcat\bin\
 - {Lucee installation directory}\tomcat\bin\
- **Insert** the following argument in the Catalina.bat file.
 - set CATALINA_OPTS=-Djava.compiler=NONE -Xnoagent -Xdebug -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000

If you have any other applications running on port 8000, change the port number to a free port.

- **Start** the Railo or Lucee services.

LINUX

- **Stop** all running Railo or Lucee services.
- **Locate** the **setenv.sh** file. Generally located in the following directory.
 - {Railo installation directory}\tomcat\bin\
 - {Lucee installation directory}\tomcat\bin\
- **Insert** the following JVM arguments in the **setenv.sh** file and at the end of the **CATALINA_OPTS** argument:
 - -Djava.compiler=NONE -Xnoagent -Xdebug -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000

If you have any other applications running on port 8000, change the port number to a free port.

- **Start** all Railo or Lucee services.

Below you will find an example of the "**setenv.sh**" arguments file.

```
JAVA_OPTS="-Xms256m -Xmx1024m -javaagent:${CATALINA_HOME}/lucee
/lucee-inst.jar"
CATALINA_OPTS="-Djava.compiler=NONE -Xnoagent -Xdebug -Xrunjdpw:
transport=dt_socket,server=y,suspend=n,address=8000"
```

Getting Started - Install FusionDebug

Child Pages

- Download and Run Eclipse
- Eclipse - Views and Perspectives

- Installation of FusionDebug via the Eclipse Software Install Manager
- Installing and Activating a License
- Configuring a Project in Eclipse
- FusionDebug Perspective - Basic Introduction
- Getting Started - Configuration Example - ColdFusion 2016, Lucee 4.5.3
- Uninstall FusionDebug via the Eclipse Software Install Manager

Download and Run Eclipse

Introduction

This page will guide users step-by-step through the process of installing and running Eclipse in their environment.

Getting Started

If you have not installed Eclipse in your environment, please access the link listed below in order to get a copy of the latest Eclipse version.

- [Download Eclipse](#)

Eclipse is provided as a zip (compressed) file and when the download process has been completed, please uncompress the file to a folder somewhere in your Hard Disk. The uncompress process can be achieved with a tool of your preference OR if you are using a Windows 7/8/10 machine, you can also right click on the .zip file and choose the "**Extract All**" option.

Windows: Eclipse can be started by double-clicking the **eclipse.exe** executable file.

Linux: Eclipse can be started by running the eclipse script (**eclipse-inst**) from the Terminal Window.

When Eclipse starts up for the first time, it will display the Welcome screen, as shown in Figure 10. You may like to place a shortcut to Eclipse on your desktop to make it easier to launch in future.

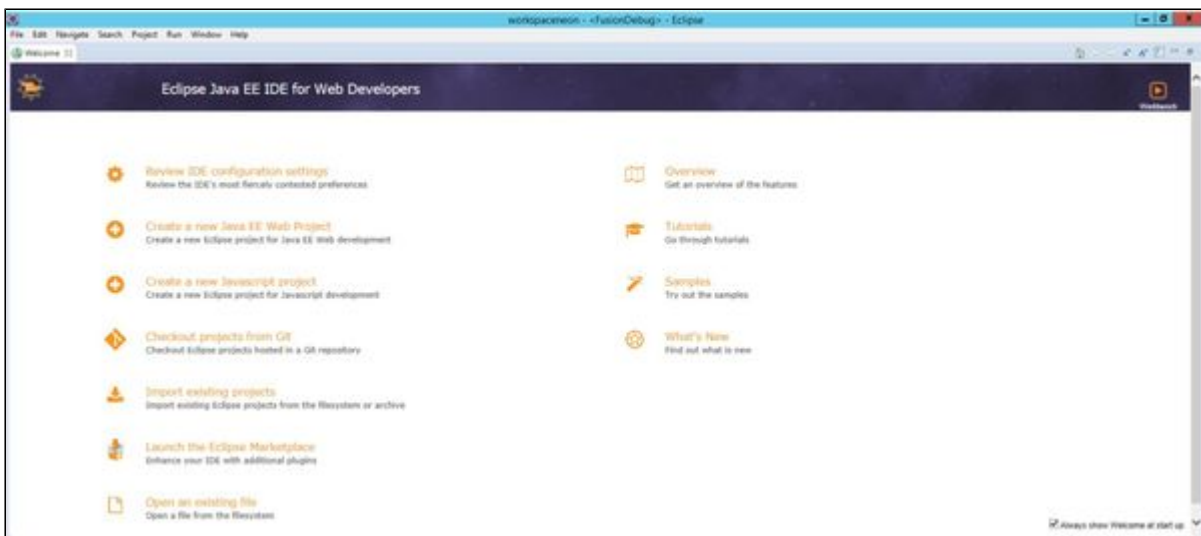


Figure 10: Eclipse Neon Welcome Screen

Don't worry if you have existing ColdFusion projects on your disk – Eclipse will be able to use those too, even if they are not located in the workspace and you can also change the location of the workspace later.

After launching Eclipse for the first time, Eclipse will ask where it should locate your **Workspace**. This is an Eclipse storage area where **new** projects will be created by default, and where Eclipse will store its settings. Accept the default location, and optionally check the box marked "**Use this as the default and do not ask again**".

Proceed directly to the workbench by selecting the **Workbench** icon (the rightmost round button) OR by simply closing the Welcome screen.

Eclipse - Views and Perspectives

Eclipse is now installed and ready for use, see Figure 11. This is the **Java Perspective** and you can see the currently-active perspective at the top-right of the window.

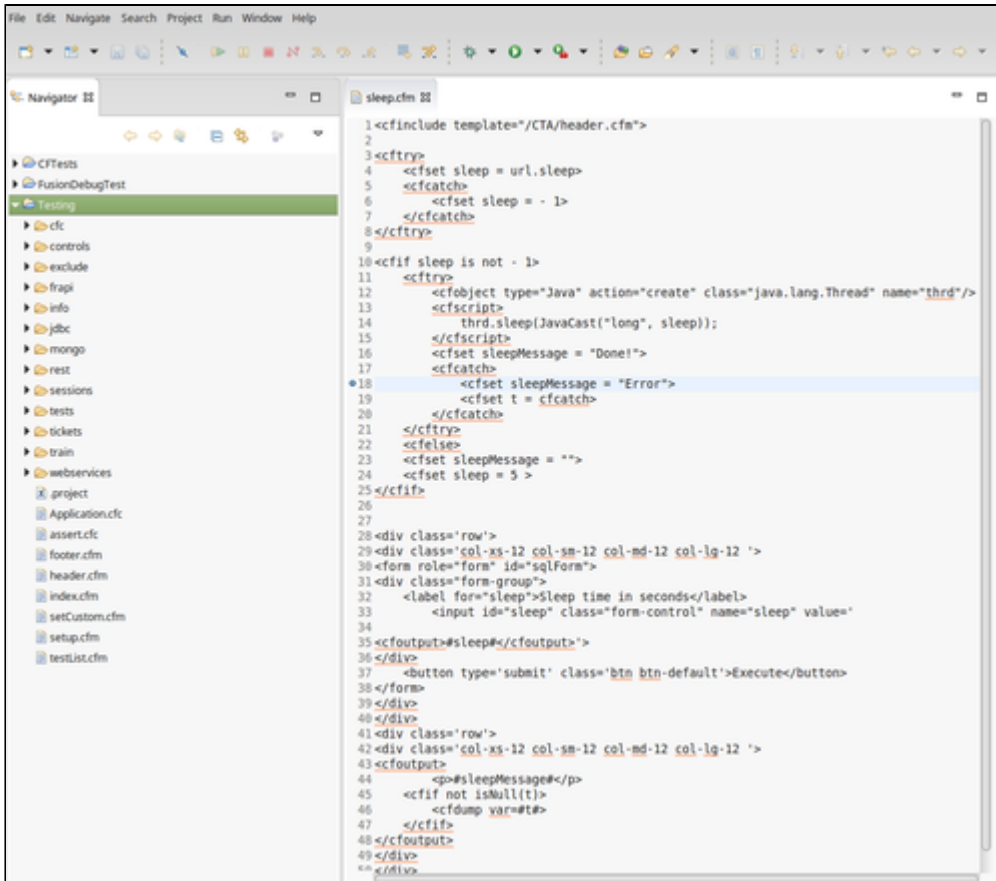


Figure 11: Navigator and current projects

An Eclipse Perspective is a collection of related Views. You can see that this

perspective contains (amongst others) the **Package Explorer** view on the left, and the **Outline View** on the right. Views can easily be dragged into different locations.

If you accidentally close a view, it can be reopened again by selecting **Windows > Show View**. If it doesn't appear in that menu, selecting **Other** will allow you to find the view in the complete list.

Installation of FusionDebug via the Eclipse Software Install Manager

Introduction

This section will guide users step-by-step through the process of installing FusionDebug in Eclipse via the Software Install Manager feature.

Getting Started

Once Eclipse has been installed and configured, you will be ready to install FusionDebug using the **Eclipse Install New Software Manager**. With the use of the Eclipse Manager, the process of **downloading**, **checking** and **installing** of FusionDebug will become easier in the future.

Please note that the installation process might be different if you are using different versions of Eclipse.

Installation Process

1. **Open Eclipse** and navigate to **Help > Install New Software**. A new dialog is going to appear, see Figure 12.

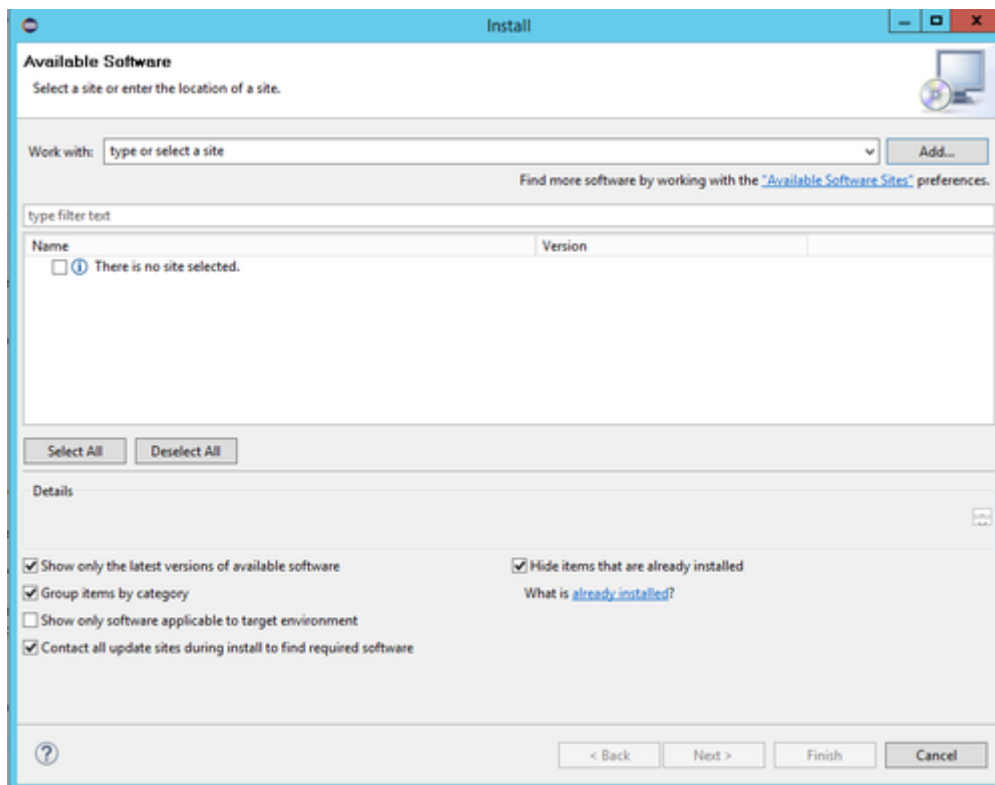


Figure 12: Eclipse Update Search Result Dialog without FusionDebug

- **Add** the Intergral FusionDebug software site by clicking the **Add** button. A new dialog is going to appear. See Figure 13.

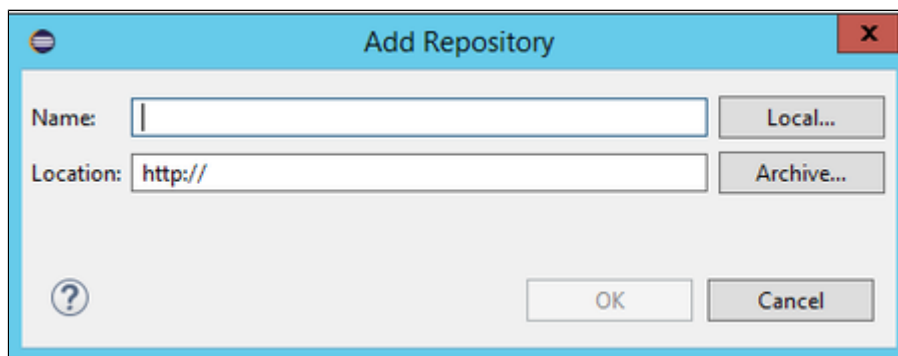


Figure 13: Add Repository Dialog

- **Name:** FusionDebug Update Site
- **URL:** <http://www.fusion-debug.com/wp-content/builds/latest>

Please make sure that the box in front of the FusionDebug is ticked and then press the Next button.

- Click the **OK** button and make sure the FusionDebug icon appeared. See Figure 14.

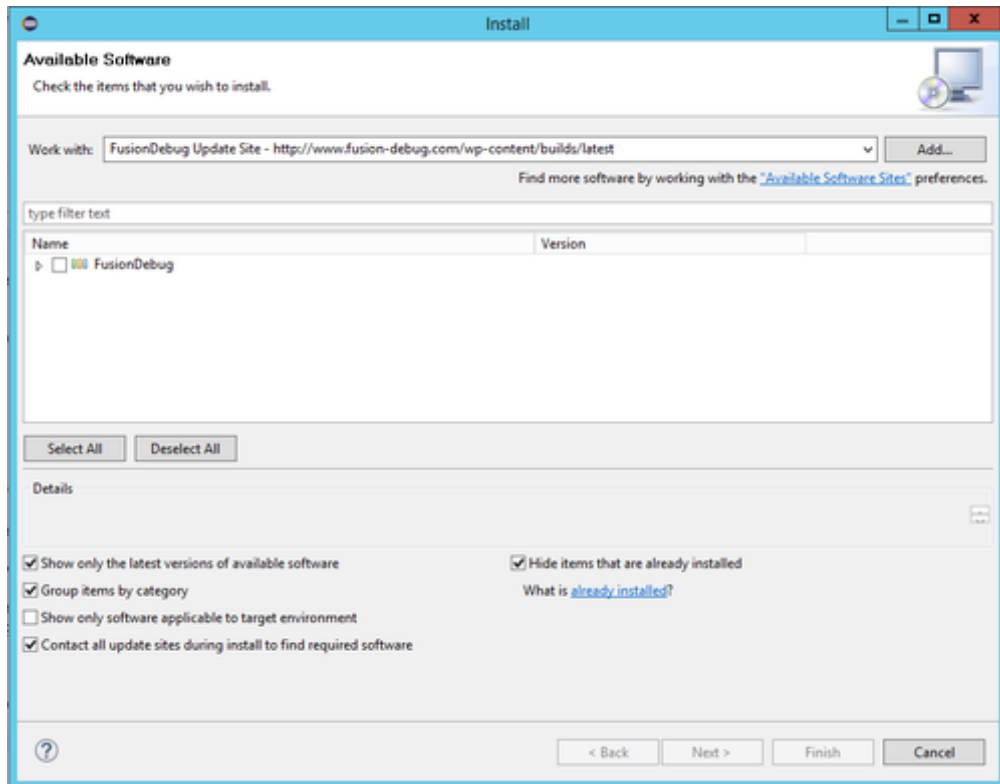


Figure 14: FusionDebug

Please make sure that the box in front of the FusionDebug is ticked and then press the Next button.

2. The Software Manager will connect to the fusion-reactor.com website and will locate the new software. A new "Install Details" dialog will appear, similar to that pictured in Figure 14. Please review the installation and click **Next**.

3. Read the license agreement which appears in the next step, if you accept its terms then please select "**I accept the terms of the license agreement**" and then click **Finish** to complete the process.

4. The software manager will download and install the package automatically (about 14 MB).

5. When the "**Security Warning**" dialog appears, click the **OK** button.

6. Eclipse will ask if it should restart the Workbench. Please press the "**Restart Now**" option to this question. Eclipse will restart automatically and then you should be able to use the FusionDebug.

Installing and Activating a License

Introduction

FusionDebug runs for **20 days** without requiring an active license. After this period, you will not be able to configure FusionDebug or launch any debug configurations until a valid license is installed and activated.

In order to get a valid license key, please visit our website and find all the information you need about pricing, FusionDebug pricing.

Installing a License

When you have received your license file, it can be installed as follows.

1. Open Eclipse and select **Window > Preferences**.
2. Open **FusionDebug** and select the **License** node.
3. Use the Browse button to locate your license file, and select Open.
4. Finally, click Install License.

You now have 10 days to activate this license.

If you have internet access from the machine you are using, we recommend immediately activating the license key.

Activating a License

Once you have installed a license file you will have **10 days to activate the license key**. We recommend activating the license key immediately and this can be done in two ways, **Automatically** or **Manually**.

The Figure below presents the FusionDebug License page after a license has been installed and activated.

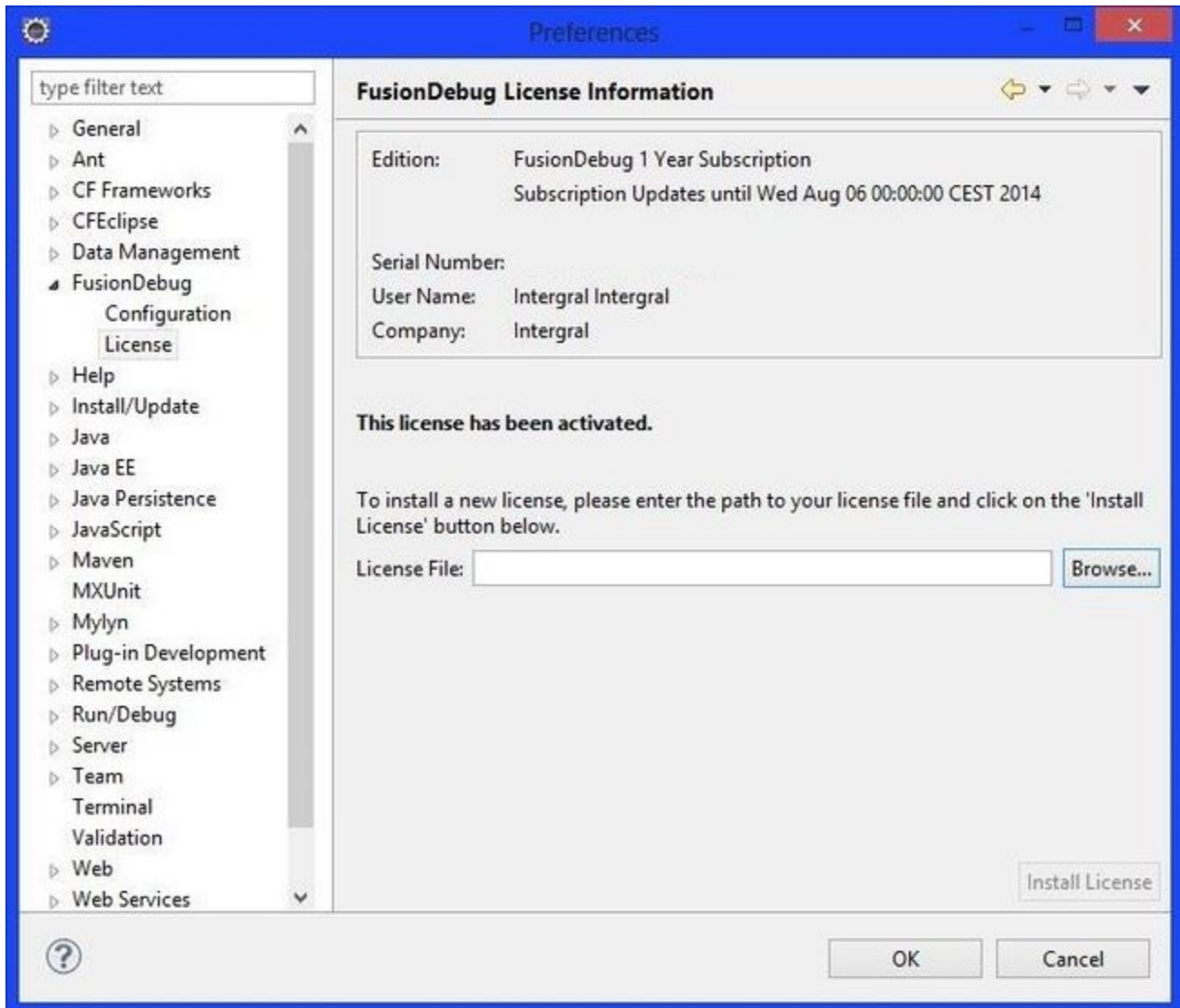


Figure 1: An Activated License Key

ACTIVATING A LICENSE AUTOMATICALLY

If you have internet access from the licensed machine, you can perform an automatic activation as follows;

1. Open the License panel as described above.
2. Click Activate License.

ACTIVATING A LICENSE MANUALLY

If you have not got internet access, a license key can be manually activated as follows;

1. **Open** the License panel as described above.
2. **Select** the **Manual Activation link**, and note down the Activation Input String.

Do not restart Eclipse until you have completed this procedure.

3. On a machine with internet access, visit the URL mentioned in the dialog.
4. **Enter the Activation Input String** into the form and select **Submit**.
5. When the page returns, note down the Activation Key.
6. On the machine to be licensed, **enter the Activation Key** into the dialog box.
7. Click **OK**.
8. Make sure the license key has been activated.

Configuring a Project in Eclipse

Introduction

Before Eclipse can be used to debug any ColdFusion application, you should configure a **new project** in the Navigator. This enables FusionDebug to locate the correct source code to respond to debugging events like breakpoints, steps etc. This section will guide readers step-by-step on how you can create a new Project in eclipse.

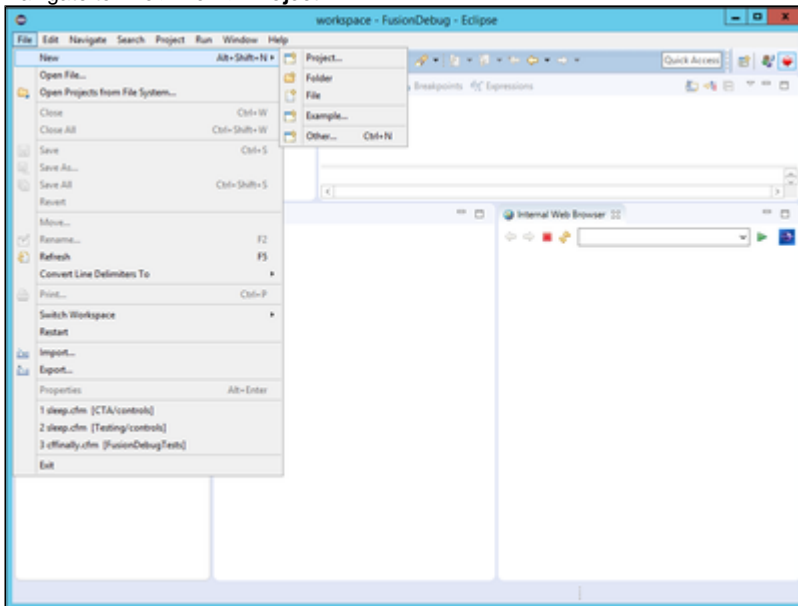
Creating a new Project in Eclipse

For this example, we are going to install a number test files inside the default IIS web folder and we are going to use these as the basis of our project. You can download the FusionDebugTest files from here, **FusionDebugTests.zip**

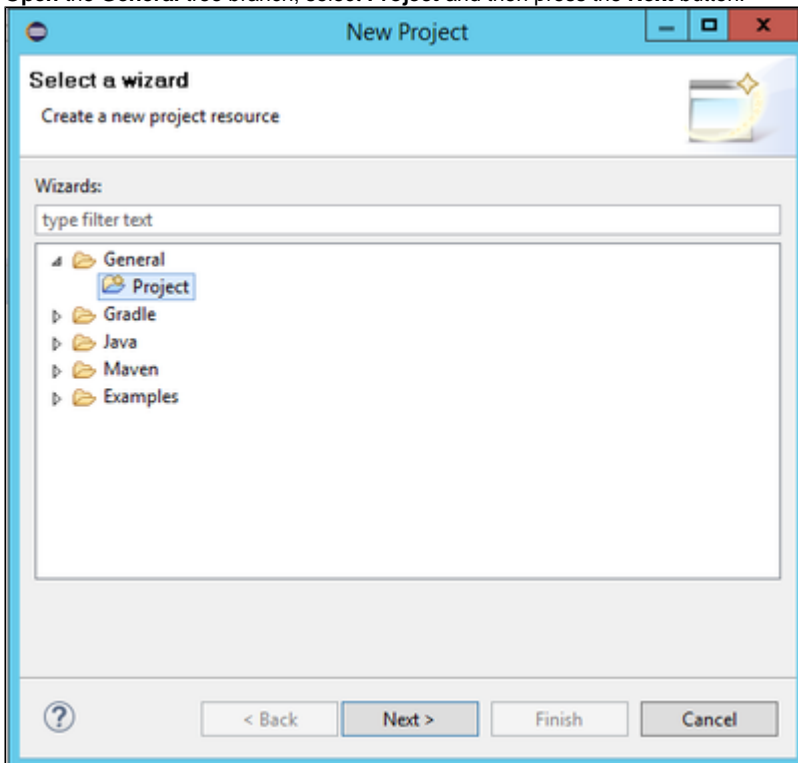
Please, follow the steps below in order to create a new project.

1. **Start** Eclipse if it's not already running, and **switch** to the **FusionDebug Perspective** if it does not open automatically (Eclipse remembers the last perspective you used before you shut it down last).

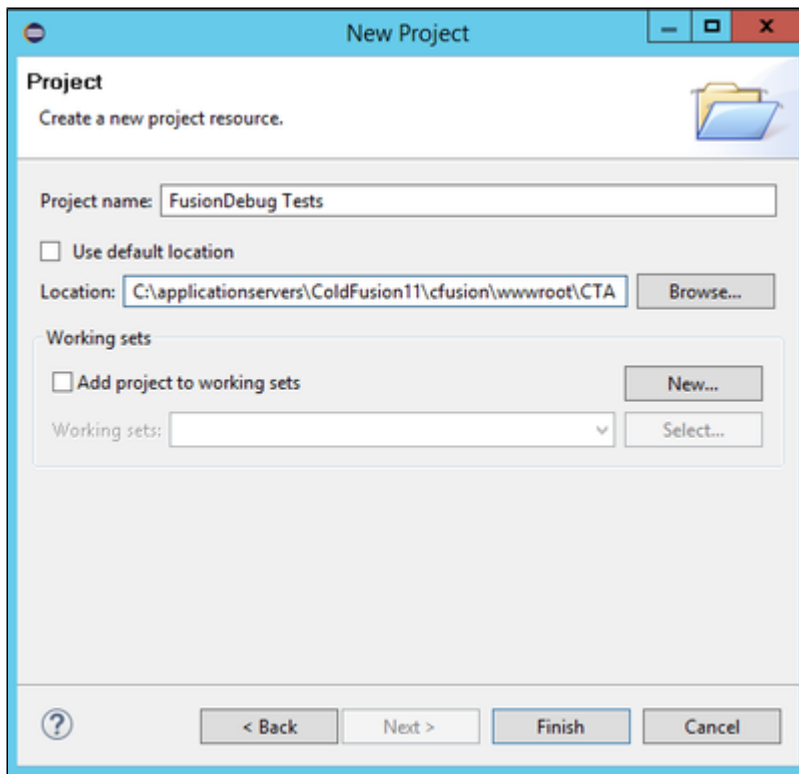
2. Navigate to **File > New > Project**.



3. A new wizard is going to appear.
4. **Open** the **General** tree branch, select **Project** and then press the **Next** button.

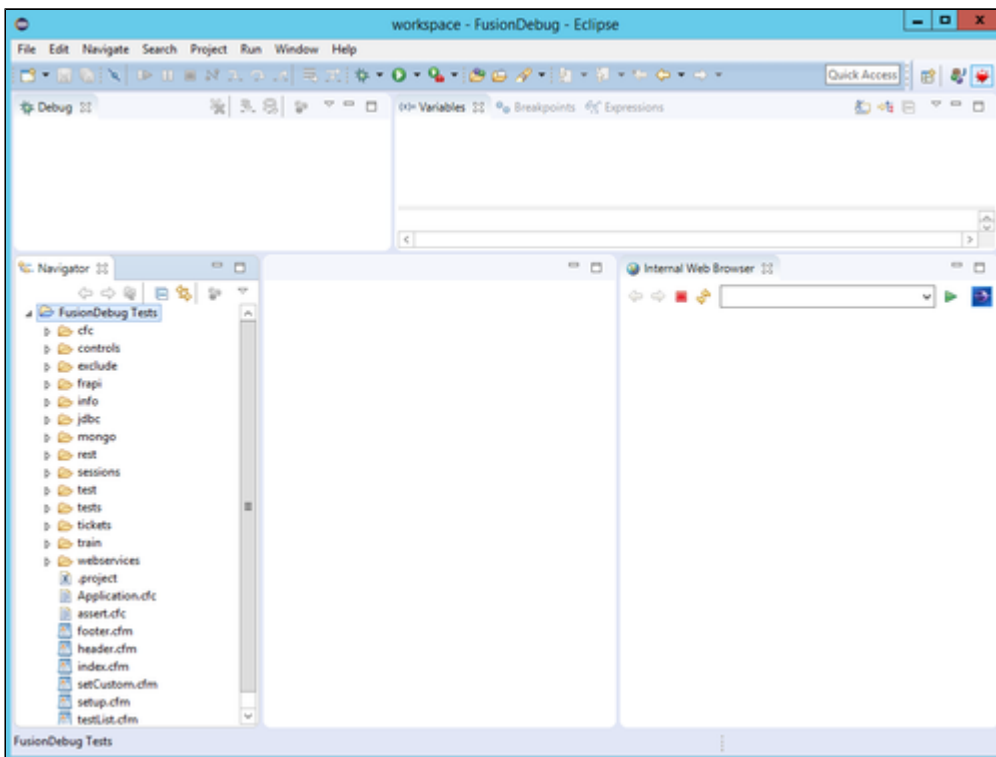


5. **Enter a meaningful name** for your project. For this example I have chosen the name "FusionDebug Tests". **Uncheck** the option "Use default location" and since we already have existing files, we can use these instead of starting a new project in the workspace.
6. **Locate** the folder containing the files by clicking the **Browse** button, and when you locate the files then click OK.
 - For this example we have used the following directory: C:\applicationservers\ColdFusion11\cfusion\wwwroot\CTA



7. Click the **Finish** button and complete the wizard.

After completing the wizard, the new project with name FusionDebug Tests is going to appear and if you double-click on any file within the project then you should be able to begin editing them. See screenshot below.




Do not delete the **.project** file – that's an Eclipse internal file used to keep track of your project.

Useful Links

- Create a FusionDebug Configuration

FusionDebug Perspective - Basic Introduction

Introduction

After the successful installation of **FusionDebug** within the Eclipse environment, a new Eclipse perspective will be available and is called "**FusionDebug**" Perspective OR you can simply identify the debug icon, . This can be found on the top right corner of the Eclipse environment. See Figure 14.

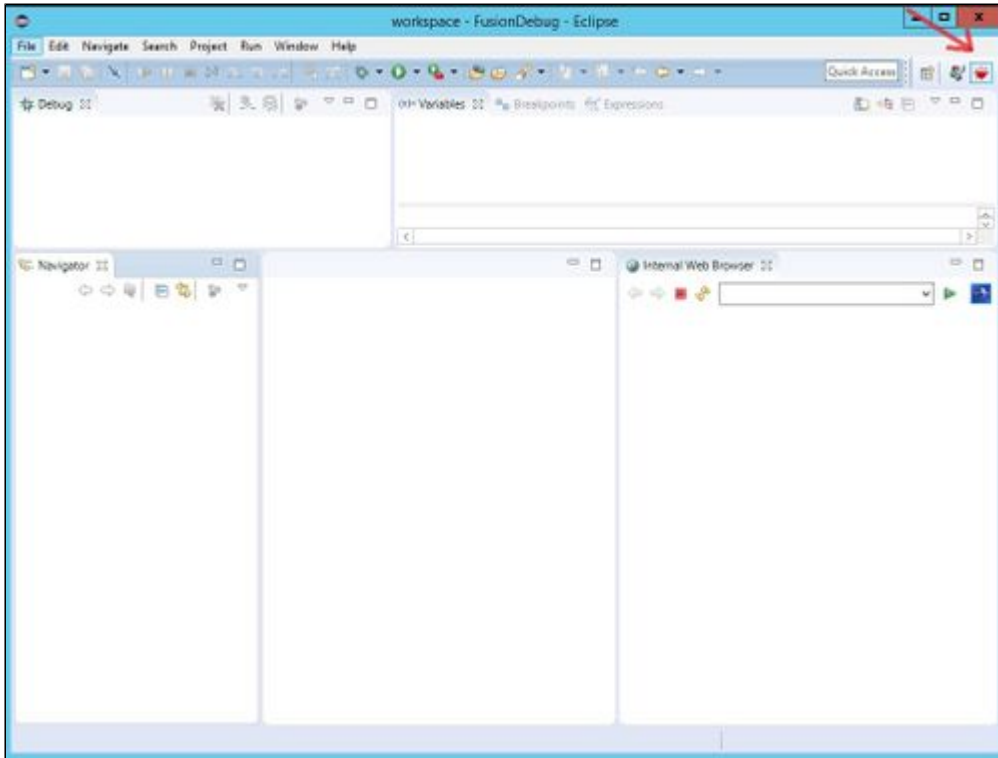
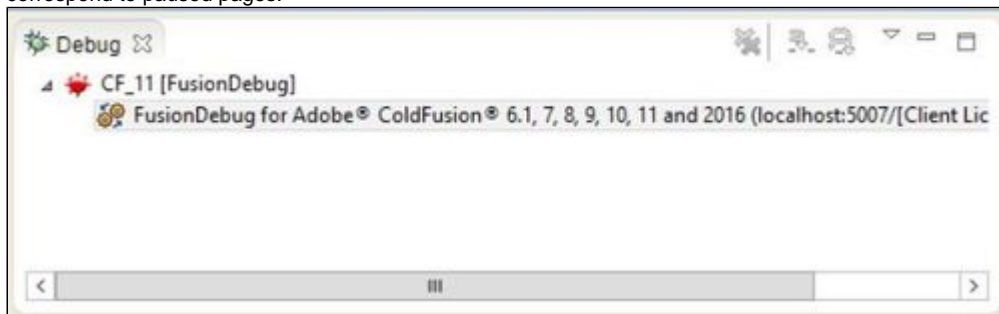


Figure 14: FusionDebug Perspective

This perspective is designed to get you start using the FusionDebug and start debugging ColdFusion pages. Broadly, they are as follows:

For the examples listed below I have used a ColdFusion 11 server, however, depending your environment you will have different data.

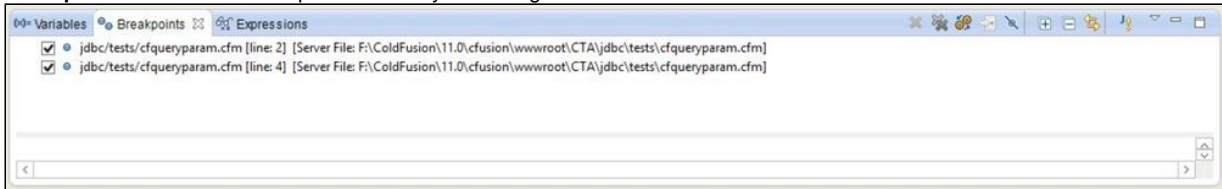
- **Debug View:** shows the current Debug Target to which FusionDebug is connected, along with any Stack Frames which correspond to paused pages.



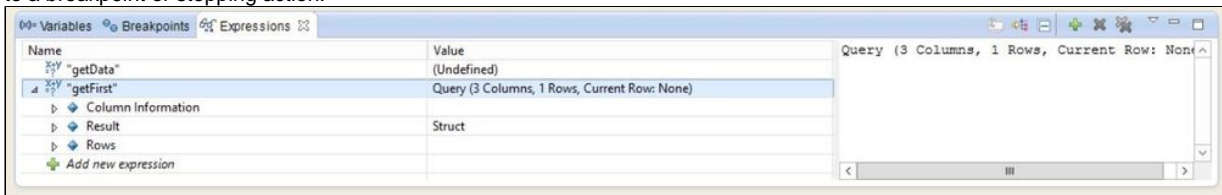
- **Variables:** appear when a page is **suspended**, this view shows all the ColdFusion scopes and variables currently available to the page.



- **Breakpoints:** lists all the breakpoints currently in use together with their exact location on the server.



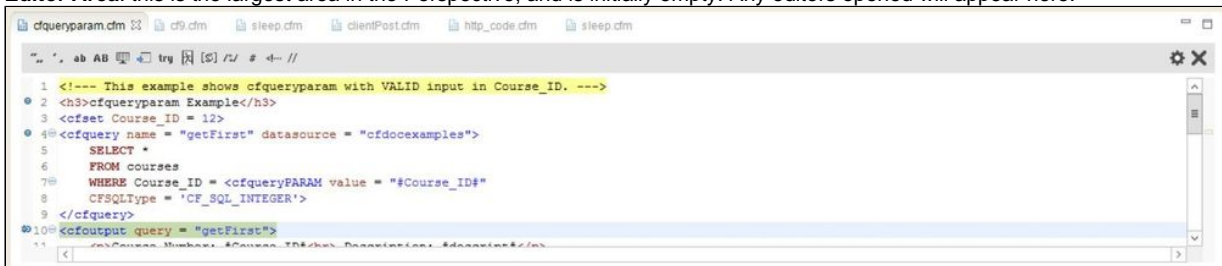
- **Expressions:** lists all the expressions currently being watched. This view is updated whenever a ColdFusion page pauses due to a breakpoint or stepping action.



- **Navigator:** shows all projects and folders currently in the workspace.



- **Editor Area:** this is the largest area in the Perspective, and is initially empty. Any editors opened will appear here.



- **Internal Web Browser:** this is a simple web browser and can be used to call pages and trigger breakpoints.



This perspective can be opened by selecting **Window > Open perspective > Other** and selecting the **FusionDebug** perspective.

- FusionDebug Feature Reference
- FusionDebug Concepts - Quick Tour

Getting Started - Configuration Example - ColdFusion 2016,Lucee 4.5.3

- Introduction
- Installation Process
- Activate a valid license key
- Open the FusionDebug Perspective
- Modify the JVM arguments file for debugging
 - ColdFusion 2016
 - Lucee 4.5.3
- Create a new Project that contains the source code you want to debug
- Create a FusionDebug Configuration
 - ColdFusion 2016 - FusionDebug Configuration
 - Lucee 4.5.3 - FusionDebug Configuration
- Create and Fire Breakpoints

Introduction

This documentation will provide readers with the information needed to setup and configure FusionDebug successfully. We will assume that readers have already downloaded FusionDebug in their environments.

For this example, we will use the following;

1. Windows Server 2012 R2 64 bit.
2. ColdFusion Builder 2016.
3. ColdFusion 2016.
4. Lucee 4.5.3 (with the latest underlying tomcat version)

Please note that **Lucee 4.5.4** and **Lucee 5.x** will not work with the current FusionDebug version. A fix for that issue will be available soon.

Please note that the following configuration can be applied to an Eclipse environment.

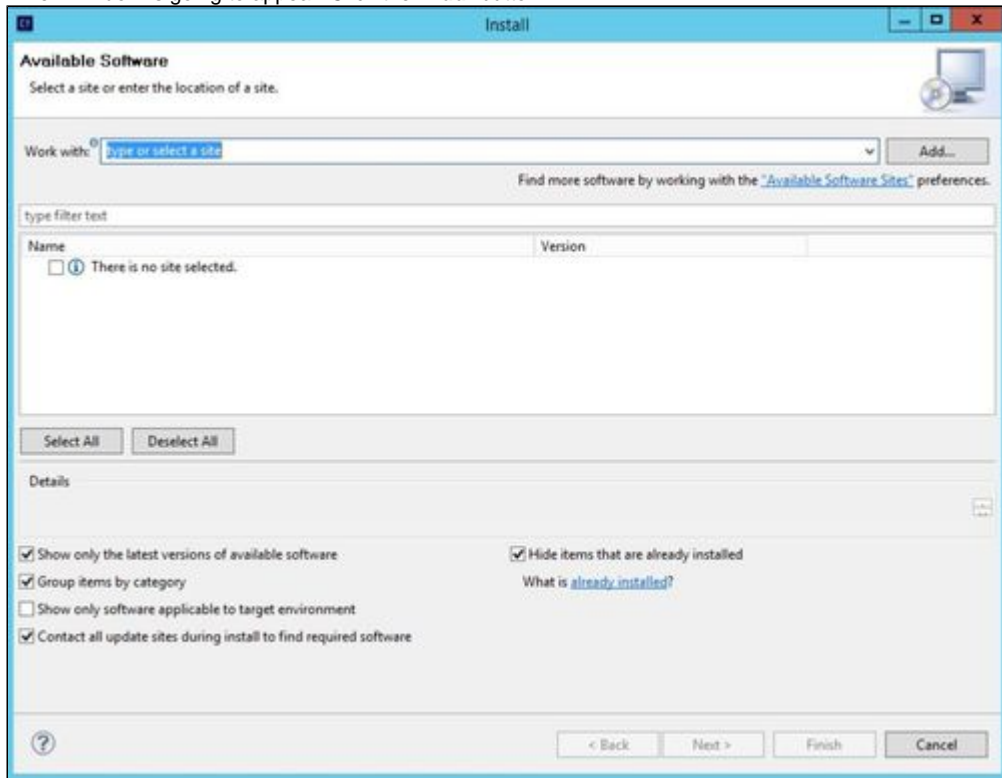
Installation Process

As we mentioned above, we will assume that you already have downloaded FusionDebug. In case you have not download the latest FusionDebug version, please access the FusionDebug website here to download a copy.

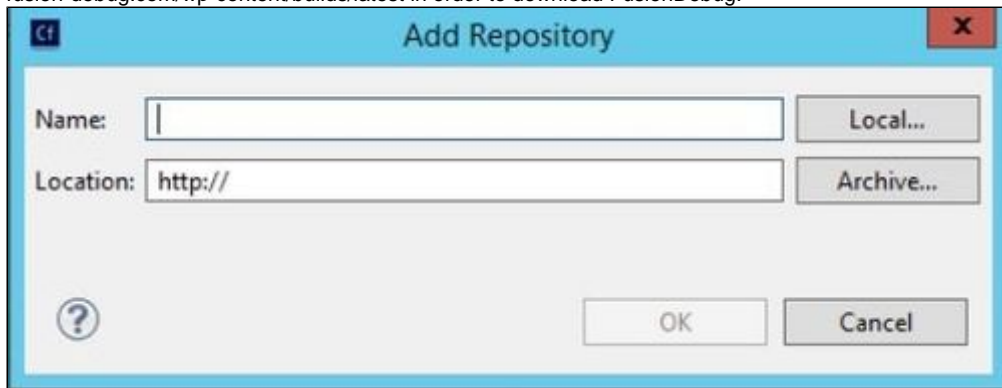
The first step, is to install FusionDebug via the ColdFusion Builder Software Install Manager. In order to do this, please follow the steps below.

1. **Open** the ColdFusion Builder 2016.
2. **Navigate** to Help > Install new Software.

3. A new window is going to appear. Click the "Add" button.

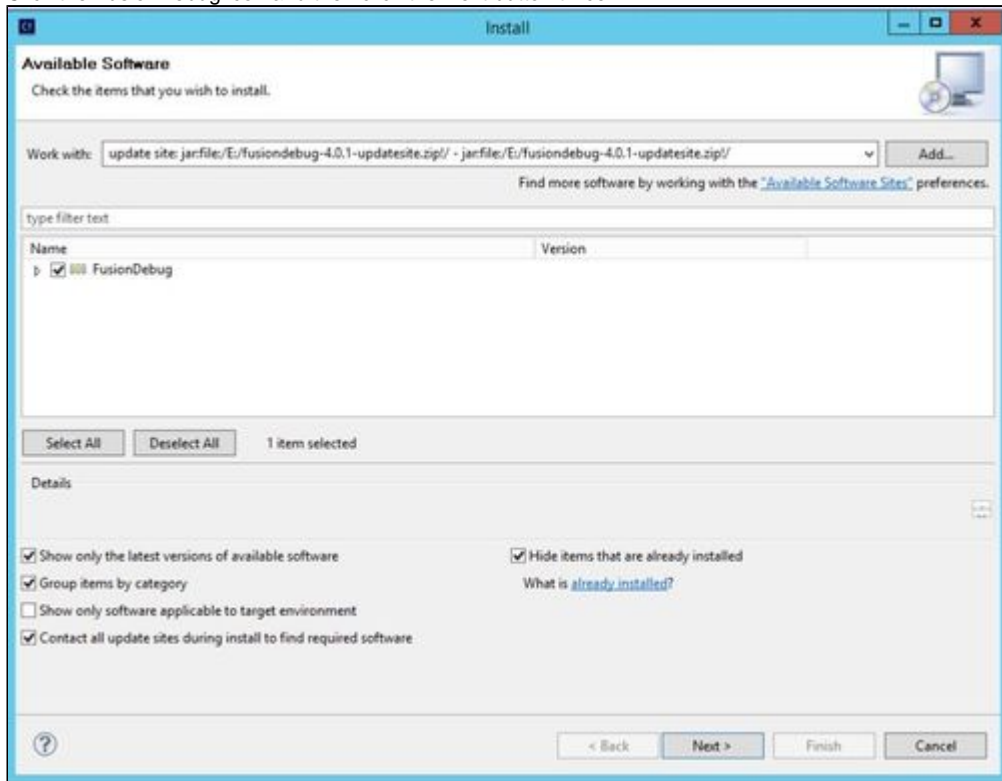


4. Click the "Local" or the "Archive" buttons if you have a copy in your local box, alternatively, add the following URL; <http://www.fusion-debug.com/wp-content/builds/latest> in order to download FusionDebug.

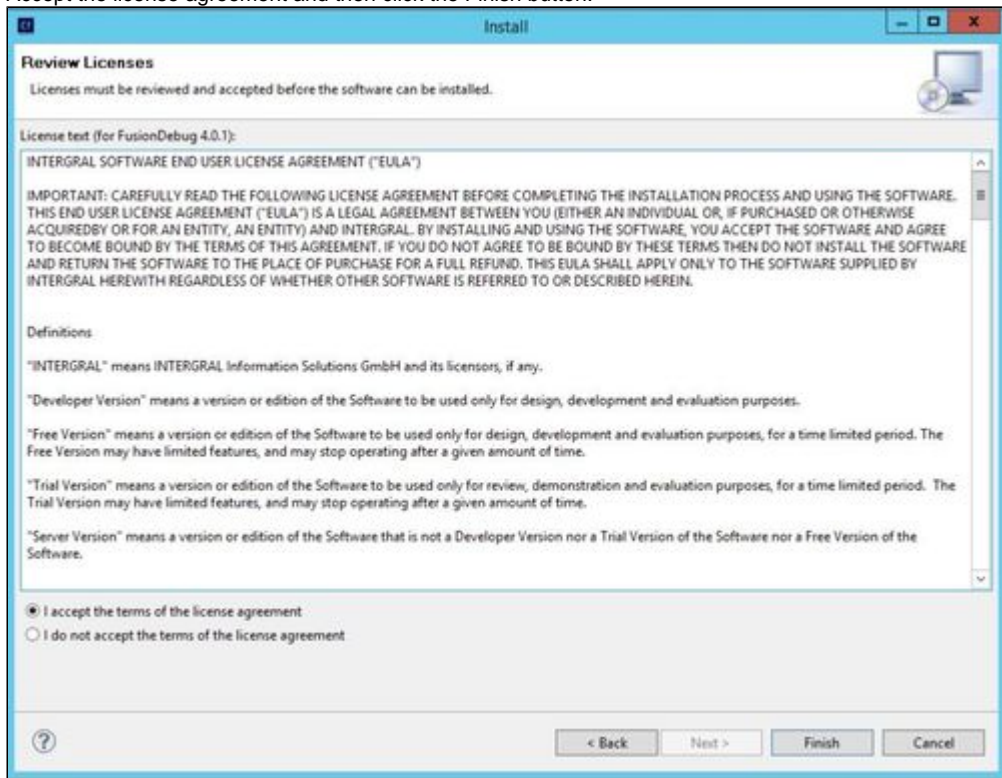


5. Click OK.

6. Click the FusionDebug icon and then click the Next button twice.



7. Accept the license agreement and then click the Finish button.



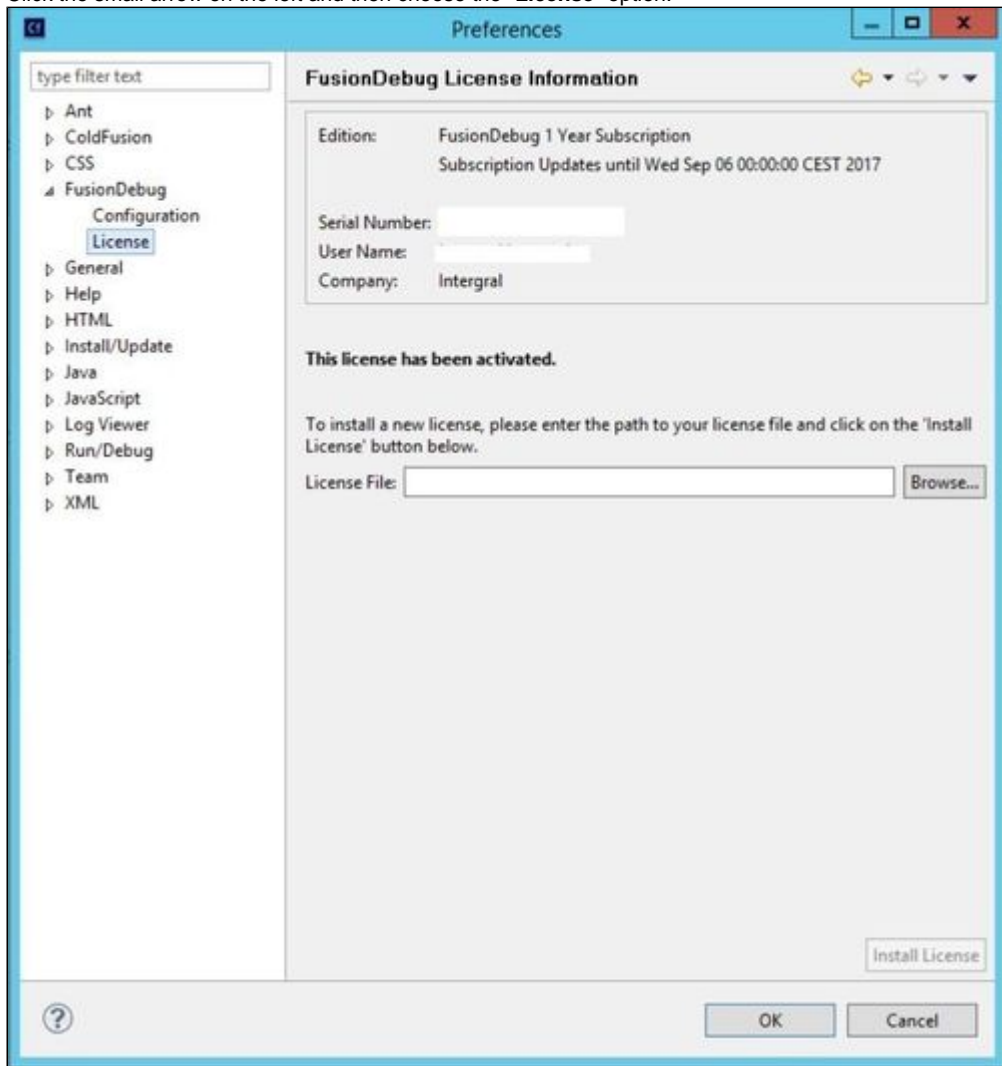
Make sure that FusionDebug has been installed successfully.

Activate a valid license key

In case you are using a trial license key with FusionDebug, you can skip that section.

If you have purchased a new license key for FusionDebug, please follow the steps described below.


1. **Start** the ColdFusion Builder if it is not running.
2. **Navigate** to **Window > Preferences > FusionDebug**.
3. Click the small arrow on the left and then choose the **"License"** option.



4. Click the **"Browse"** button and locate the license key file.
5. Click the **"Install License"** button.
6. Click the **"Activate License"** button.

Open the FusionDebug Perspective

After ColdFusion Builder is restarted, you are able to open the FusionDebug Perspective. This can be done by following the steps below.

1. **Start** the ColdFusion Builder if it is not running.
2. Click the **"Perspective"** icon which is located on the top right corner. 
3. Choose the FusionDebug option.
4. When this action is complete, in the left side of the window you will be able to see the FusionDebug area.

For more details about the FusionDebug view, please check out the following link, [FusionDebug Feature Reference](#).

Modify the JVM arguments file for debugging

DO NOT SKIP THIS STEP!

This is one of the most important steps in order to achieve a successful FusionDebug configuration without any issues. In order for FusionDebug to be able to read what line of code it is currently looking, the JVM needs to be put into debug mode. This procedure adds the required debugging parameters to a standard ColdFusion, Lucee/Railo installation.

Below, you will find two example configurations, one for the ColdFusion 2016 application server and one for the Lucee 4.5 application server.

ColdFusion 2016

1. **Stop** the ColdFusion server.
2. **Locate** the JVM arguments file. Usually, this can be found in the following directory; **{ColdFusion 2016 installation directory}/cfusion/bin/jvm.config**
3. **Open** the JVM arguments file with a text editor of your preference.
4. **Locate** the **java.args** line and locate the following line; **-Xdebug -Xrunjdw:transport=dt_socket,server=y,suspend=n,address=5005**.
5. if it exists it should be changed to match the following or if it does not then you need to insert the following after the java.args= - **Xdebug -Xrunjdw:transport=dt_socket,server=y,suspend=n,address=8000**. **Please make sure that the port is not used by another application in your system and make sure all the Java arguments are on the same line.**
6. **Save** the changes.
7. **Start** the ColdFusion server.

Lucee 4.5.3

1. **Stop** the Lucee 4.5 server.
2. **Locate** the JVM arguments file. Usually, this can be found in the following directory; **{Lucee 4.5 installation directory}/tomcat/bin/Luceew**. In case you are using a Unix OS, the Java arguments file can be found in the setenv.sh file.
3. **Open** the JVM arguments file with a text editor of your preference.
4. **Locate** the java.args and add the following line; **-Xrunjdw:transport=dt_socket,server=y,suspend=n,address=8001**. **Please make sure that the port is not used by another application in your system and make sure all the Java arguments are on the same line.**
5. **Save** the changes.
6. **Start** the Lucee 4.5 server.

More information about these configurations can be found in the following link, [Setting up Java for Debugging](#).

Create a new Project that contains the source code you want to debug

DO NOT SKIP THIS STEP!

In order to start debugging your code, first, you need to make sure that a new Project has been created and that Project contains the source code you want to debug. In order to achieve that, follow the steps listed below.

1. **Start** the ColdFusion Builder if it is not running.
2. **Navigate** to **File > New > Project**.
3. **Open** the **General** tree branch, select **Project** and then press the **Next** button.
4. **Enter a meaningful name** for your project. **Uncheck** the option "**Use default location**" and since you already have existing files, use these instead of starting a new project in the workspace.
5. **Locate** the folder containing the files by clicking the **Browse** button and when you locate the files, click **OK**. For the ColdFusion 2016 server, my code is located in the following directory; **C:\applicationserver\coldfusion\2016\cfusion\wwwroot\mycode** while for the Lucee 4.5 server, the code is located in the following directory; **C:\applicationserver\lucee4.5\tomcat\webapps\ROOT\mycode**.
6. Click the **Finish** button and complete the wizard.

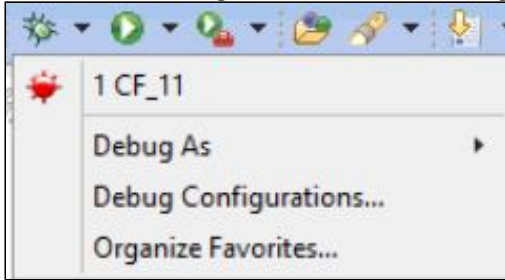
Create a FusionDebug Configuration

DO NOT SKIP THIS STEP!

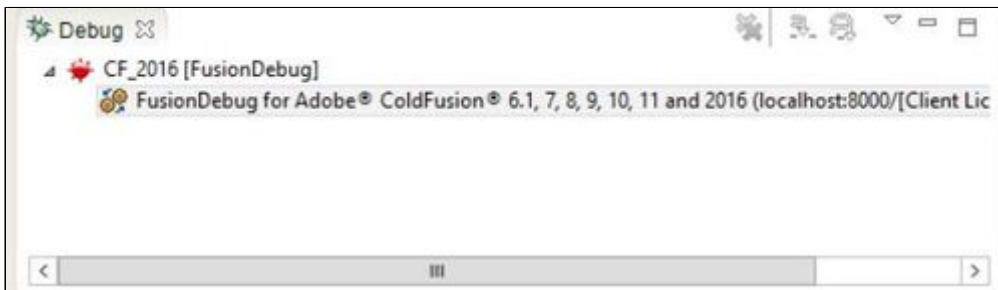
Another important configuration is the FusionDebug configuration which will allow you to debug your code. In order to create a new FusionDebug configuration, please follow the steps listed below.

ColdFusion 2016 - FusionDebug Configuration

1. Click the **FusionDebug** icon and choose the "**Debug Configurations**" option.

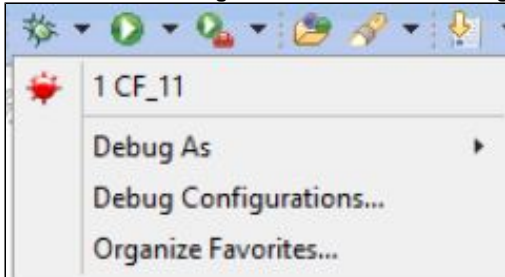


2. Locate the FusionDebug icon, right click on it and choose the "**New**" option.
3. Access the "**Connect**" tab and fill the fields accordingly. For this example, the ColdFusion 2016 server is on the same box as ColdFusion Builder 2016 (Host: localhost) and the debug port is set to 8000 (Port:8000).
4. Then, access the "**Source Code Lookup**" tab.
5. In the "**Source Code in the Eclipse Project**" specify the new project you have created in section "**Create a new Project that contains the source code you want to debug**".
6. Check the "**Server is local**" option and then browser to the directory where your code is located. For this example I have navigated the following directory; **C:\applicationserver\coldfusion2016\cfusion\wwwroot\mycode**
7. Click the **Add** button and then the **Apply** button.

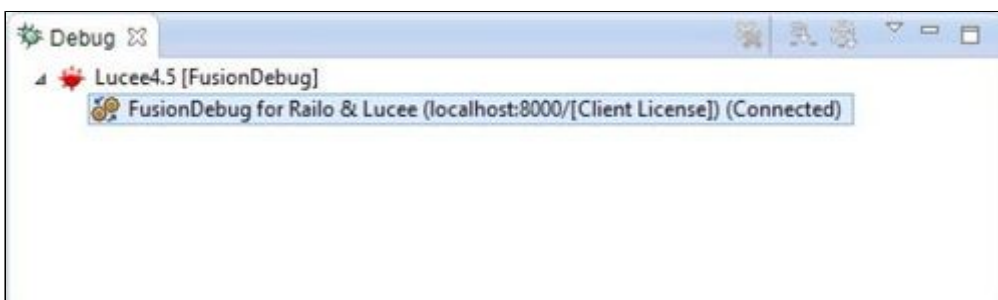


Lucee 4.5.3 - FusionDebug Configuration

1. Click the **FusionDebug** icon and choose the "**Debug Configurations**" option.



2. Locate the FusionDebug icon, right click on it and choose the "**New**" option.
3. Access the "**Connect**" tab and fill the fields accordingly. For this example, the Lucee 4.5 server is on the same box as ColdFusion Builder 2016 (Host: localhost) and the debug port is set to 8001 (Port:8001).
4. Then, access the "**Source Code Lookup**" tab.
5. In the "**Source Code in the Eclipse Project**" specify the new project you have created in section "**Create a new Project that contains the source code you want to debug**".
6. Check the "**Server is local**" option and then browser to the directory where your code is located. For this example I have navigated the following directory; **C:\applicationserver\lucee4.5\tomcat\webapps\ROOT\mycode**
7. Click the **Add** button and then the **Apply** button.



If the above configurations are correct and the source code mapping is correct, you should be able to achieve a successful connection with the ColdFusion 2016. Any FusionDebug connections can be found under the FusionDebug view.

In case you are not able to achieve a successful FusionDebug connection, please make sure 1. the FusionDebug Configuration is using the correct debug port, 2. The application server is running.

More details about the FusionDebug Configuration can be found in the following link, FusionDebug Configuration.

Create and Fire Breakpoints

All you need to know with regards to the Breakpoint configuration can be found in the following link, BreakPoints.

Uninstall FusionDebug via the Eclipse Software Install Manager

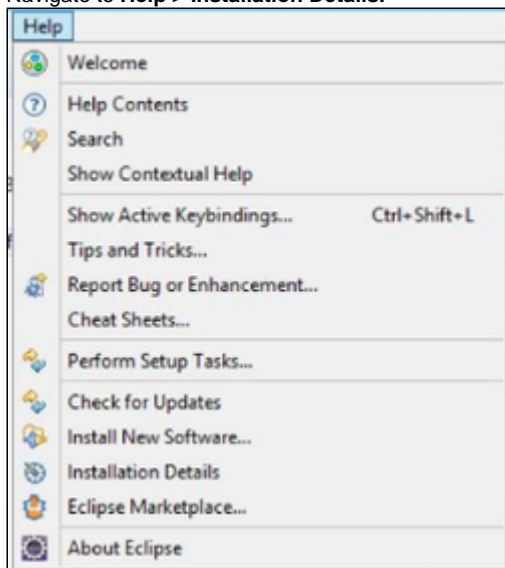
Introduction

This section will guide readers step-by-step through the process of uninstalling FusionDebug via the Eclipse Software Install Manager.

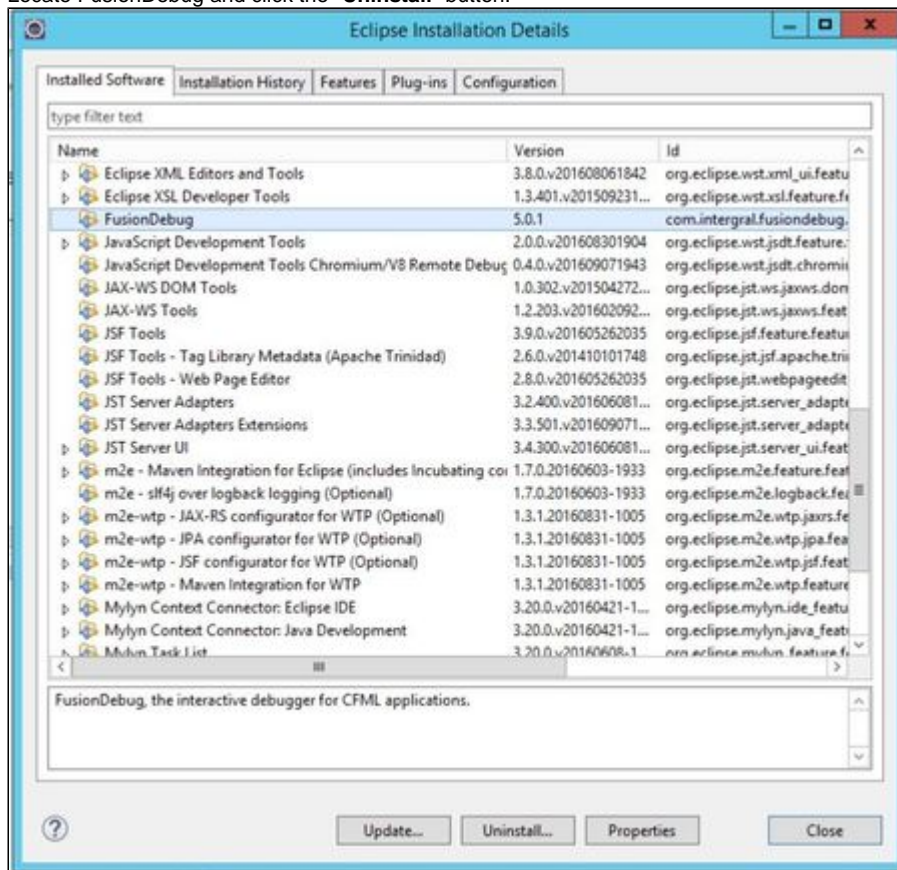
Uninstallation Process

If for any reason you would like to uninstall FusionDebug from your environment, please follow the steps listed above.

1. Open Eclipse if not running.
2. Navigate to **Help > Installation Details**.



3. Locate FusionDebug and click the "Uninstall" button.



4. The process will start.
5. In the "Uninstallation Details" page click the "Finish" button.

When Eclipse will be restarted, please make sure that FusionDebug has been successfully removed.

FusionDebug Concepts - Quick Tour

This section will guide readers through all the required concepts in order to get started with the FusionDebug. There are a number of useful screenshots that are used to show what each item in the FusionDebug perspective looks like and what each item does.

Child Pages

- Exploring FusionDebug
- FusionDebug Configuration
- Debug Target
- Threads
- Stack Frames and the Stack
- Variables
 - Change Value Dialog
- Expressions
- Breakpoints and the Current Instruction Pointer
- Stepping
 - Step Over
 - Step Into
 - Step Return
 - Resume
 - Auto-stepping
- Terminate
- Source Code Lookup

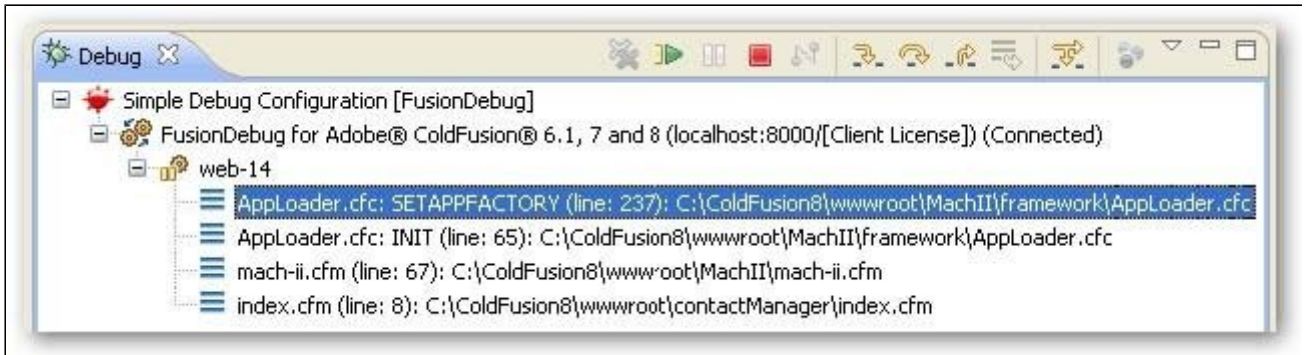
Exploring FusionDebug

Introduction

The section below is going to provide you with a quick overview of the **main features** and **concepts** of FusionDebug. For a detailed explanation of the main features and concepts, please check out the following link, FusionDebug Reference.

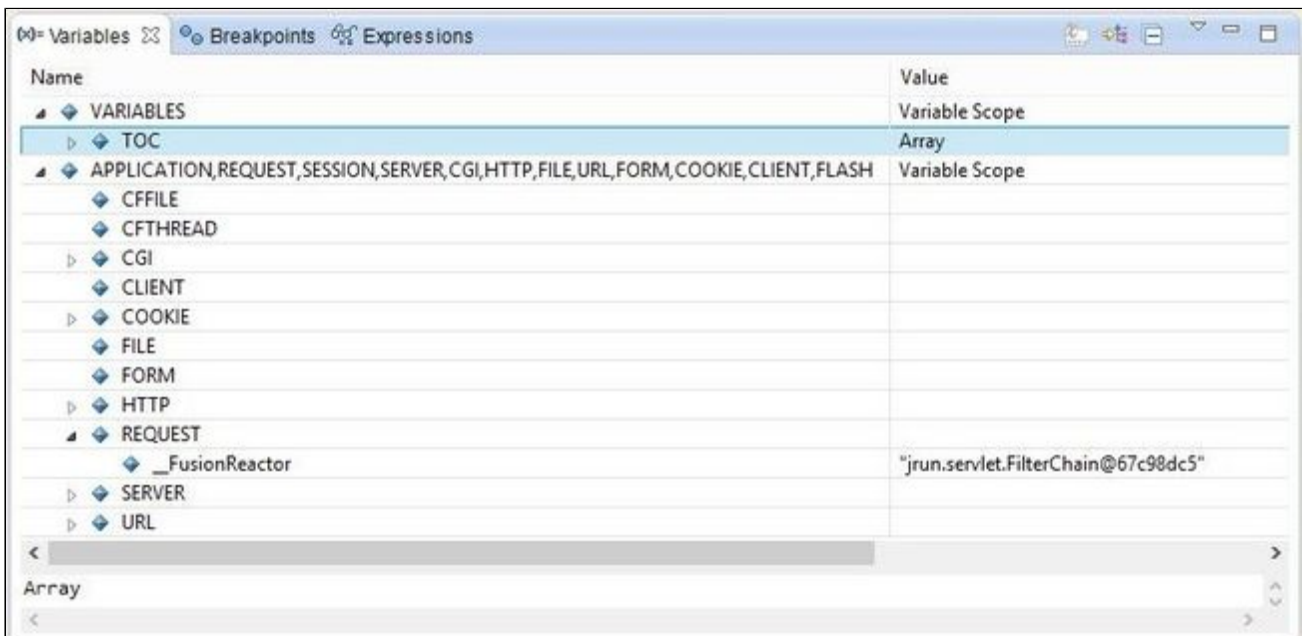
Main Features

EXPLORE STACKS



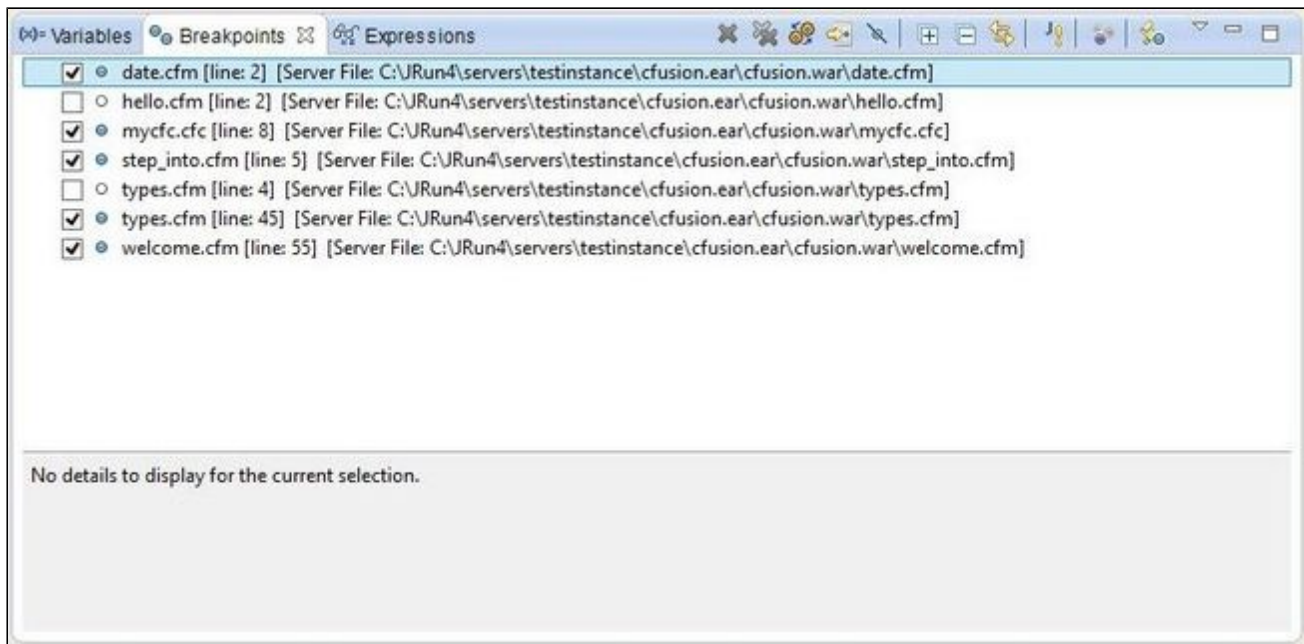
FusionDebug shows you **exactly** what **CFCs**, **pages** and **objects** are involved in your request. If you're inside a CFC function, the name of the function is also displayed.

EXPLORE VARIABLES



FusionDebug shows you all the ColdFusion variable scopes, including handy **Current Query**, **Local Variables** and **Function Local variables** too.

[EXPLORE BREAKPOINTS](#)



FusionDebug keeps track of all your project breakpoints, allowing you to **delete**, **disable** or **skip** them altogether. The **Breakpoint View** also shows exactly which server file the breakpoint is in.

[EXPLORE EXPRESSIONS](#)



You can add almost any ColdFusion expression and watch the value. The values are updated in real time, whenever a thread steps or pauses, so you can see exactly what ColdFusion sees. Additionally, you can **Inspect any variable** at any time by highlighting it in your code and selecting "Inspect Expression" from the right-click menu.

Useful Links

- FusionDebug Reference


FusionDebug Configuration

Introduction

A **Debug Configurations** is a collection of parameters which allow FusionDebug to **find, connect** and **use a Debug Target** – a CFML server instance – as well as enabling FusionDebug to look up for your source code whenever it needs to.

Getting Started

In order to access the **Debug Configurations**, please follow the steps below.

1. **Start** Eclipse if it's not started.
2. Press the small arrow new the debug icon, .
3. Choose the **Debug Configurations** option. A new windows will appear, see screenshot below.
4. Press **Right-Click** on the FusionDebug and press the **New** option. By doing that, a new Debug Configuration is going to be created.
5. Provide a Name for the new Debug Configuration or you can keep the default option.
6. In the **Connect** tab, fill all the details of your application server that you want to debug the code. When the changes are made, press the **Apply** button.

Please note that the **Port field** has the correct value and the port there is reflected to the Debug port. For more information about the Debug port configuration, please check out the link, [Setting up Java for Debugging](#).

7. Access the **Source Code Lookup** tab and configure there the source code of your application server. When the changes are made, press the **Add** button and then the **Apply** button. See screenshot below.
8. If you are happy with the configuration, then press the **Debug** button in order for the FusionDebug to start.

Please note that for every new debug configuration you make, you will see a source code lookup error at the top of the window. you will have to go into the "Source Code Lookup" tab to map your files.

Useful Links

- Configuration Example - ColdFusion 2016, Lucee 4.5.3

Debug Target

The **Debug Target** is the CFML server instance to which you want to connect and perform debugging.

The Debug Target is specified in the **Debug Configurations** dialog and appears in the **Debug View**.

In Figure 2, you can see the Debug Configurations (Simple Debug Configuration) and the Debug Target containing the Server Connector type, host, debug port, and license type (localhost:8000/[Client License]).

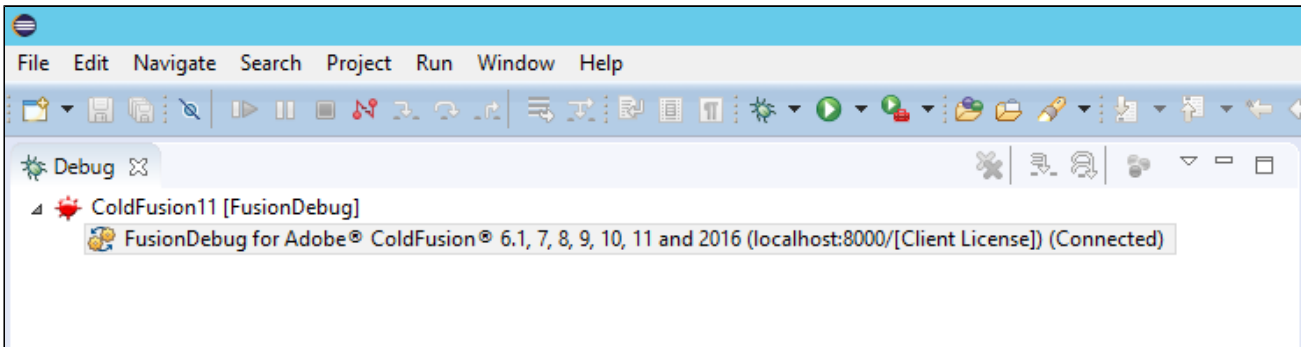


Figure 1: Debug Target

If you can not see the Debug Toolbar, click the drop down button and ensure "Show Debug Toolbar" is checked.

Useful Links

- FusionDebug Configuration

Threads

A **Thread** is a "worker unit" that runs ColdFusion pages.

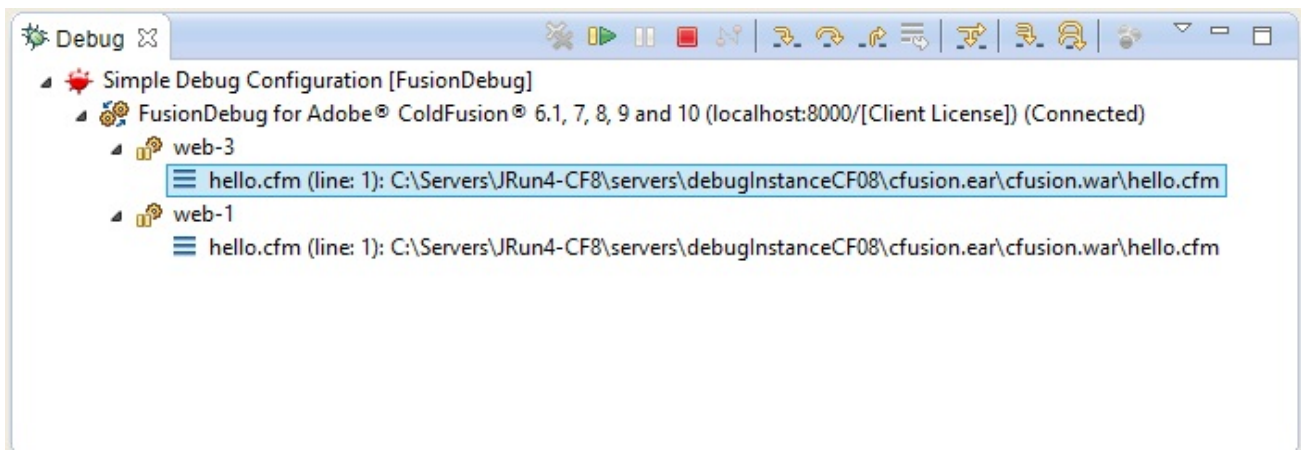


Figure 2: Showing multiple threads running at the same time

Threads are idle most of the time, sitting and waiting for web server connections. When a page is requested, the CFML server picks the next idle thread and gives it the request to execute. The thread's job is then to execute the script, performing any actions the script requires, then to return the generated page back to the web browser. While the thread is busy running a request, it can't handle another request.

When the thread's job is complete, it returns to idle state.

In ColdFusion, these worker threads are named **jrpp-X** or **web-X**, in Railo they are named **btpool0-X**, where **X** is a number (Doesn't need to increment by 1, as shown in Figure 2). You'll see them appear in the debugger as you learn to pause and inspect them.

In Figure 2 for example, we see that two threads are currently handling requests: **web-3** and **web-1**.

If you can not see the debug toolbar, click the drop down button and ensure "Show Debug Toolbar" is checked.

Stack Frames and the Stack

The **Stack** is a list of pages currently being **executed by a thread**, along with the current line number of that page. You will be able to see the stack in the **Debug View**. See Figure 3.

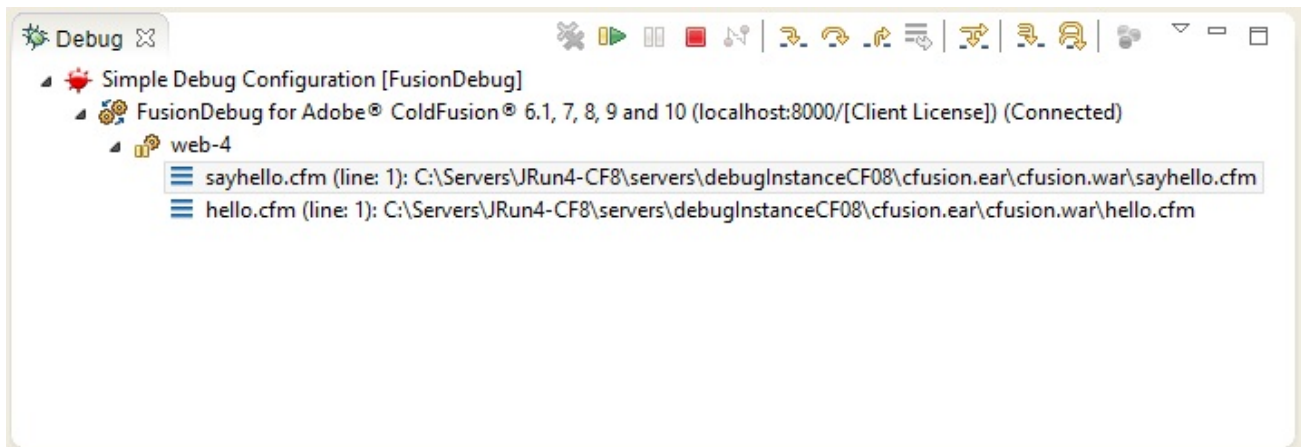


Figure 3: Stack With Two Frames

If you have only a single simple page, the stack will only contain one element – that page. If you have a more complex architecture, your stack may contain more elements.

For example, if you have a simple page – hello.cfm – and you run this page, the stack will contain just this page. If your page calls a sub-tag sayhello.cfm, the stack will contain two levels: hello.cfm and sayhello.cfm, see Figure 3.

We can see that the page which ran first - hello.cfm - is at the deepest level: the bottom of the stack. We call each of these levels a **Stack Frame**. The stack is composed of frames, which represent each page in the current call.

If you can not see the debug toolbar, click the drop down button and ensure "show debug toolbar" is checked.

Variables

When a thread is paused, the **Variables View** shows you all the ColdFusion variables currently available in the selected stack frame.

We are going to use the **sayhello.cfm** example, the **Variables View** would show all the ColdFusion scopes and variables available within that page.

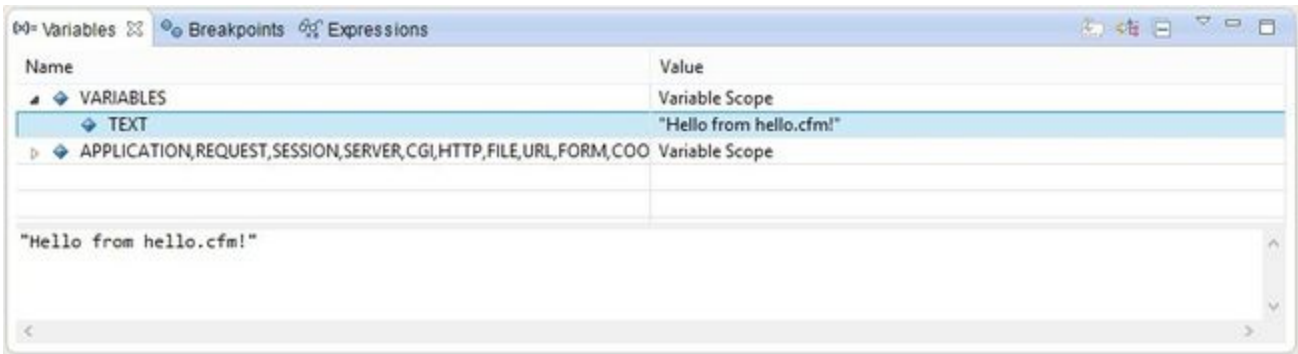


Figure 4: Variables View

Change Value Dialog

The **value** of a variable can be changed during **run-time** directly from the **Variables view**.

This can be done by right clicking on the variable you wish to change and then by selecting the **Change Value**. A dialog will display the full name of the variable as well as a text field to enter the new value of the variable, see Figure 6.

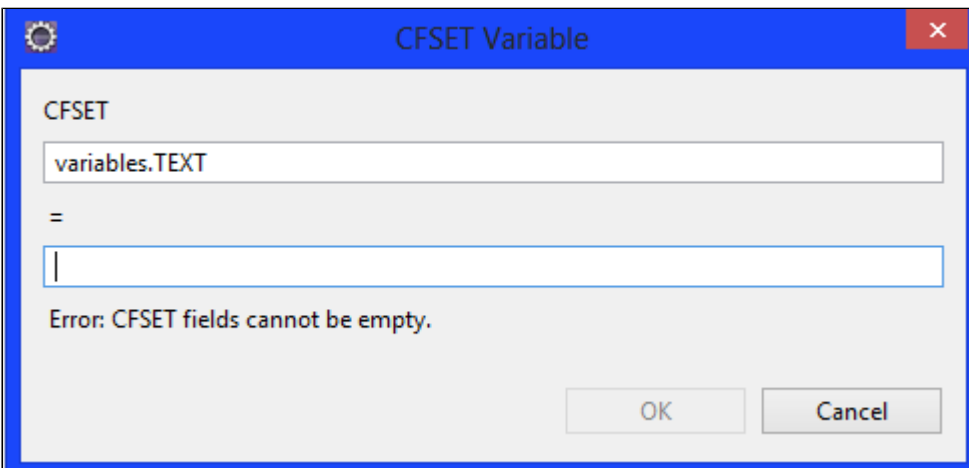


Figure 5: Modifying a Variable

The variable will be updated with the new value, if the value is valid, and by clicking the **OK** button when the changes have been implemented.

Expressions

The **Expressions View** allows you to carefully inspect just the variables you're currently focused on.

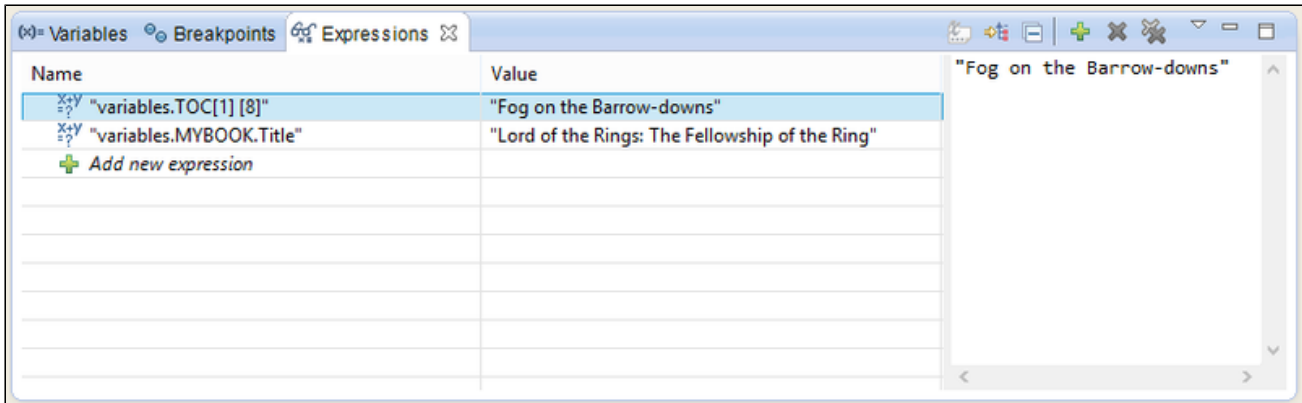


Figure 6: Expressions View

Expressions are variables that are specifically interested in watching, or expressions that you refer to frequently. The Expressions View is evaluated every time a **thread pauses**, so you can quickly see how the variables are changing during the execution of a page. Most ColdFusion expressions are valid in the Expressions View.

Figure 6 shows an Expression View inspecting two conditions of a variable "i".

Breakpoints and the Current Instruction Pointer

Introduction

Breakpoints are a **key feature** of FusionDebug. The breakpoints instruct FusionDebug to **pause any request** whenever a specific breakpoint is hit. Usually, we use the expressions "A breakpoint has been hit" or "A breakpoint has been fired", both of the expressions are equally correct.

Example

Breakpoints appear within Eclipse editors in the left margin as small **blue** dots. Figure 8 shows a breakpoint set on the `<cf_sayhello>` tag.

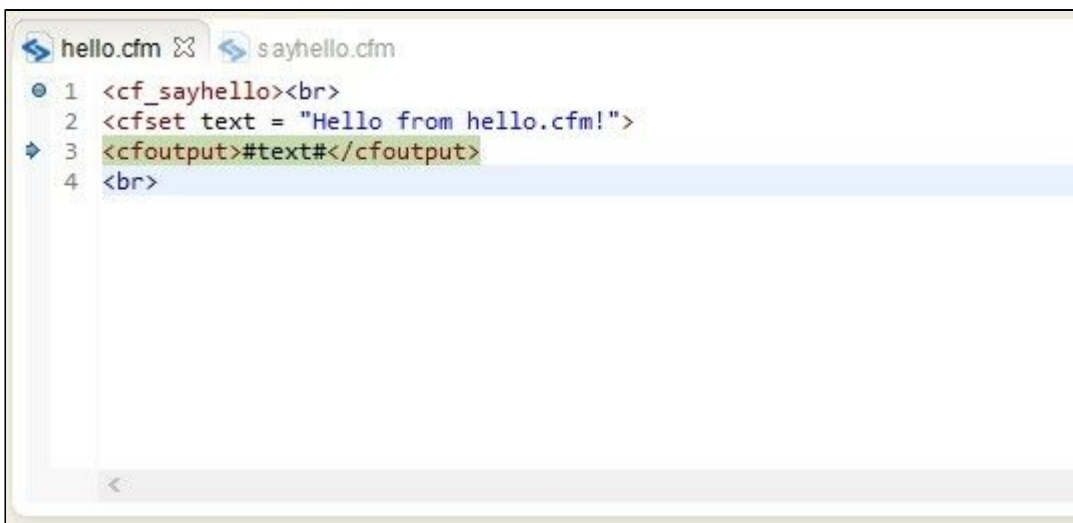


Figure 7: Breakpoints and the Current

Instruction Pointer

Please be aware that in order for a breakpoints to fire, the associated **project must be open**. Breakpoints within closed projects are disabled.

Breakpoints can be **enabled** or **disabled**. You can also **remove** them and **reset** them at any time and this is the only way to cause threads to pause inside FusionDebug. Once they are paused, the Variables and Expressions views come alive and allow you to examine the state of your application.

All current breakpoints are displayed in the **Breakpoints View**. In addition to the exact file and line, FusionDebug also displays the full path to the file. The full path is computed using the **Source Code Lookup** tab.

Current Instruction Pointer





The **Current Instruction Pointer** (CIP) is a small blue right-pointing arrow which appears in the left margin of the Eclipse editor. It shows you the line of code that ColdFusion is about to execute.

The CIP moves when you execute one of the Stepping commands to show you exactly which line will be executed next. Figure 8 shows the CIP on the <cfoutput> tag, on the line immediately below the breakpoint. The CIP is useful during **Stepping** operations to help you see what ColdFusion is executing.

Stepping

Once a thread has **hit a breakpoint** and **paused** then you are able to perform one out of four actions on it.

The actions are listed below:

1. Step Into 
2. Step Over 
3. Step Return 
4. Drop to Frame 

These actions are known as **Step** actions. They're accessible using the icons at the top of the Debug View, as shown in Figure 9.



Figure 8: Debug View with Debug Toolbar

If you can not see the debug toolbar, click the drop down button and ensure "show debug toolbar" is checked.

Step Over

The **Step Over** action simply tells FusionDebug to execute the line of code currently at the **Current Instruction Pointer** and no other action is performed.

If the **Current Instruction Pointer** is located at the last ColdFusion statement of a page and there are no other pages in the stack, the **Step Over** command is equivalent to the Resume command, and the page will run to completion. If there are further pages in the stack, **Step Over** will execute the command and return to the next page in the stack (the caller).

Step Into

The **Step Into** action is useful when the **Current Instruction Pointer** is located at a ColdFusion tag which would cause a **sub-tag** or **CF C to be executed**.

If you execute a **Step Into** command on such a tag, then FusionDebug will attempt to locate the source code for that page (or CFC) and step into it, pausing on the first line of that page. If the **Current Instruction Pointer** is not located on a sub-tag or CFC call, this command is equivalent to the **Step Over** command.

Step Return

The **Step Return** action is only available from within a sub-tag and a CFC method. This action causes FusionDebug to immediately finish executing the sub-tag or method, and return to the calling page, where execution will pause ready for further Step commands.

Resume

The **Resume** action will cause FusionDebug to start the execution of the page once again until any further breakpoints are hit OR the page has been completed.

Auto-stepping

Auto-step will automatically step through a page based on preferences that can be set in FusionDebug Preferences. You can set the step interval, which is the number of milliseconds between each step operation, for both stepping into or over your code.

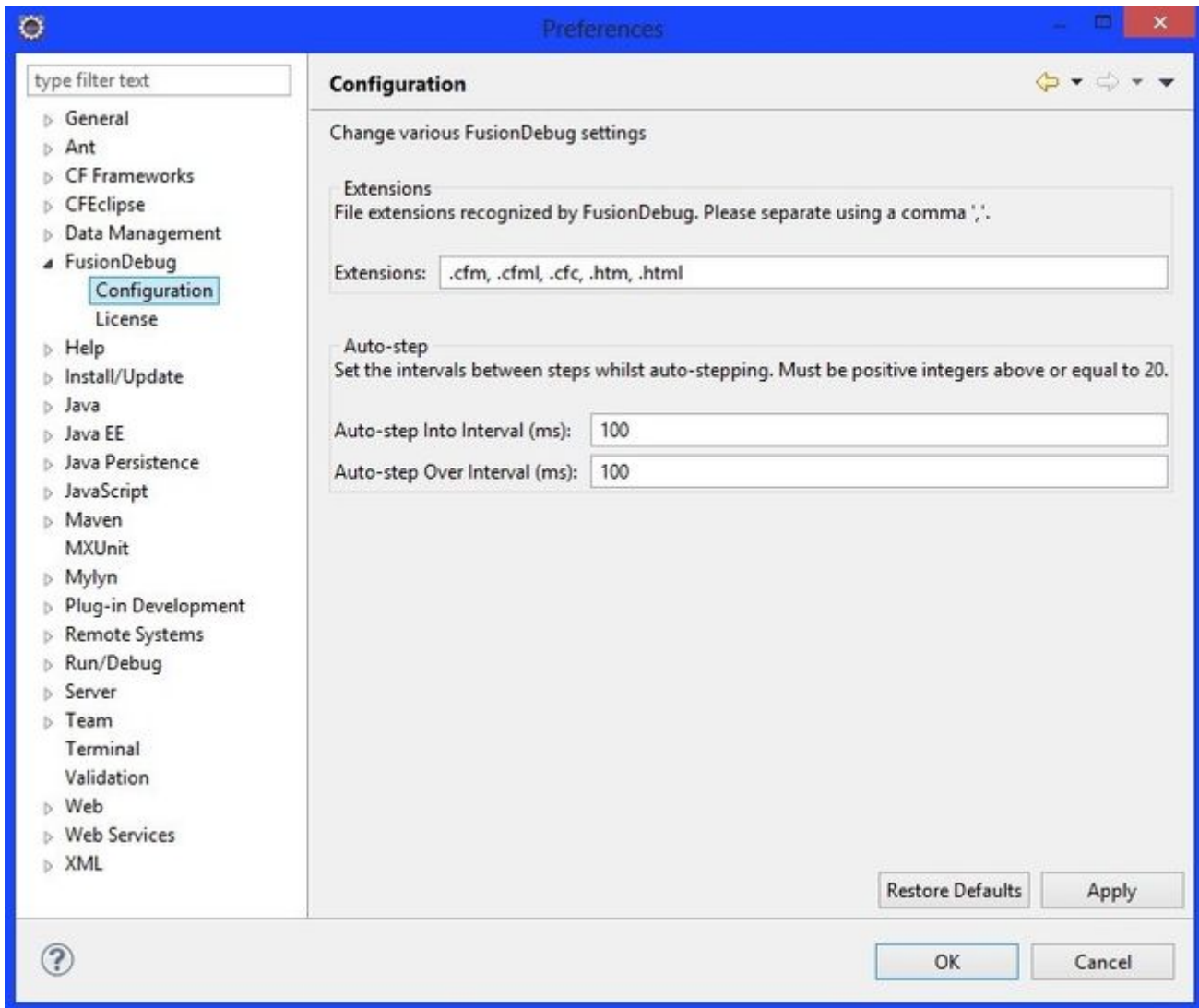


Figure 9: FusionDebug Configuration preference page

To use **Auto-step** first halt the page on a breakpoint, select the thread or stack frame and toggle **Auto-step** on using either of the Auto-stepping buttons in the Debug view toolbar or the entries in the right-click menu. You can stop Auto-stepping by toggling off a button or menu entry. The Into and Over variations of Auto-stepping are mutually exclusive so if you toggle Over while using the Auto-step Into, stepping into will stop and you will begin Auto-stepping over instead.

Auto-step is useful for watching variable / expression changes and execution paths. It can also be used to locate bottlenecks in code. Any line that doesn't step at the regular rate of the rest of the page is taking longer than the step interval to execute. This line can then be investigated.

Only one thread can be auto-stepped, so you must stop auto-stepping and then select another thread and start auto-stepping. The auto-step configuration can be changed whilst auto-stepping a thread. The changes will be reflected after applying the preferences.

Terminate



If a thread is **suspended**, the **Terminate** command causes it to be stopped completely. The terminate button is the following. This command can be used by selecting either a stack frame or a thread – in either case it is the thread itself which is terminated.

When terminated, no further execution will take place in the thread, including all pages, CFCs and objects associated with that request, and nothing further will be returned to the browser. This might be useful to cancel a thread which is about to do something destructive, or begin a time-consuming task.

Caveats

There are a couple of points to be aware of with the **Terminate** command.

- To be as reliable as possible, FusionDebug uses a very harsh method of halting the thread. This stops the request exactly at the point at which it's currently suspended, and cancels all further execution. If you have database transactions pending, they should be rolled back automatically, however, **Terminate** itself does not guarantee this.
- Terminate may not be effective during the following types of operations:
 - CF/CFC pages using Java objects via the <cfobject> tag, which are executing native methods (used for things like file and socket operations)
 - CF/CFC pages performing file operations themselves
 - CF/CFC pages waiting for data from a socket (e.g. gateway pages etc.)

In these situations, **Terminate** may either be delayed until these operations complete, or may not be effective at all. However since this command is only available when a thread is stopped, these situations should only arise rarely.

Source Code Lookup

When you set a breakpoint within a ColdFusion source file, FusionDebug must translate the breakpoint information into a form that ColdFusion can understand. To do this, FusionDebug must know exactly **where the page is on the server**. When a breakpoint fires, or a stepping action occurs, the reverse must also happen, meaning that FusionDebug must translate the debugging data from the server into a form that FusionDebug can use.

Pages can be addressed in many different ways (e.g. Web services, CFCs and subtags), and the **same name** may occur many times on your server (e.g. **Application.cfm**). When dealing with this name, FusionDebug needs an unambiguous way of telling which page is which.

The first way FusionDebug differentiates different pages is using the **Eclipse Folder Structure**, but that's not always enough because some projects have the same structure – framework-based projects especially. In order to provide another level of checking, FusionDebug needs to know **where** on your server these pages are and to **which** Eclipse project these paths are associated.

The **Source Code Lookup** tab allows you to enter one or more **Lookups**. These **lookups** enable FusionDebug to accurately find files when you set a breakpoint and when it needs to open a source file, for example during a **Step Into** operation. It is worth stating that the **Source Code Lookup** system is explained in much more detail later sections.

- Source Code Lookup

The Source Code Lookup Tab

Introduction

The Source Lookup tab is used to tell FusionDebug where the source files in your Eclipse projects are placed on to your CFML server.

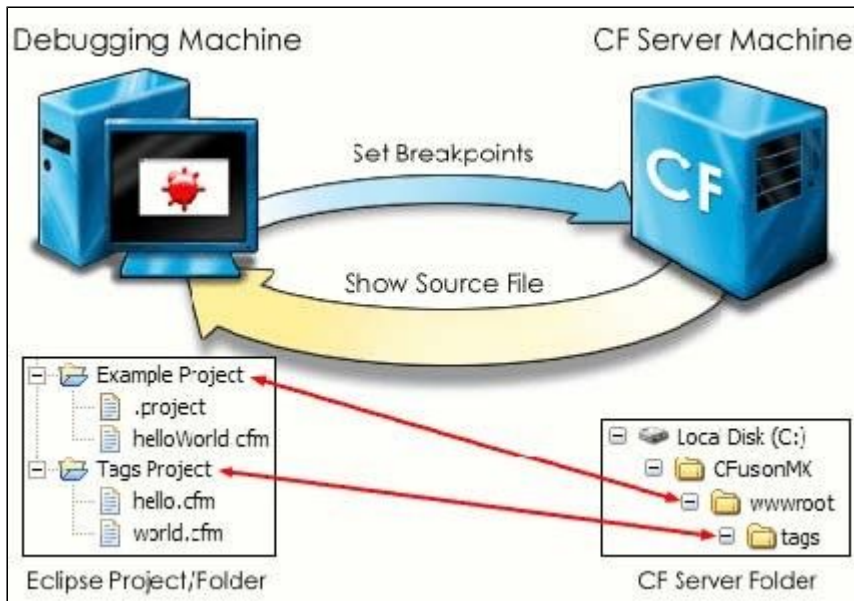


Figure 9: How Lookups are Used

FusionDebug uses the source code lookup rules to tell the CFML server where to set

breakpoints and to tell Eclipse which source code to display when a breakpoint is hit. Breakpoints are set within Eclipse but must also be transferred to the CFML server for them to become active.

By configuring your Source Code Lookups, FusionDebug can support multiple Eclipse projects, mapped CFCs and include files and provide support for the various frameworks that can be used with CF.

What's new?

This has had a slight upgrade since FusionDebug 3.5, in that you can now browse for your server files on your local machine. Simply click the "Browse" button and navigate through the hierarchy until you find the CFML folder that contains your .cfm or .cfc files of the server you wish to Debug. Of course, the path to the CFML folder can still be entered manually if you would prefer.

Child Pages

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

What is a Lookup?

Often, the structure of the files in your Eclipse projects is not the same as the structure when deployed to your CFML server. This leads to the problem that FusionDebug is unable to work out which files in Eclipse related to which files on your CFML server. The issue is solved by creating **Source Code Lookups** in your FusionDebug configuration. Each lookup consists of an **Eclipse project path** and the **absolute path** to the folder on the CFML server that contains the same files.

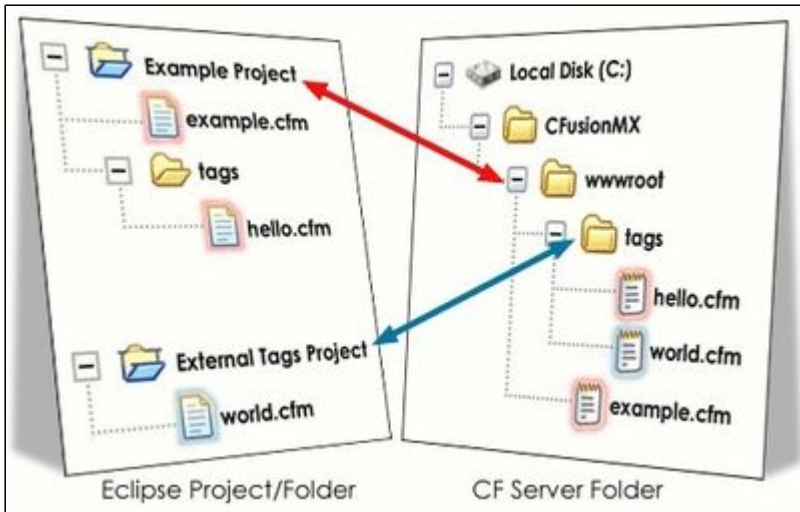


Figure 10: How Lookups are Used

If you take the Eclipse Project/Folder paths and absolute CFML server paths from Figure 11, then your Source Code Lookups will look like this:

Eclipse Project/Folder	CF Server Folder
Example Project	C:\CFusionMX\wwwroot
External Tags Project	C:\CFusionMX\wwwroot\tags

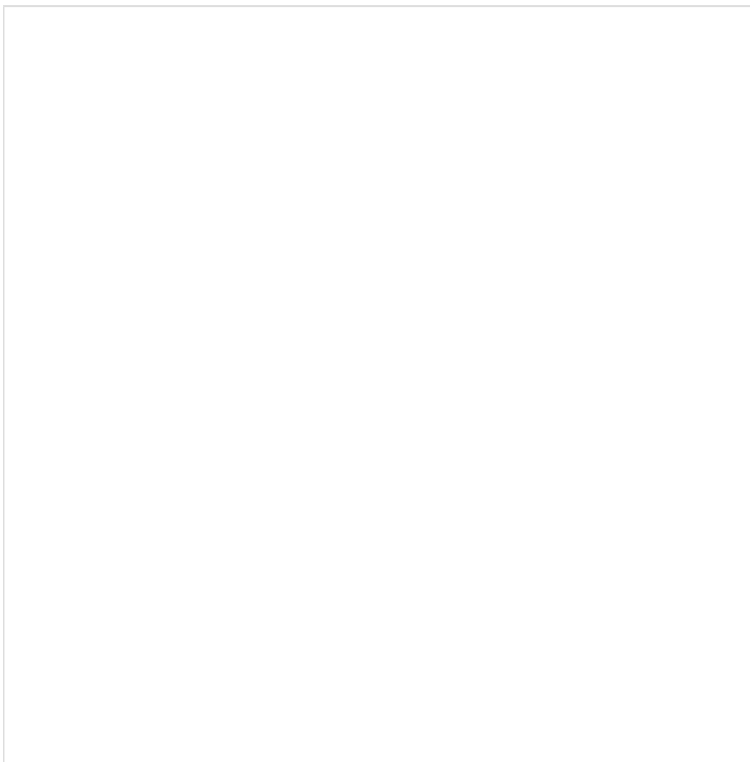
Figure:11 Setting Lookups for the Example

How to find the Source Code Lookup Tab

To get to the Source Code Lookup Tab (Which is the third of 4 tabs you will see in order to set up your FusionDebug connection), you have 2 main options:

Option 1

Navigate to **Run > Debug Configurations** in Eclipse. See Figure 12 below.



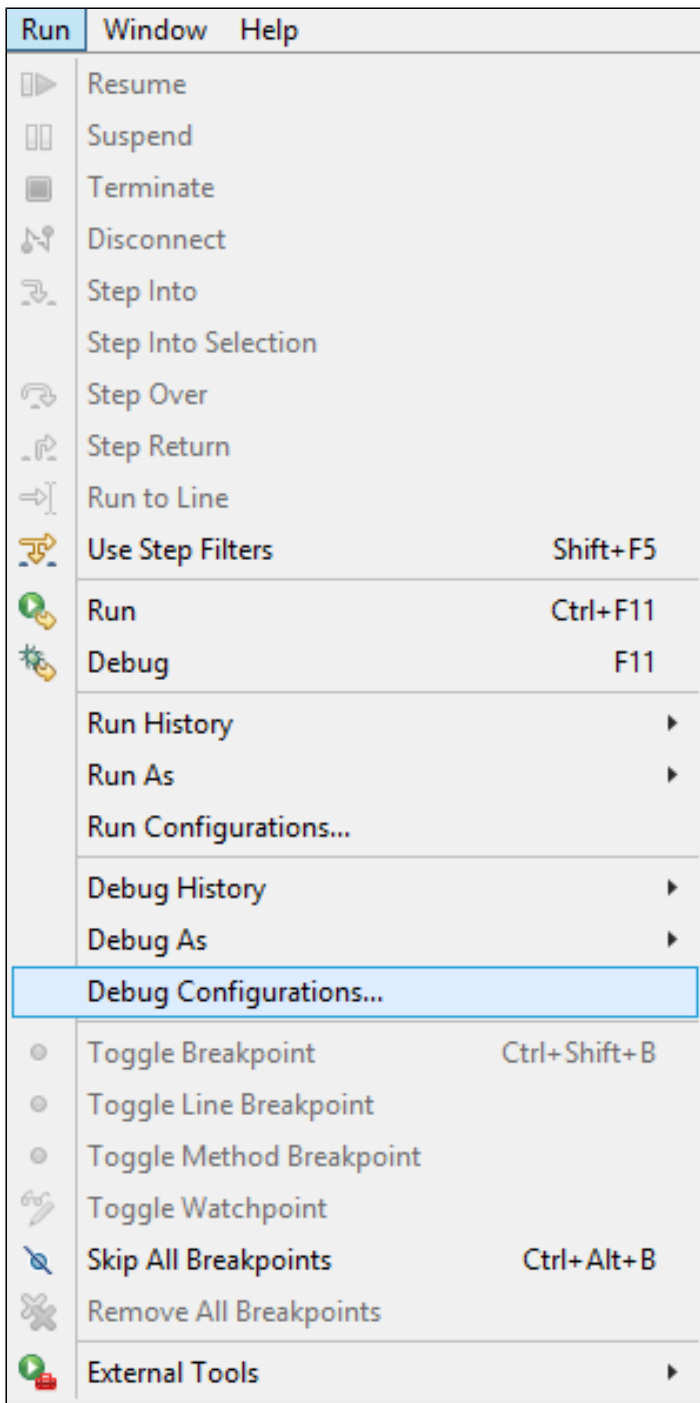

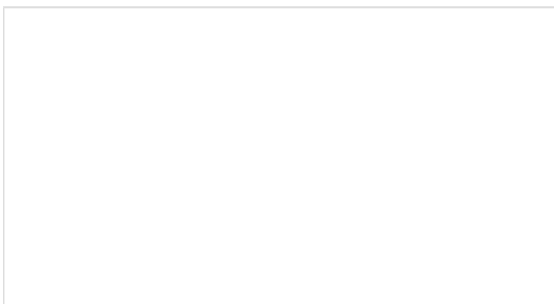


Figure 12: The Run > Debug Menu

Option 2



Alternatively, you can go to the green debug icon,  in your Eclipse toolbar and click the small arrow on its right-hand side. This brings up a small menu and you can select the Debug Configurations item from there.



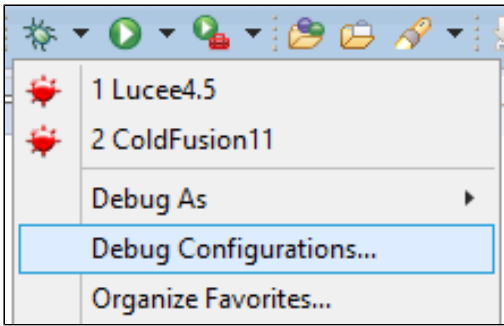


Figure 13: The Eclipse Toolbar Debug Dropdown

Debug Configuration

Once the debug dialog is visible you can bring up the Source Code Lookup tab by opening the FusionDebug item and clicking on an existing debug profile, or by right-clicking on the FusionDebug item and selecting New from the menu.

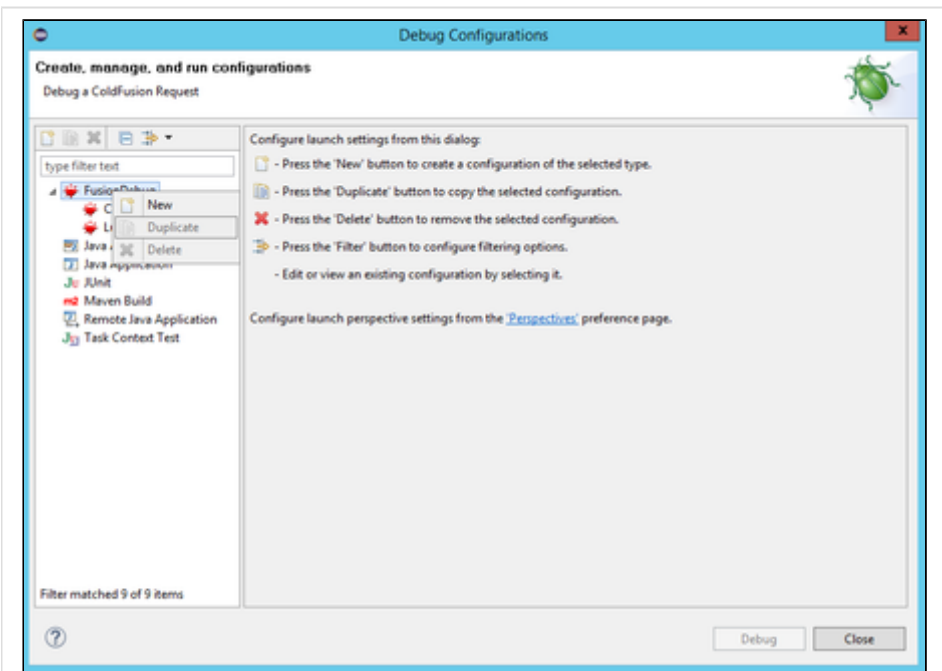


Figure 14: Creating a New FusionDebug Configuration

How to use the Source Code Lookup Tab

Introduction

The Source Code Lookup Tab is split into two sections. The first section allows you to scan for Servers where your CFML files might be located, and add/modify the lookups.

Section 1 - "Add/Modify"

At the very top of the Source Code Lookup Tab is the "Add/Modify" section. From here you have to type in the path to your CFML Server Folder. If the CFML Server folder is located locally you can tick the "Folder is located Locally" check-box and press the "Browse..." button to browse all the files on your machine. You can then locate the CFML Server Folder, click add, and it will populate the drop-down box with the path to the CFML Server Folder for you.

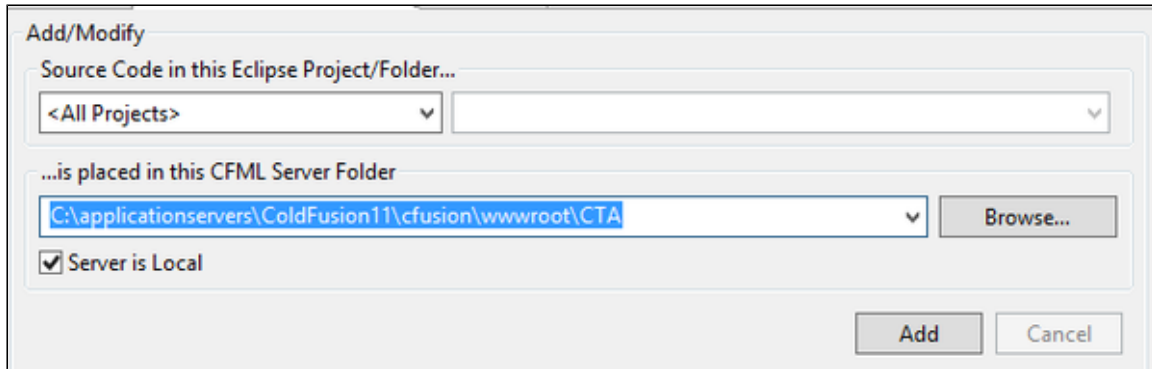


Figure 15: The Source Code Lookup Add/Modify Panel

Section 2 - "Current Source Code Lookup"

At the bottom of the Source Code Lookup Tab you will see the "Current Source Code Lookup" section. Here is where the list of lookups currently configured are contained. If you highlight a lookup and click "Edit", you can edit the Path and re-add it using Section 1. If you highlight a lookup and click "Remove" the lookup will be deleted and a new one will have to be re-added.

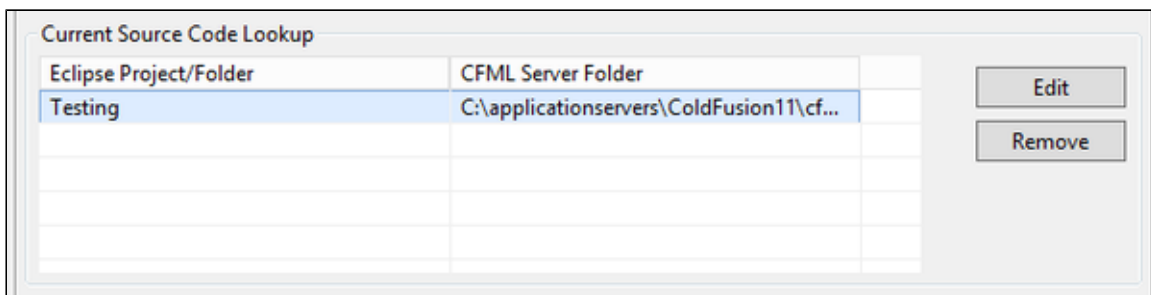


Figure 16: The Source Code Lookup Rules Panel

Warnings

As a lookup is needed in order to make a connection, you will see an error until a lookup is added: "[Source Code Lookup]: There must be a least one source code lookup defined!". Simply navigate to the Source Code Lookup tab and add a lookup to get rid of the error:

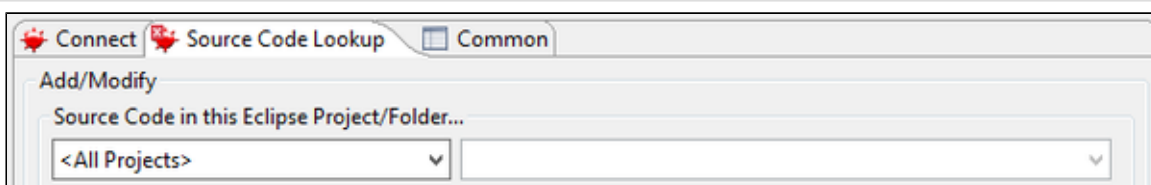


Figure 17: The Source Code Lookup Rules Panel

Alternative Webserver Setup

For simplicity, all of the examples so far have used the CF internal webserver. This doesn't mean that you can't use other web servers. Below is an example of a lookup for IIS:

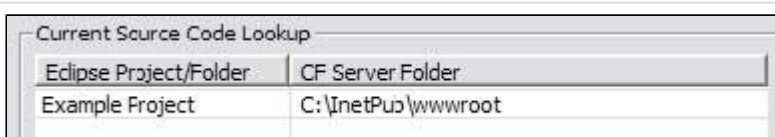


Figure 39: IIS Lookup

Below is a lookup for Apache:



Eclipse Project/Folder	CF Server Folder
Example Project	C:\Program Files\Apache Group\Apache\htdocs

Figure 40: Apache Lookup

Both of the above examples use the default directory for each webserver but you can also configure these webserver to use any folder (Virtual Directories). In this case you can just create a lookup to that folder:

Eclipse Project/Folder	CF Server Folder
Example Project	C:\myApplication

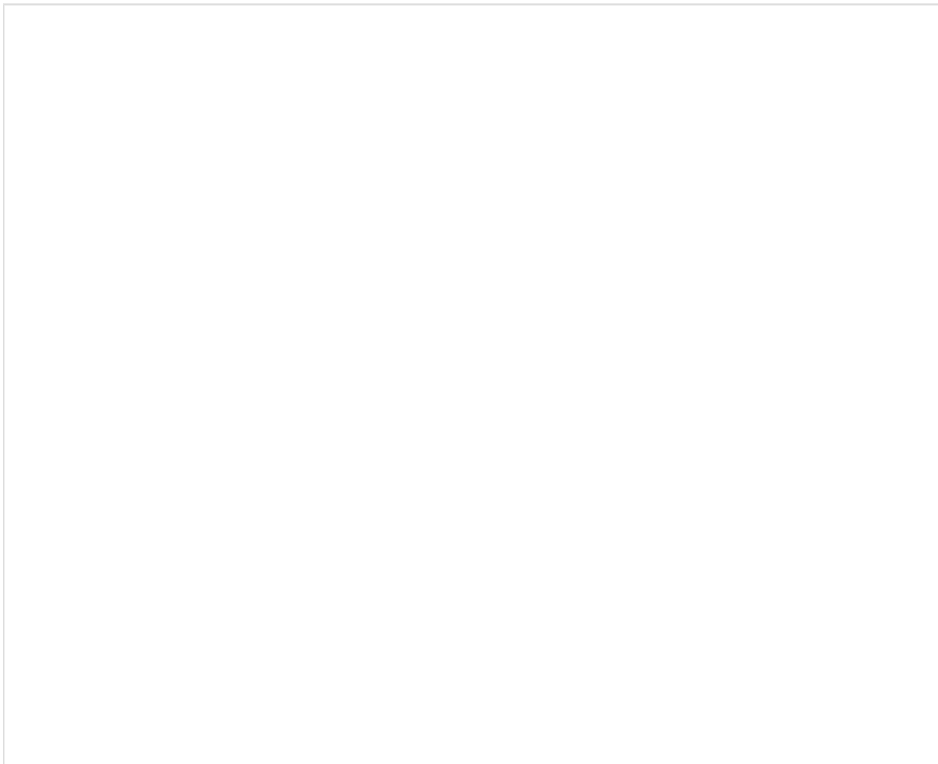
Figure 41: Virtual Directory Lookup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

CF Mappings Setup

From the CF Administrator you can set up Mappings. These allow you to store cfm code in any folder on your server. They can then be referenced within your CF pages by using their logical path. To debug these files you need to add a lookup to the directory path specified in the CF mapping.



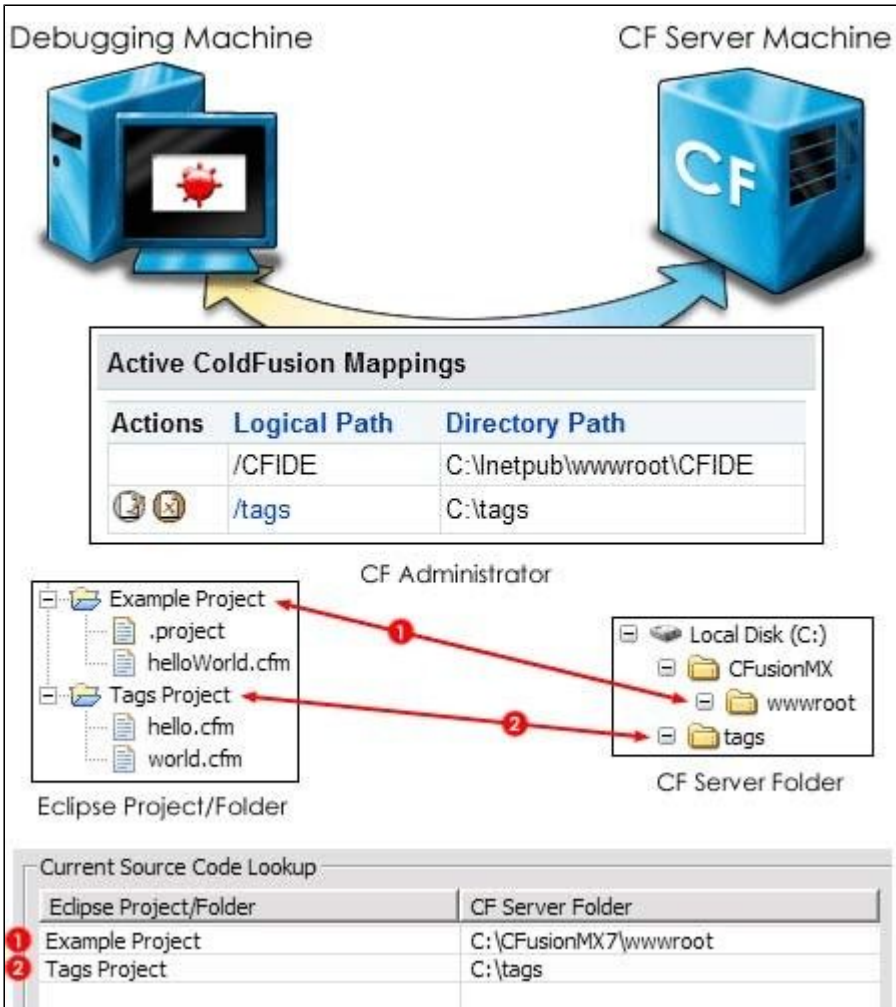


Figure 38: CF Mappings Setup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

Linux and UNIX Setup

For simplicity, all of the examples so far have used a CFML server on a Windows machine but FusionDebug can also debug CFML servers running on Solaris and UNIX systems. Below you will see a lookup to a UNIX machine running Apache.

Eclipse Project/Folder	CF Server Folder
Example Project	/opt/apache/htdocs

Figure 42: UNIX Apache Lookup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

Multiple Project Setup

Let's say that you have an application which uses a tag library. In Eclipse that tag library is a separate project which has two folders within it. On the webserver, the contents of both of these folders end up in a single folder called "tags". You would need three lookups in this case. One from your application project to the CF webserver root and then another two from each folder in the library project to the "tags" folder.

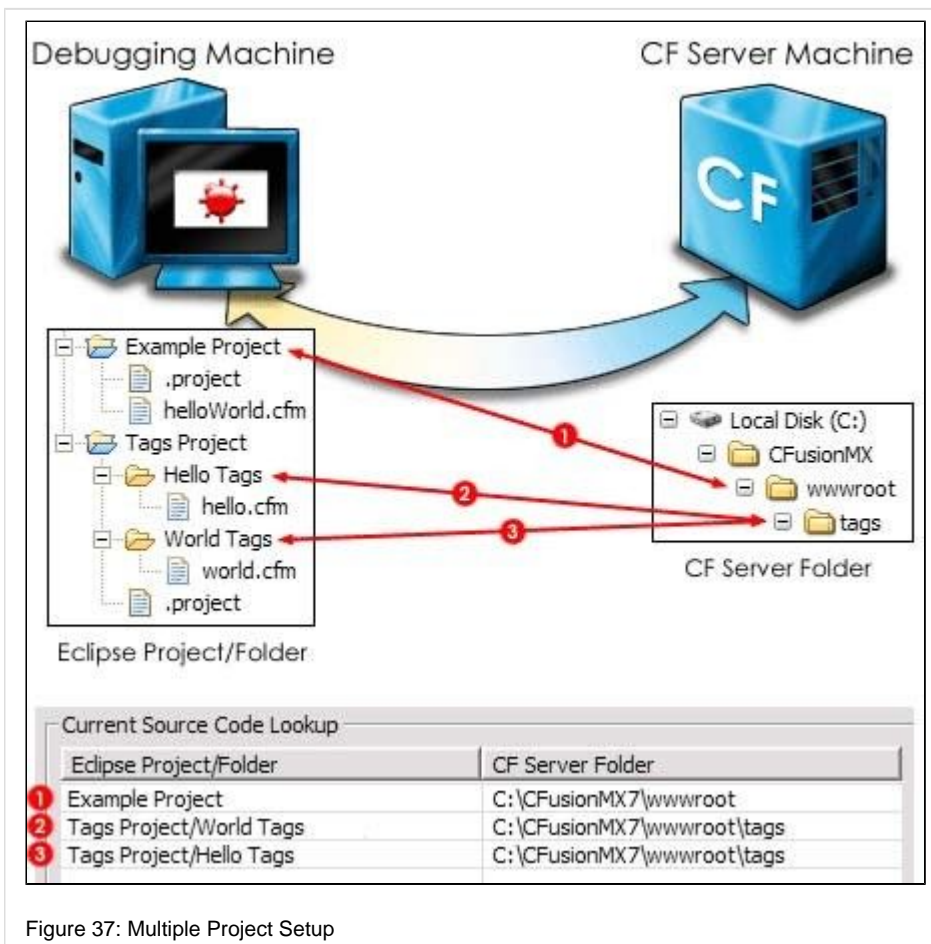


Figure 37: Multiple Project Setup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup

- Single Machine Setup

Remote Debugging Setup

If you copy your project source files directly to a remote server without changing the structure then the configuration is equally simple. You create a single lookup from your Eclipse project to the CF webserver root on that machine.

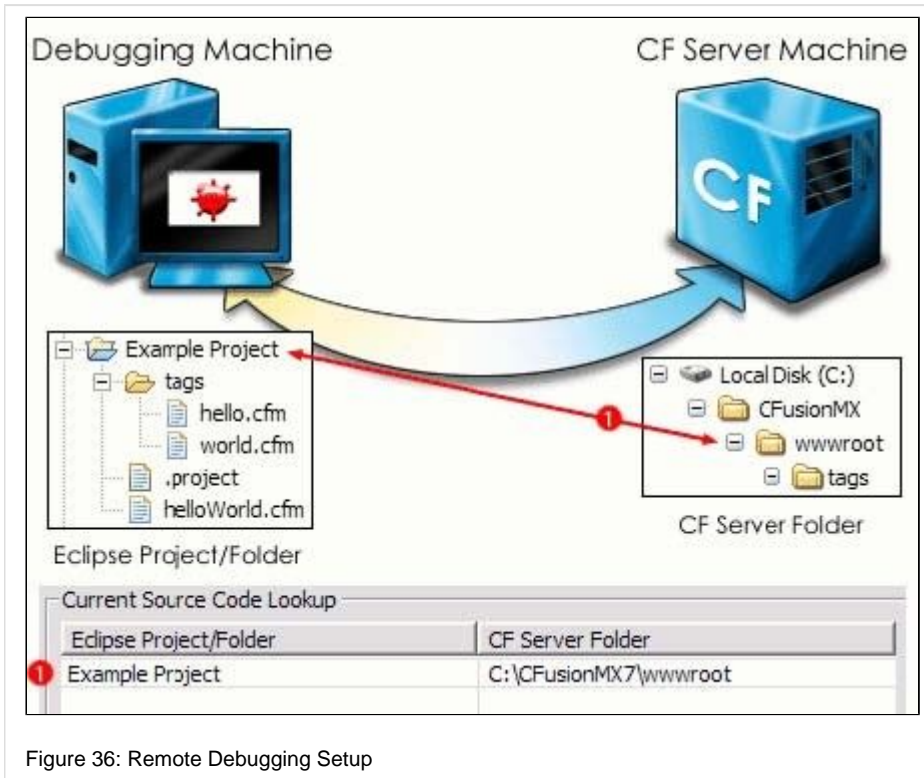


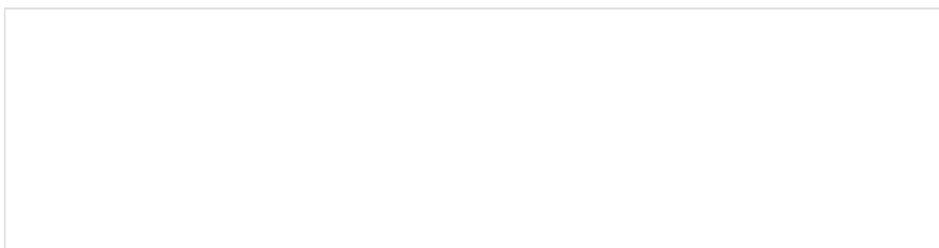
Figure 36: Remote Debugging Setup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

Single Machine Setup

The simplest configuration of FusionDebug would be if you had set up your Eclipse workspace to be the same as your CF webserver root. You would be editing the same files that are being served and you would only need a single lookup.



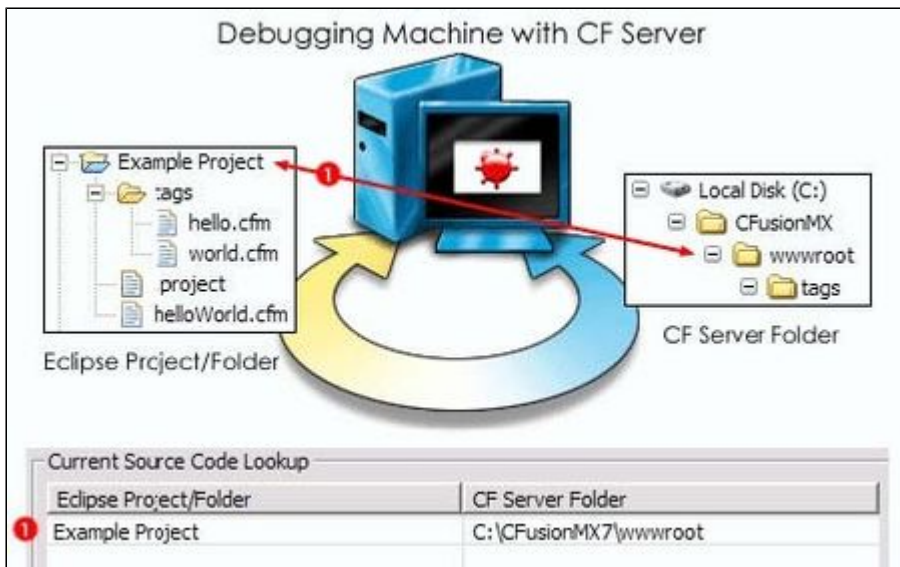


Figure 35: Single Machine Setup

The Source Code Lookup Tab:

- What is a Lookup?
- How to find the Source Code Lookup Tab
- How to use the Source Code Lookup Tab
- Alternative Webserver Setup
- CF Mappings Setup
- Linux and UNIX Setup
- Multiple Project Setup
- Remote Debugging Setup
- Single Machine Setup

FusionDebug - Configuration Examples

Child Pages

- Creating a FusionDebug Configuration
- Working Example - Remote Debugging Setup

Creating a FusionDebug Configuration


Introduction

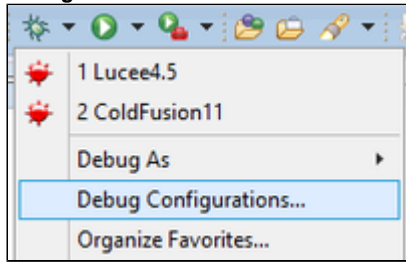
This section is going to guide you step-by-step on how to set up a FusionDebug configuration in order to connect to your local machine and begin debugging.

Getting Started

It's very important to carefully read and understand these instructions, since setting up a working FusionDebug Configuration is a key step.

In order to create a new FusionDebug configuration, please follow the steps described below.

1. Navigate to **Run > Debug Configurations** or you can simply access the **green** Debug icon  and navigate to the **Debug Configurations**. See screenshot below.



2. **Right-click** the FusionDebug element on the left of the dialog and select **New**. The FusionDebug configuration page appears in the dialog.
3. Enter a meaningful name in the **Name** field. The name should be something which describes this configuration. For this example, we are going to use the name "**CF 11**".
4. Select the Connector; this should be based on the type of server you are debugging. Currently supported servers are listed below:
 - ColdFusion 6.1, 7, 8, 9, 10, 11 and 2016
 - Railo 3 and 4
 - Lucee 4.5
5. Since we're going to connect to our local machine, leave the Host field as it is "**localhost**" and the default debugging port of the ColdFusion 11 server is port 8000.

Please make sure that the default debugging port of your application server hasn't been changed to another port. If the default port has been changed, then please the value of the Port field to the port that you have configure.

6. Optionally select which runtime exceptions to break on.
7. Before we can complete the configuration, we must add at least one Source Code Lookup to tell FusionDebug where the ColdFusion files are located on the server. Click on the "**Source Code Lookup**" tab to bring up the Lookup Table.

This table tells FusionDebug where the files are located on the server. You must enter at least one rule in this table.

When you launch the configuration later, FusionDebug will compute the actual locations of the breakpoints, and display them along with the breakpoints themselves in the Breakpoints View. Since we only have one project, we'll use the global <All Projects> rule and the **C:\FusionDebugTesting\ColdFusion\ColdFusion11\cfusion\wwwroot\FusionDebugTests** folder. Enter this data in the upper area and click Add. Figure 16 shows the Source Code Lookup tab containing the All Projects rule.

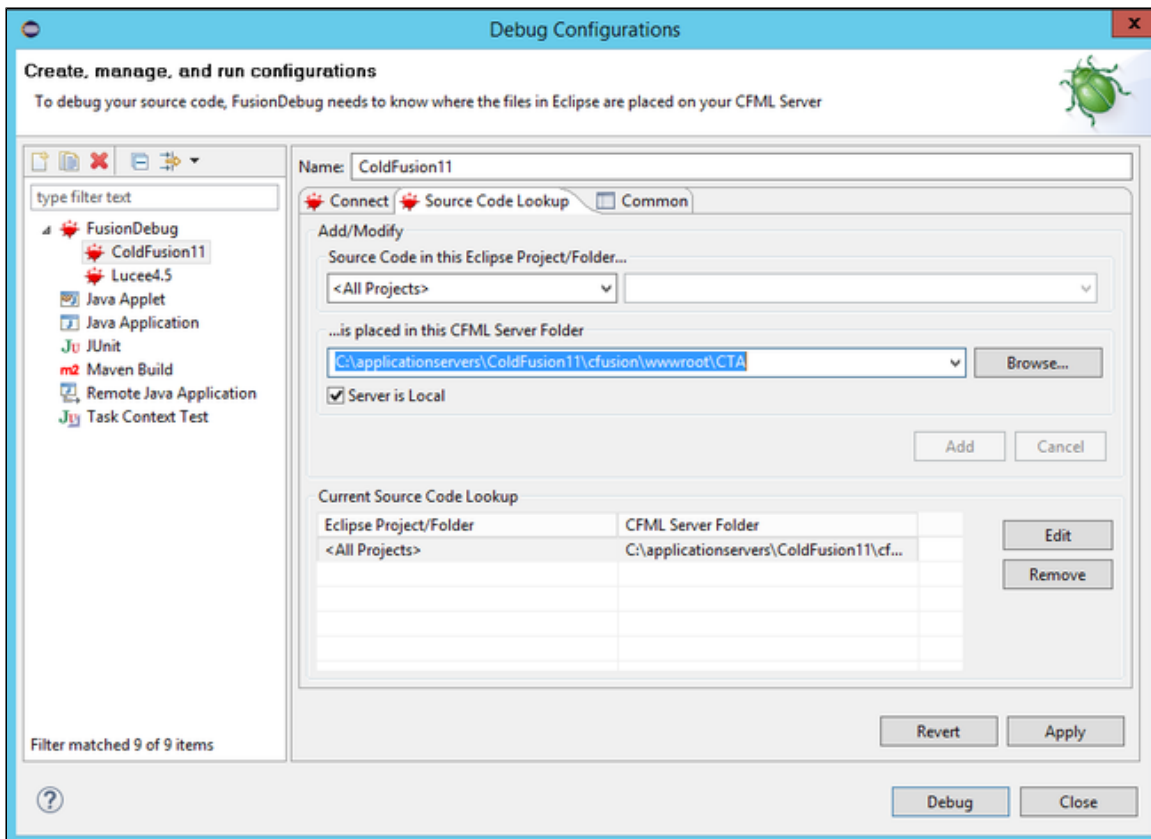


Figure 15: Setting the 'All Projects' Mapping

8. With the configuration looking similar to Figure 15, select Apply.

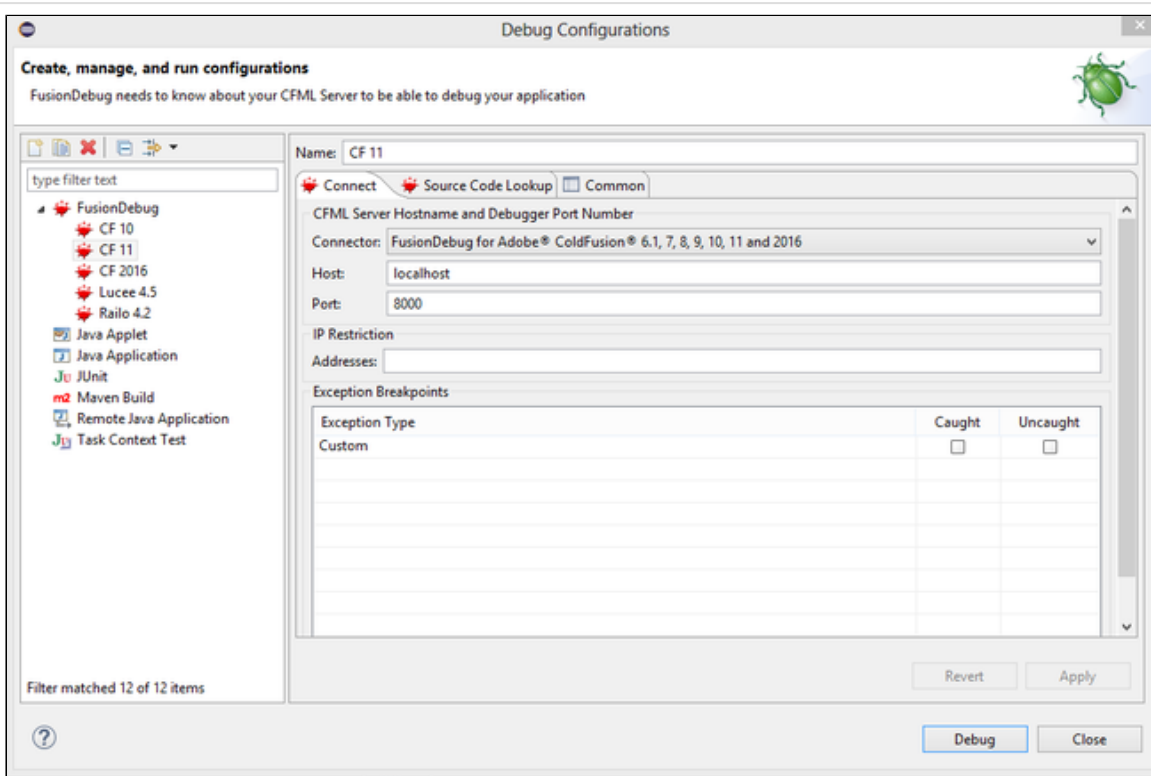


Figure 16: FusionDebug Configuration for Local Machine

9. Finally, select Debug. The dialog should disappear, and FusionDebug should connect to the local Java instance. If this succeeds, the Debug View will display the debug configuration and debug target, as shown in Figure 18.

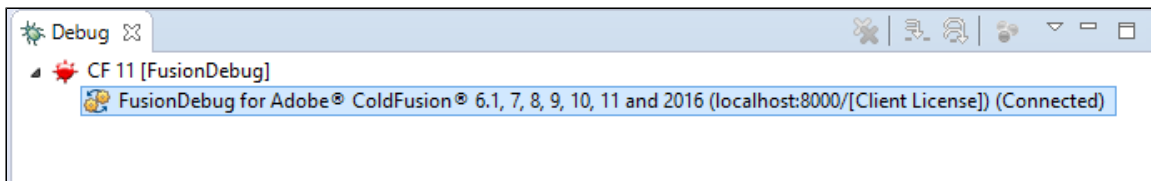


Figure 17: Successful Connection

Please note that if you want to create a new FusionDebug configuration for your Lucee or Railo application server, then in the "Connect" tab the **Connector** field should be "FusionDebug for Railo & Lucee". See Figure 18 below.

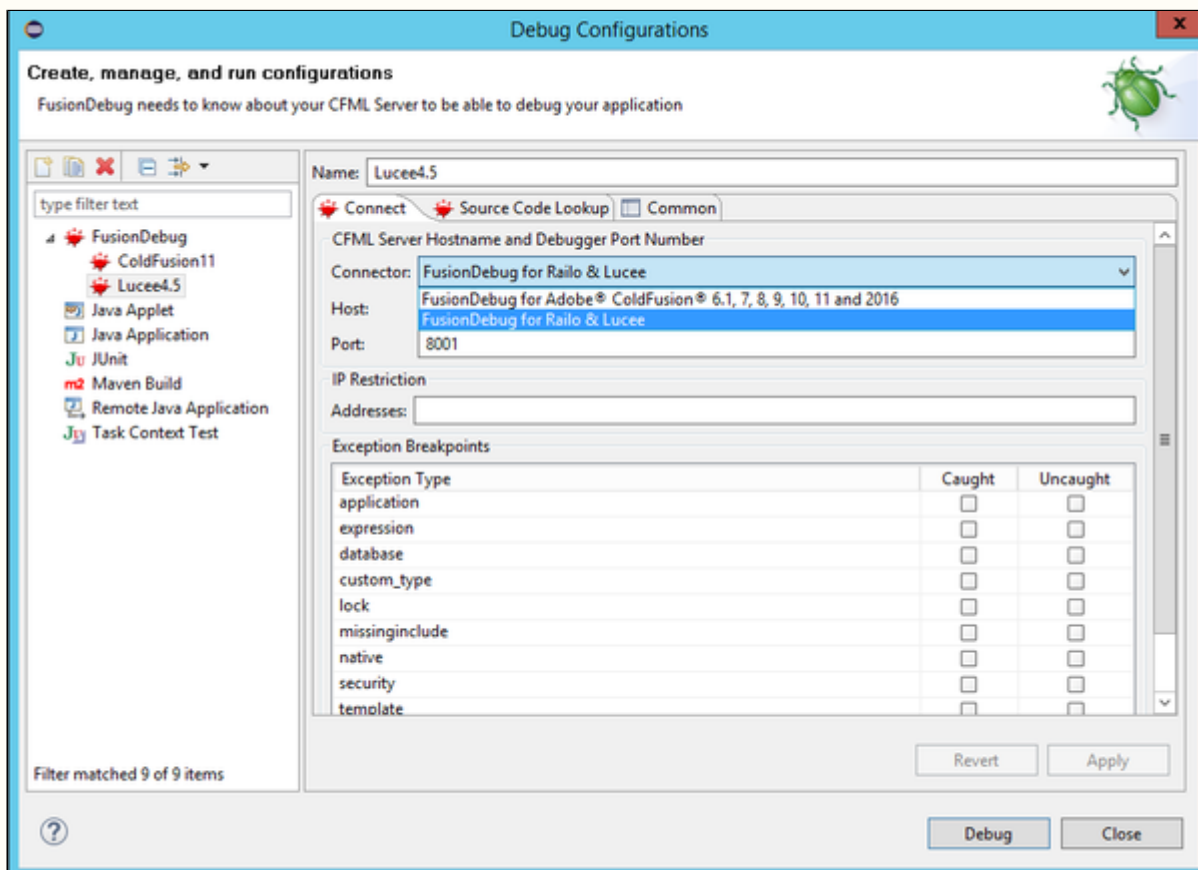


Figure 18: FusionDebug configuration for Lucee or Railo

Working Example - Remote Debugging Setup

Introduction

This section is going to guide you through the process of configuring a remote debugging setup step-by-step. For this example, we are going to use a remote ColdFusion 11 server which is hosted on a Windows Server 2008 machine while the testing environment is also hosted on a Windows Server 2008 machine.

In our testing environment we have already installed the ColdFusion Builder 3 and the FusionDebug software. If you want to download the latest FusionDebug version, check out the link, [FusionDebug Download](#).

If you don't know how to install FusionDebug in the ColdFusion Builder 3 environment, please check the following link, [Install FusionDebug](#).

Please note that this documentation can be used with any Eclipse version (excluded Eclipse Neon) and not only with the ColdFusion Builder 3.

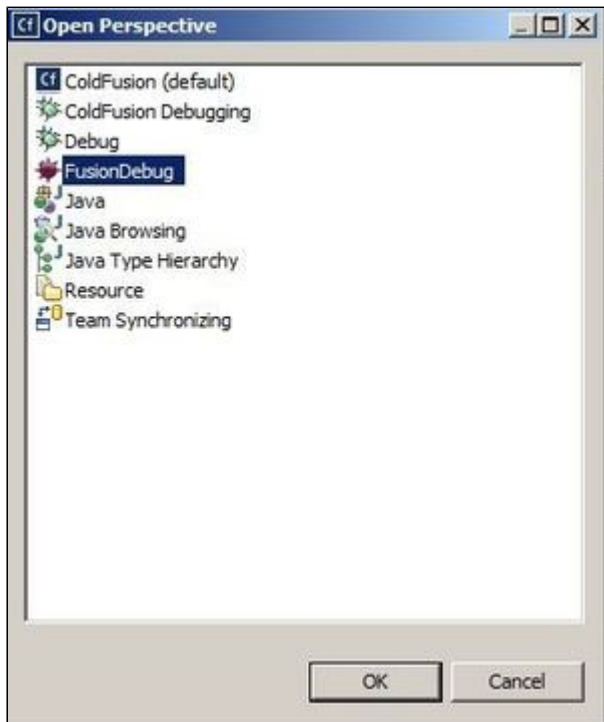
Getting Started

Step 1: Enable the FusionDebug Perspective

If you have successfully install FusionDebug in your environment, a new Perspective is going to be created. In order to enable the



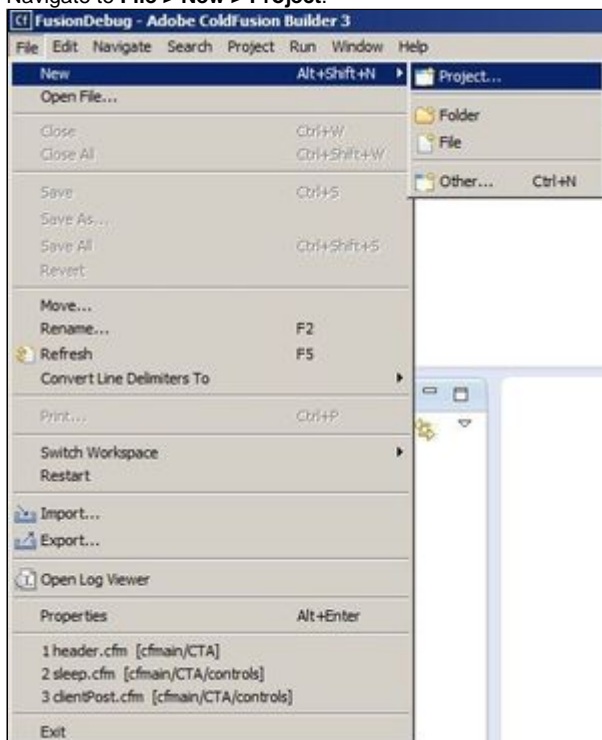
FusionDebug Perspective, press the "Open Perspective" button and then choose the FusionDebug option. See screenshot below.



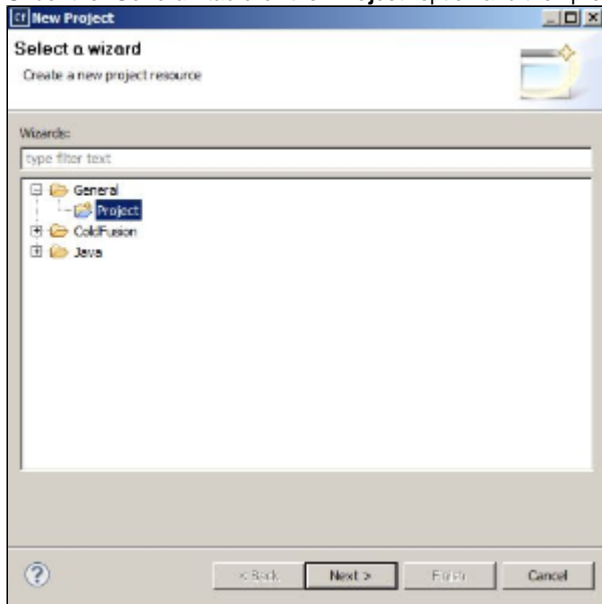
Step 2: Create a new Project in ColdFusion Builder 3

In this section we are going to access the ColdFusion Builder 3 from our testing environment and we are going to create a new project that contains the code that we want to debug. This can be done by following the steps listed below.

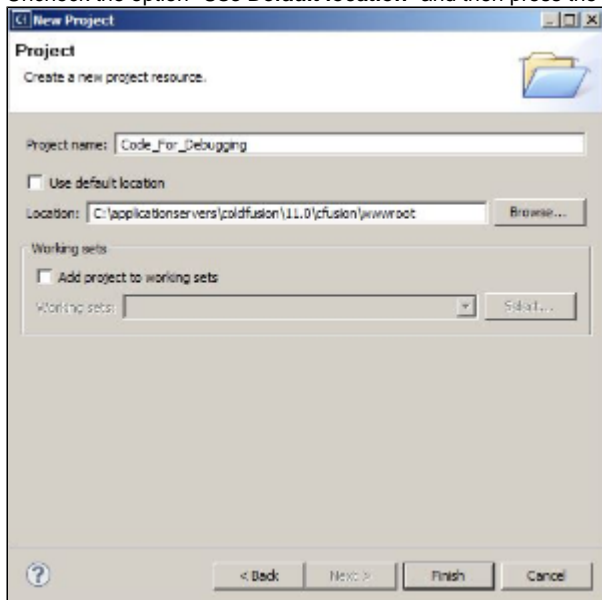
1. Launch ColdFusion Builder 3.
2. Navigate to **File > New > Project**.



- Under the **General** tab click the **Project** option and then press the **Next** button.



- Provide a name for the new project.
- Uncheck the option **"Use Default location"** and then press the button **"Browse"** and locate the code that you want to debug.

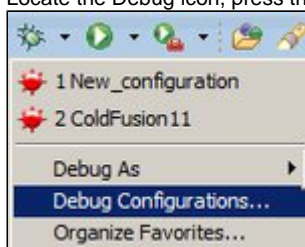


- Press the **Finish** button when the configuration is completed.

Step 3: Create a new Debug Configuration for a remote server

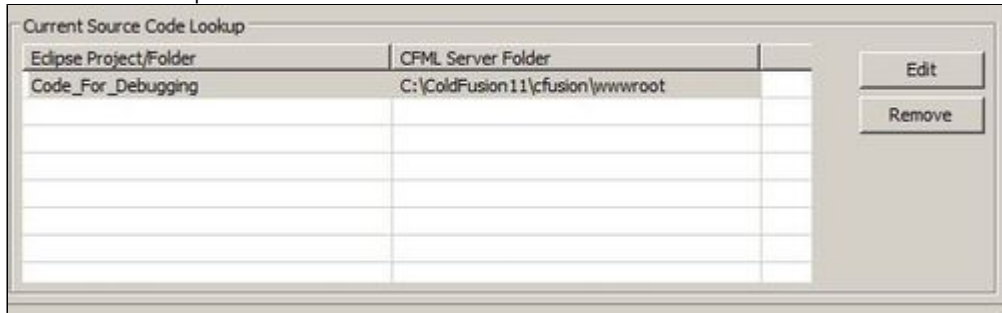
In order to create a new Debug configuration for a remote server, please follow the steps listed below.

- Locate the Debug icon, press the small arrow and then choose the option **"Debug Configurations"**.

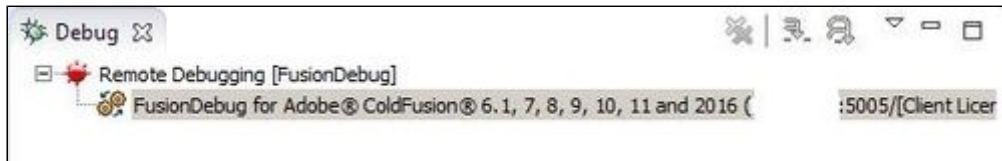


- Locate FusionDebug.

- When you are satisfied with the configuration in step 6, press the **Add** button in order to add the source code in the "Current Source Code Lookup" section.



- Press the **Debug** button in order to start debugging.
- If the connection with the remote server has been successful, you should be able to see the following.



In case you have connection issues, please check out the links listed below.

- [Troubleshooting](#)
- [Source Code Lookup Tab](#)

Step 4: Add a Breakpoint

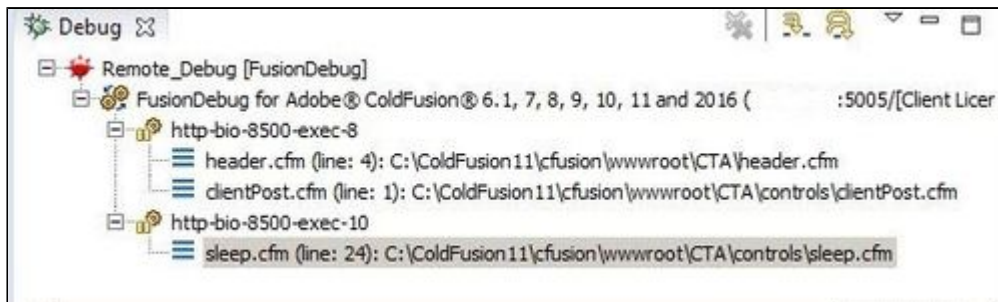
If you want to start debugging your code, then you need to add Breakpoints and inspect your code. More information about how you can add a Breakpoint can be found here, [Breakpoints](#).

More information about the Breakpoint view can be found here, [Breakpoint view](#).

Example

In my environment I have executed all the sections described above and I have set two Breakpoints. One Breakpoint has been configured in a clientPost.cfm file and another on in a sleep.cfm file. When you try to execute these files from a web browser, you can see that the Breakpoints are triggered and you are now able to debug your code. By "debug your code" we mean to use the options that the FusionDebug provides like **Step Into**, **Step Over**, **Step Return** etc.

BreakPoints



Stepping Options



More information about the stepping process can be found here, [Stepping Options](#).

Useful Troubleshooting Link

- [Non-Intrusive Debugging](#)

Useful Video Tutorials

- Video Tutorials

FusionDebug Feature Reference

Introduction

In case you have used a Debugger before or you have taken a look at the "FusionDebug Concepts - Quick Tour" page, you will be familiar with most of the features a Debugger may have.

This section will provide readers with in-depth details on how to access/use all of these Debugging features from within FusionDebug, as well as advanced concepts such as "Non-Intrusive Debugging" and modifying variables at run-time.

Child Pages

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

Debug View

The debug view reflects the state of the ColdFusion server being debugged.

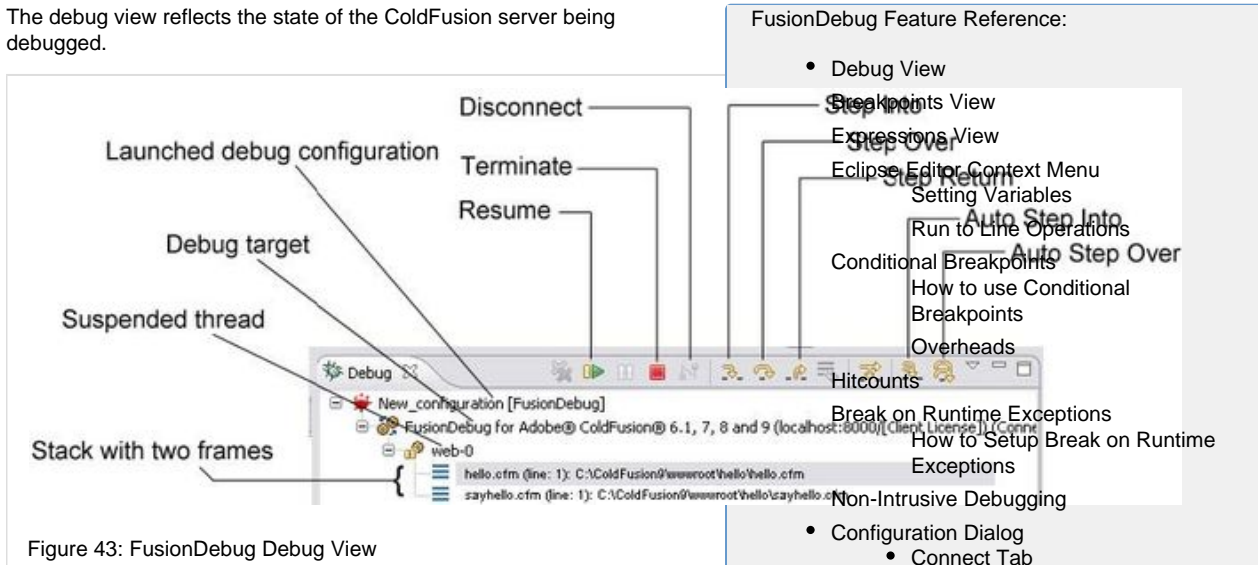


Figure 43: FusionDebug Debug View

Elements of the Debug View

La un ch ed de bu g co nfi gu	Displays the Debug Configuration used to connect to the target.
--	---

rat ion	
D e b u g t a r g e t	Displays the target (machine and port) of this debugging session.
S u s p e n d e d t h r e a d	If any threads are suspended because of a breakpoint hit, they are displayed here. If the thread is waiting for the server then you will see a status message after the thread name. (Listed below.)
St a c k w i t h t w o f r a m e s	Displays the stack of pages and CFCs involved in the current request. Each page or CFC is displayed as a frame, along with the function (if any) and line number.
Di s c o n n e c t	Active on a configuration or target. Causes FusionDebug to disconnect.
Te r m i n a t e	Active on a frame or thread, or configuration. When clicked: on a configuration, causes FusionDebug to disconnect. on a thread or frame, Causes the selected thread to be stopped.
R e s u m e	Active on a target, thread or frame. When clicked: on a target, causes all suspended threads to resume. on a thread or frame, causes that thread to resume.
St e p I n t o	If the Current Instruction Pointer (CIP) is located on any tag which would cause a sub-page (CFC constructor, CFC function invoke, page function, sub-tag) to be called, Step Into instructs FusionDebug to continue stepping inside that page. If the current tag would not cause a sub-page to run, this is equivalent to Step Over.
St e p O v e r	Causes ColdFusion to execute the tag currently under the CIP, then stop awaiting further stepping instructions.
St e p R e t u r n	If the CIP is currently located inside a sub-page (e.g. A sub-tag) causes ColdFusion to finish running the code inside the sub-page and return to the caller, where execution will stop, awaiting further stepping instructions.
A u t o - s t e p I n t o	Will automatically step into tags based on the step interval configured, which is the number of milliseconds between each Step Into operation.
A u t o - s t e p	Configured the same as Auto-step Into but automatically uses the Step Over operation.

Over	
------	--

Debug View Status Messages

(ColdFusion connector only)

The following status messages will appear after the Suspended Thread name to let you know what is currently happening:

Waiting for CF Compiler	ColdFusion is currently compiling a page required for the Step operation. The Debug View will update once the compiler has finished.
Waiting for CF Class loader	ColdFusion is searching for and loading a page, CFC or tag into memory. This often occurs after the Waiting for CF Compiler message, as part of the compile-load-run cycle.
Stepping	FusionDebug has asked ColdFusion to run the current line, and perform the requested Step operation, and is waiting for a report that the step has completed.
Waiting for Tag Validation	ColdFusion is currently validating a subtag, CFC or invoked page. The Debug View will update once the tag has been validated.

Breakpoints View

This view displays all breakpoints set in the current session.

Breakpoints can be set in any supported file (ColdFusion / CFC source files) by right clicking on the required line, and selecting Toggle Line Breakpoint, using the key combination ctrl-shift-B, or with a double-click in the ruler to the left of the Code View.

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu

The screenshot shows the 'Breakpoints' view in FusionDebug. It contains a table of breakpoints with columns for file name, line number, and server file path. Each row has a checkmark in the first column, indicating the breakpoint is enabled. Annotations point to various parts of the interface: 'Remove all breakpoints' and 'Remove breakpoint' point to icons at the top; 'Source lookup' points to a magnifying glass icon; 'Hit counts' points to a column of numbers; and 'Break on Runtime Exceptions' points to a checkbox. A 'FusionDebug Feature Reference' box is overlaid on the right side of the screenshot, listing various features and their locations.

Figure 44: FusionDebug Breakpoints View

Elements of the Breakpoint View

Breakpoints	Lists all current breakpoints. Breakpoints can be individually enabled or disabled using the check mark,
-------------	--

kp oi nts	in which case they are displayed as an empty circle: Each breakpoints specifies the file and line. Double-click the breakpoint to locate it in an editor.
R e m o v e b r e a k p o i n t	Removes the currently-selected breakpoint. Also available using the 'delete' keyboard shortcut.
R e m o v e a l l b r e a k p o i n t s	Removes all breakpoints.
G o t o F i l e	Locates the file containing the selected breakpoint in an editor. Equivalent to double-clicking the breakpoint.
S k i p a l l b r e a k p o i n t s	When selected, causes all breakpoints to be temporarily disabled. Can be used with the Resume command to run a page to completion without any intervening breakpoints firing. When activated, all breakpoints are displayed with a diagonal line:
S o u r c e l o o k u p	Real location of the file in which this breakpoint is set. Computed using the Source Lookup Tab of the FusionDebug debug configuration.

Expressions View

This view displays any watched variables or expressions. Expressions can be entered by selecting them in an editor, right-clicking and selecting Watch Expression, or by right-clicking in the Expressions View and selecting Add Watch Expression.

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging



Figure 45: FusionDebug Expressions View

Elements of the Expressions View

Expressions	Lists the expressions which will be evaluated.
Remove all expressions	Removes all expressions from the view.
Remove expression	Removes the selected expression from the view.

Valid Expressions

Almost all ColdFusion expressions are valid. If an expression is valid in CF Script, it is a valid Watch Expression. All variables, scope and structure references are valid:

- url.username – the 'username' variable from the URL scope.
- now() - the result of the now() function – the current date and time
- y – the value of the local variable 'y'.
- http – the 'http' variable scope (this will be expandable to reveal its variables)

Evaluation

All expressions are evaluated whenever a thread pauses, either because it has hit a breakpoint or as a result of a step action. During evaluation, the expression will display "(pending)".

Expressions can be manually re-evaluated by right-clicking the expression and selecting Reevaluate Watch Expression.

Expressions can be disabled from evaluation by right-clicking and selecting Disable. The expression will be tagged "(disabled)".

Eclipse Editor Context Menu

When right-clicking within a supported source code file in an Eclipse editor, five FusionDebug elements are available, identifiable by their red FusionDebug tags (With the exception of **Run to Line**).

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View

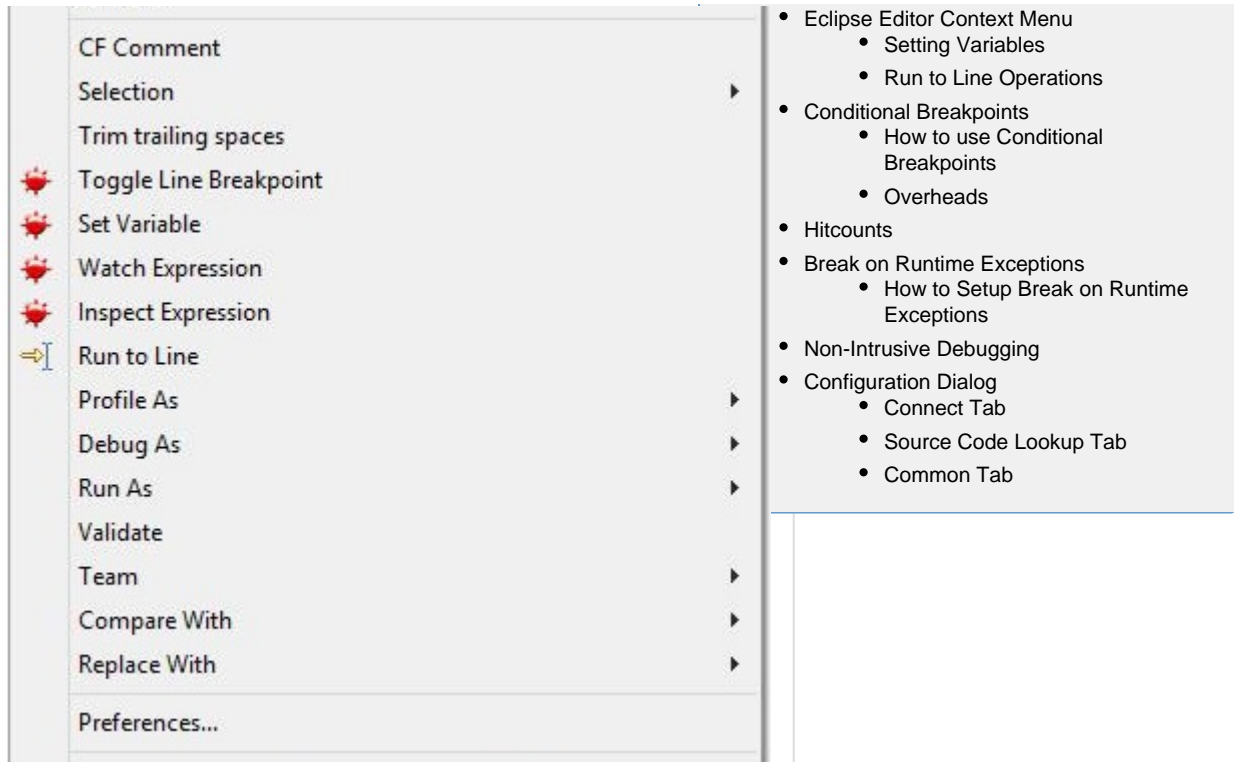


Figure 46: Eclipse Editor Context Menu

Elements of the Context Menu

Toggle Line Breakpoint	Toggles a breakpoint at this line.
Set Variable	Allows the user to set the value of a variable.
Watch Expression	Copies the selected variable or expression to the Expressions View and evaluates it.
Inspect Expression	Immediately evaluates the value of any highlighted expression and displays it in a window.
Run to Line	Perform a Run to Line operation on the currently selected line.

Setting Variables

The Set Variable option allows the user to change the value of a variable.

- The currently highlighted variable will be entered automatically or a different variable name can be entered.
- The value of the variable will be set in the currently-selected thread.
- The page running in that thread will immediately see the new value of that variable.
- String values must be encapsulated in double quotes ("Hello, world!")
- Expressions may also be entered – they will be evaluated and their result assigned to the variable.

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts

This option is equivalent to running a `<CFSET var=value>` at the location of the Current Instruction Pointer.

- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

Run to Line Operations

The Run to Line function in the context menu allows you to select a line in the editor, and perform a Run to Line operation to this location. This function works when execution is already stopped at some point i.e. A stack is already selected. This function allows you to traverse large blocks of code without the need to set new breakpoints or step through all of the code.

This feature resumes the execution of code until this point or another breakpoint is reached. You can allow breakpoints to be skipped (or not) with the honored preference within the Window, Preferences, Run/Debug tab.

- FusionDebug Feature Reference:
- Debug View
 - Breakpoints View
 - Expressions View
 - Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
 - Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
 - Hitcounts
 - Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
 - Non-Intrusive Debugging
 - Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

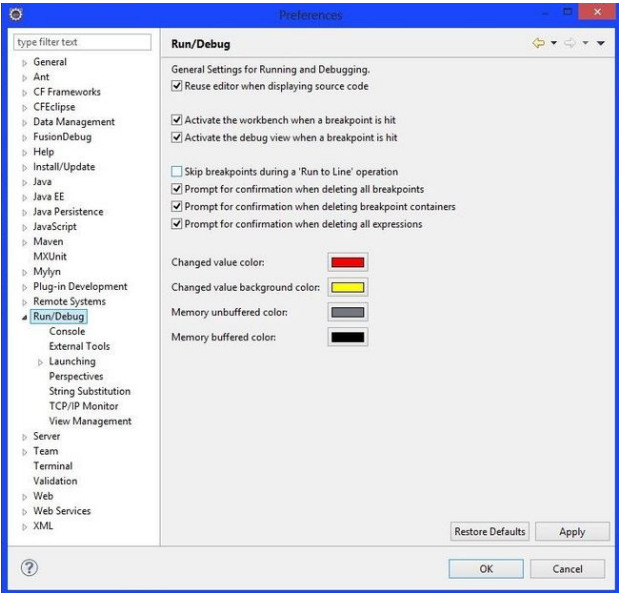


Figure 47: Run/Debug Preferences

- The Run to Line 'hot key' persists, as Eclipse users are used too; Ctrl+R.NOTES:
- Run to Line will take the currently selected cursor location, not mouse position.
- A Run to Line operation may not fire if the code location is not hit, depending on your breakpoints and Run /Debug setting for skipping breakpoints, this may cause a page to run to completion.

Conditional Breakpoints

Child Pages

- How to use Conditional Breakpoints

- Overheads

How to use Conditional Breakpoints

Conditions can be added to any FusionDebug breakpoints to give the user more control over when the breakpoints will fire. The condition must be in the form of a CFML expression that evaluates to either TRUE or FALSE. Conditions can be added or changed during runtime of an application. To add or change a breakpoint's condition right-click on a FusionDebug breakpoint in the Breakpoints view and select Breakpoint Properties.

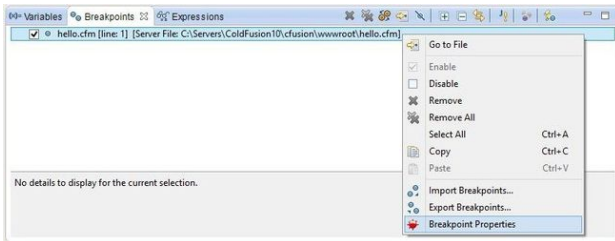


Figure 48: Breakpoint Context Menu

On the Breakpoint Properties window, the breakpoint can be enabled or disabled and its condition can be enabled or disabled. The large text area is where the condition must be entered if the Enable Condition checkbox is ticked.

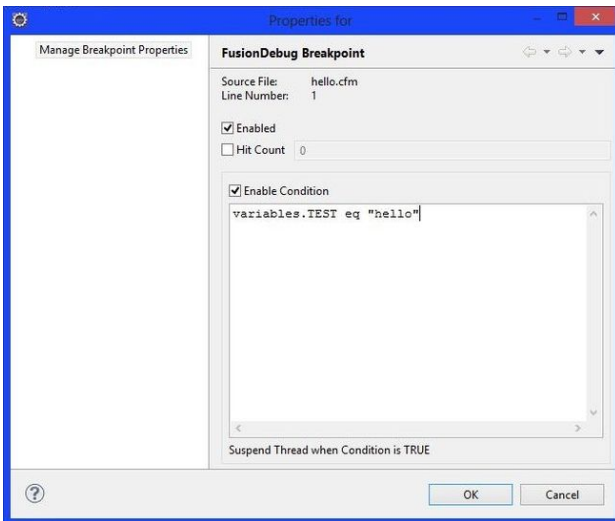


Figure 49: Breakpoint Properties

Now when the page is executed, if the breakpoint is hit and the condition is true, the breakpoint will fire, otherwise it will be ignored.

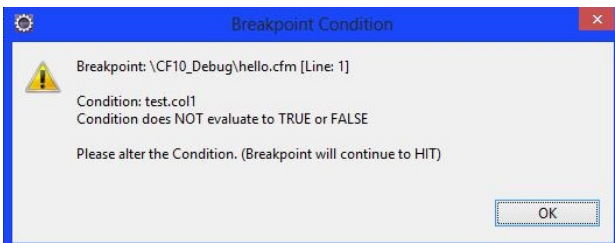


Figure 50: Invalid Breakpoint Condition Warning

If a condition can not be evaluated to TRUE or FALSE, the breakpoint will fire as if it were a standard breakpoint and a

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

warning will be displayed. If the condition is not changed, and the breakpoint is reached again, it will continue to fire as usual and re-display the message.

Overheads

There are certain overheads to be aware of when using conditional breakpoints. An evaluation has to take place to check the condition of the breakpoint. For example, in a large loop with a conditional breakpoint inside, the evaluation must be made on each iteration of the loop.

Example:

On a core 2 CPU machine with 3 GB of RAM iterating over a 2000 row query:

```
3 <cfquery name="qry" datasource="test1">
4   select * from TestData
5 </cfquery>
6
7 <cfloop query="qry">
8   <cfoutput>#qry.col1#<br></cfoutput>
9 </cfloop>
```

Condition: qry.id LT 0

This will never return true, and the breakpoint condition will be evaluated 2000 times.

These evaluations take on average 45 seconds for 2000 – this gives an average evaluation time of 22.5 milliseconds. Note: evaluation time will differ depending on machine, server type and other factors

Hitcounts

Breakpoints can also have a hitcount. If a breakpoint has a hitcount it will only fire after the count has been reached, otherwise the breakpoint will be skipped and the count will be incremented. Hitcounts are useful in loops to halt on a specific iteration or in a query to halt on a specific row.

A breakpoint hitcount is set in the Breakpoint Properties page, shown here:

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab

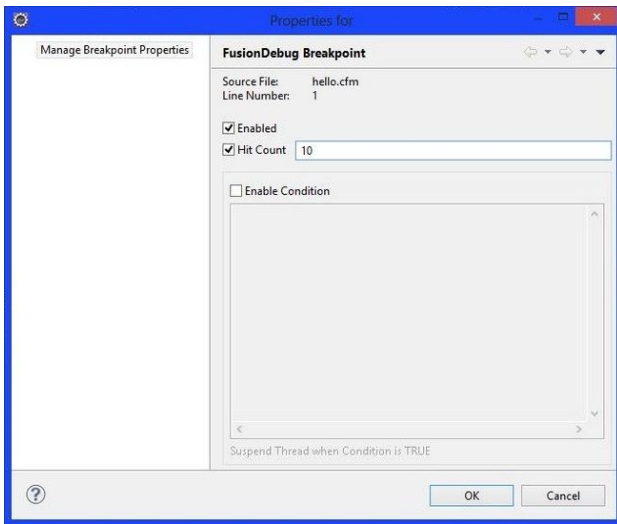


Figure 51: Setting a hitcount

Hitcounts can be used in conjunction with breakpoint conditions. The count will only increment when the breakpoint condition is true.

Break on Runtime Exceptions

Debug configurations include the option to break on runtime exceptions. This means that when an exception of a certain type, caught and/or uncaught, is thrown, the page will halt at that line of code as if there had been a breakpoint set. Only runtime exceptions can be detected by FusionDebug, any other exceptions or errors can not be handled.

The option to break on caught and uncaught exceptions is given in the debug configuration. A caught exception is one that is thrown whilst wrapped in a `<cftry>` tag body, and properly caught with a `<cfcatch>` tag. However, this only applies to Railo servers. ColdFusion defines a caught exception as one thrown in a `<cftry>` body, but does not need to have a corresponding `<cfcatch>` tag. An uncaught exception is one thrown outside of a `<cftry>` tag body.

There are differences between the exception types available in the ColdFusion connector and the Railo connector. The ColdFusion connector can only break on custom exceptions. The Railo connector can break on application, expression, database, custom_type, lock, missinginclude, native, security and template exceptions.

How to Setup Break on Runtime Exceptions

An important point to note is that when setting up Break on Runtime Exceptions, the debug configuration must be re-launched so that the connector can re-connect. If the debug configuration is changed while connected you will see no affect of this change until re-connecting. To enable the detection of runtime exceptions you have to select

- Source Code Lookup Tab
- Common Tab

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View

which type you wish to break on and whether caught and/or uncaught variations are included.

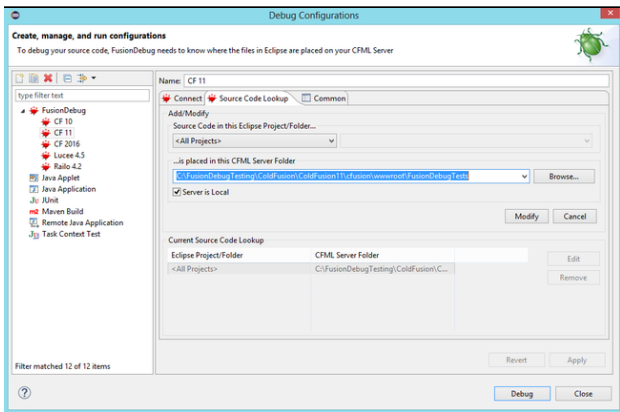


Figure 52: Exception Breakpoints Setup

Here, the lock exception has been selected for both caught and uncaught variations.

Now when a lock type exception is thrown, either caught or uncaught, the application will halt. Both caught and uncaught situations are shown below, demonstrating the cause of exception and the position of the break.

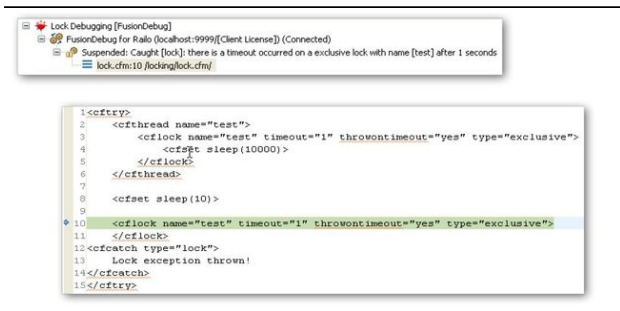


Figure 53: Caught Exception Breakpoint Fired

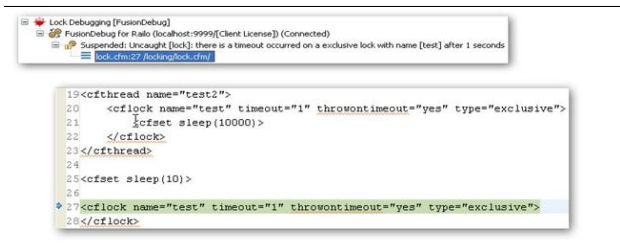


Figure 54: Uncaught Exception Breakpoint Fired

Non-Intrusive Debugging

Introduction

A new feature of FusionDebug 3.5 is the ability to only fire breakpoints if any of the specified IP addresses are connected.

- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View

This could previously be done by enabling a breakpoint condition on all of the breakpoints.

This feature is still available in 3.6 and this section will cover how to configure and use the IP Restriction feature in FusionDebug 3.6.

How does Non-Intrusive Debugging work?

The **IP Restriction** feature in FusionDebug is comparing the CGI.REMOTE_ADDR with the IP addresses entered in the Debug Configuration screen. Only if the remote address variable matches any of the specified addresses will the breakpoints fire.

If breakpoints do not fire when they should and IP Restriction is enabled, make sure that the IP addresses entered in the Debug Configuration screen are equal to the addresses that connect to the ColdFusion server.

How to use Non-Intrusive Debugging

Non-Intrusive Debugging can be enabled to any new or existent Debug Configurations in FusionDebug by entering the IP addresses to fire breakpoints for in the IP Restriction section available in the Connect Tab of the Debug Configuration, as shown in Figure 55.

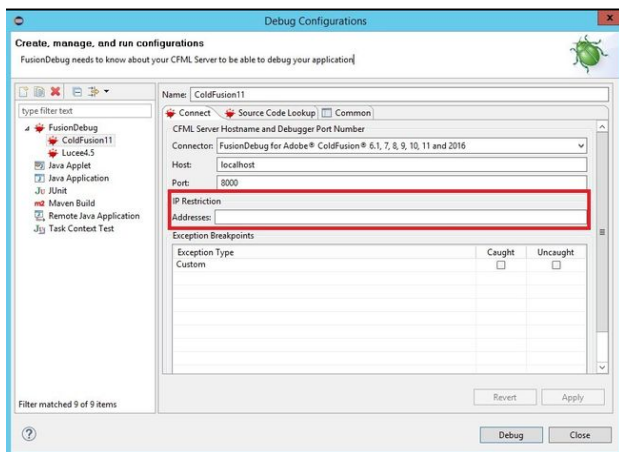


Figure 55: The Debug Configuration with Non-Intrusive Debugging

Non-Intrusive Debugging is enabled automatically by entering at least one IP address in the Addresses field. To enter multiple IP addresses use a comma (,) to separate them.

For example, if a debug configuration should only halt when connecting locally, entering 127.0.0.1 (or 0:0:0:0:0:0:1 IPv6 address) then this will enable other machines to request the file without the breakpoints firing.

Please note that the debug session will have to be relaunched for the IP Restriction configuration to take effect.

Configuration Dialog

Introduction

- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

FusionDebug Feature Reference:

- Debug View

This section details the options available in the FusionDebug Launch Configuration dialog.

Getting Started

This dialog allows the user to specify one or more launch configurations: collections of parameters used to connect FusionDebug to a target CFML server.

It is accessible from the **Run > Debug Configurations** or by selecting **Debug Configurations** from the debug drop-down menu in the Debug Toolbar, as shown in Figure 56.

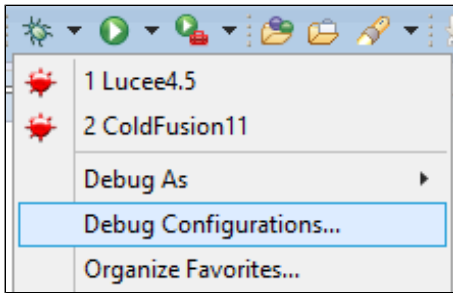


Figure 56: The Debug Drop-Down Menu

The configuration dialog for FusionDebug connections is comprised of three tabs: Connect, Source and Common. The Name: field is common to all three tabs, and specifies the name of this configuration. The name is used in the Debug View to identify which configuration is in use.

Connect Tab

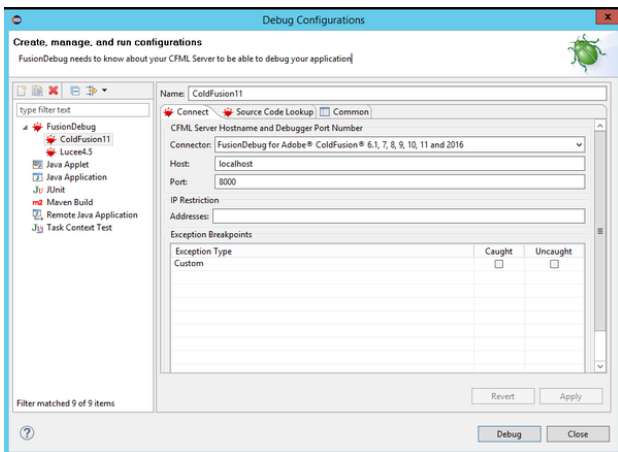


Figure 58: The Connect Tab

ELEMENTS OF THE CONNECT TAB

Connector

Specifies the CFML server that is to be debugged, either a ColdFusion or Railo server.

Host

Specifies the name of the host to which FusionDebug should connect, when this configuration is launched.

- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

Port

Specifies the TCP port on which the Java debug agent is listening. This is configured (for standard ColdFusion installations) in the `jvm.config` file

IP Restriction

The IP addresses to fire breakpoints for are specified, separated with a comma. This means that the breakpoints will only fire when any of the entered IP addresses are connected but not for anyone else.

Exception Breakpoints

Specifies the runtime exceptions that FusionDebug should break on. Different exception types exist for the ColdFusion and Railo connectors.

Source Code Lookup Tab

The **Source Code Lookup** tab allows one to specify where FusionDebug should attempt to locate source code when a breakpoint fires, or a step into or step return action occurs. This tab is also displayed when FusionReactor can't find a source file during Step operations.

When connected, changes to this data take effect immediately, and the Breakpoint View is immediately updated with the new mappings. When disconnected, changes to this data will be reflected when FusionDebug next connects.

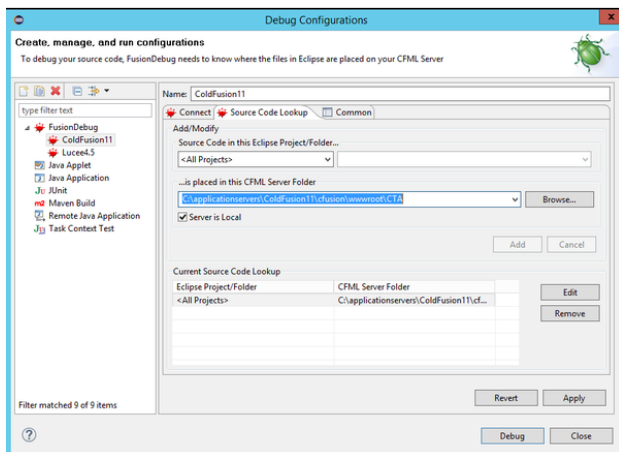


Figure 59: Source Code Lookup Tab

ELEMENTS OF THE SOURCE CODE LOOKUP TAB

Upper Panel (Add/Modify)

The upper panel allows the insertion of new lookups, and modification of existing lookups.

- The Eclipse Project drop-down contains all open projects, as well as a global All Projects mapping. The All Projects mapping is sufficient for simple projects, and is a good starting point before adding more complicated mappings.
- The Eclipse Folder drop-down contains all the paths for the project selected in the Eclipse Project. If the All Projects mapping is selected, this drop-down is disabled.
- The CFML server Folder field should contain the exact path to the mapping defined in the first two drop-downs from the point of view of the server. Any Windows drive

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

letters, Unix NFS mounts or symlinks should be as viewed by the server itself. It is possible to use the drop-down arrow in this field to select previously-entered mappings.

Lower Panel (Current Source Code Lookups)

- The lower panel displays existing mappings. To remove or edit an existing mapping, select it by clicking on it, and click on Remove or Edit respectively.

USEFUL LINKS

- Source Code Lookup tab

Common Tab

The common tab is not used by FusionDebug and can be ignored.

FusionDebug Feature Reference:

- Debug View
- Breakpoints View
- Expressions View
- Eclipse Editor Context Menu
 - Setting Variables
 - Run to Line Operations
- Conditional Breakpoints
 - How to use Conditional Breakpoints
 - Overheads
- Hitcounts
- Break on Runtime Exceptions
 - How to Setup Break on Runtime Exceptions
- Non-Intrusive Debugging
- Configuration Dialog
 - Connect Tab
 - Source Code Lookup Tab
 - Common Tab

Troubleshooting / FAQ

Introduction

This section lists answers to some of the common problems encountered using FusionDebug. You can also find support information, updates and articles at the FusionDebug Website or go straight to the FAQ located there.

Common Problems

FusionDebug will not connect

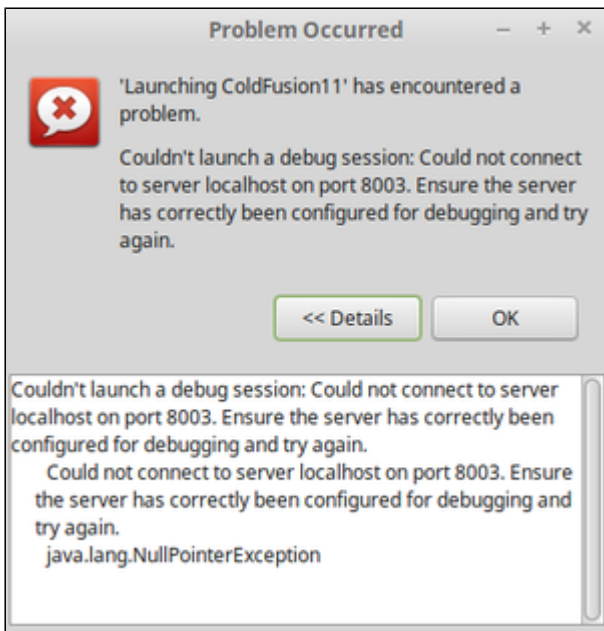
The message is: "**FusionDebug could not connect to the target system (machine:port)**".

On this page:

- Introduction
- Common Problems
 - FusionDebug will not connect
 - FusionDebug will not launch
 - Expressions do not evaluate
 - Expressions listed as undefined
 - I see many pages in Debug View
 - Breakpoints do not fire
 - Breakpoints are not active
 - Unable to display source code
- Child Pages

Where next:

- Introduction & Concepts



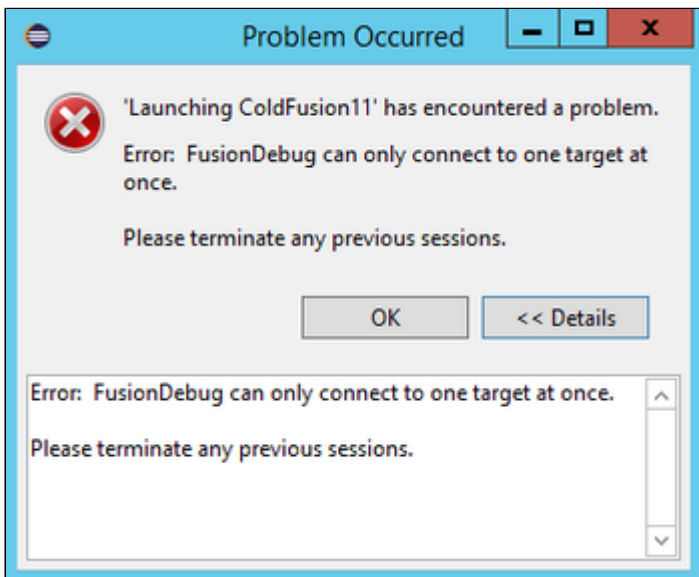
- [Setting up Java for Debugging \(ColdFusion, Railo & Lucee\)](#)
- [Getting Started - Install FusionDebug](#)
- [FusionDebug Concepts - Quick Tour](#)
- [The Source Code Lookup Tab](#)
- [FusionDebug - Configuration Examples](#)
- [FusionDebug Feature Reference](#)
- [Troubleshooting / FAQ](#)

POSSIBLE FIXES

- Check you have enabled Java debugging on a port that is available, and that the CFML server has started up correctly.
- If you are using a non-standard port (not 8000), make sure you have correctly entered that port in the configuration dialog.
- Make sure there are no firewalls or port restrictions in the network path between you and the target system.

FusionDebug will not launch

The message is: **“FusionDebug can only connect to one target at once.”**



POSSIBLE FIXES

- FusionDebug 3.6 is capable of attaching to one ColdFusion instance at a time. Terminate your previous session.
- See [Terminate](#) on how to Terminate a previous session.

Expressions do not evaluate

The Expressions View can handle most expressions which are valid in ColdFusion. A good rule of thumb is that if you can write it inside a CFScript block, or within “#” characters, it's a valid expression.

POSSIBLE FIXES

- Expressions are evaluated and re-evaluated only in the context of a suspended thread. If no thread is suspended, expressions are not evaluated. If a thread is suspended, try clicking on it. This will implicitly tell the Expressions View to re-evaluate its expressions for that thread.

Expressions listed as undefined

The Expressions View tried to evaluate the expression, but it either didn't return a value (for function calls) or the variable referenced in the function was not visible at the position the thread was suspended at.

I see many pages in Debug View

These threads have suspended because of a breakpoint. Breakpoints fire whenever a request hits them. If you see many pages in the Debug View, this is because a web browser requested that page, and the page hit a breakpoint.

POSSIBLE FIXES

- If you have a shared test server, we recommend disabling your breakpoints, or activating “Skip All Breakpoints” (See: Breakpoints View) after your breakpoint has fired, so requests from someone else do not suspend.

Breakpoints do not fire

POSSIBLE FIXES

- Check that FusionDebug is connected to the application server that you are using (the Debug View looks similar to Figure 2 on Debug Target).
- Check that the breakpoint which should fire has been looked up correctly.
 - The exact file in which this breakpoint is set is shown next to the breakpoint as the Server File. If this is not showing the correct file, or shows “Breakpoint not active. Edit Source Code Lookups”, edit your Source Code Lookups using the Debug Configuration.

Breakpoints are not active

The message is: “Breakpoint not active. Edit Source Code Lookups.” What this error means is that although you have told Eclipse that you want to set a breakpoint, FusionDebug couldn't work out in which file on your CFML server the breakpoint should actually be set.

POSSIBLE FIXES

- Go back to the Source Code Lookup tab and make sure that your project is associated with the correct folder on the server
- Changes to the Source Code Lookup tab take effect immediately after you select Apply, so this error will disappear as soon as FusionDebug can compute the real location of the breakpoint correctly.

Unable to display source code

If you step into a file whilst debugging, you may instead see a page containing the following message: "FusionDebug is unable to show you the source code for the CF Server". What this page means is that the server stepped into a file but FusionDebug couldn't work out which file in your Eclipse projects was associated with the one stepped into on the server. There are two possible reasons for this:

- The project containing the source file is closed: If you read through the page, it will tell you which lookups were attempted and will also let you know if you have any closed projects. FusionDebug cannot show source code from closed projects. Make sure all related projects are open.
- You are missing a mapping or have an incorrect lookup. At the top of the page you will see a button which will take you directly to the Source Code Lookup dialog. If you click on this then you will see that the CFML server Folder has already been filled out. All you need to do is enter the Eclipse Project/Folder information and you can add a new lookup.

Child Pages

- [FDS-128] Using FusionDebug to debug Flash Remoting and Gateway Messaging Requests
- [FDS-130] FusionDebug: Feature Focus - Non-Intrusive Debugging
- [FDS-118] OnDemand Seminar: FusionDebug Introduction (with Railo)
 - [FDS-125] Configure FusionDebug 3 for use with Railo 3.1 on Tomcat
 - [FDS-121] FusionDebug: Feature Focus - Breakpoint Conditions & Hitcounts
 - [FDS-122] FusionDebug: Feature Focus - Auto-step

[FDS-128] Using FusionDebug to debug Flash Remoting and Gateway Messaging Requests

Debugging Flash Remoting and Gateway Messaging

Non-Intrusive debugging has been introduced in FusionDebug 3.5, which allows breakpoints to only fire when requests are coming from a specified IP address. This makes it possible for a developer to debug and set breakpoints against a central server, leaving the system untouched and unaffected for everyone else.

This article will demonstrate that the breakpoint IP filter can be used to debug Flash Remoting and Gateway Messages. In order to demonstrate that FusionDebug's non-Intrusive debugging functionality is effective across a wide range of message types, below is a short example of testing the filter functionality in regards to Flash Remoting messages and Gateway messaging. I have based these tests on the test classes found at <http://www.cftips.net> and <http://www.fusioncube.net>.

Flash Remoting

I created a Flex project with an mxml file containing a script that has two functions; a function that simply populates a data grid with the results returned by a remote CFC and a second function that accepts a fault event and displays an error message in case of problems. I also need a remote object call specifying a destination named ColdFusion (which is configured with the appropriate channel for the passing of messages) and specified the location of the CFC you would like to access as well as creating a data grid to store the retrieved results.

Remoting.mxml

```
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import mx.rpc.events.FaultEvent;
        import mx.rpc.Fault;
```

```

import mx.rpc.events.ResultEvent;

private function result(evt:ResultEvent):void {
    mygrid.dataProvider = evt.result;
}

private function fault(evt:FaultEvent):void {
    Alert.show(evt.fault.message);
}

]]>
</mx:Script>

<mx:RemoteObject id="mycfc" destination="ColdFusion"
showBusyCursor="true"
source="flashRemote" result="result(event)" fault="fault(event)">
</mx:RemoteObject>
    <mx:Button label="Button" click="mycfc.GetData()" />
    <mx:DataGrid id="mygrid" width="487" height="169" />
</mx:Application>

```

Below are some changes that I found needed to be made to CF8 and CF9 Application servers' configuration before the Flash Remoting test class would function correctly;

Within flex-messaging-gateway.cfg. If your Flex server is installed as part of your CF app server, comment out localhost '#host=localhost'. Specify 'localhost' if Flex is on this machine but not installed as part of ColdFusion.

For the Remoting-config.xml add the following code within the "ColdFusion" destination properties . You are setting use-mappings to true and setting the method-access-level to remote. use-mappings is a Boolean value specifying whether the source attribute can be relative to a ColdFusion mapping. Changing the method access level to remote basically specifies the access attribute values the destination CFC must have for ColdFusion to respond to the request. The code within the property-case tags simply state to make query column names, and structure keys lowercase when converting to ActionScript.

Remoting-config.xml

```

<destination id="ColdFusion">
  <channels>
    <channel ref="my-cfamf" />
  </channels>
  <properties>
    <source>*</source>
  <access>
    <use-mappings>true</use-mappings>
    <method-access-level>remote</method-access-level>
  </access>
  .....

```

Next you will need to adjust the access rights within the CFC function definition. Placing in whatever access level has been defined within the Remoting-config.xml. Flex can can access functions that are both remote or public. This CFC server side can just be a simple database query, so long as the function returns data to populate the data grid. The CFC in this example performs a query on a table within a defined data source. If you wanted to do the same, simply create a table within MYSQL or MSSQL and set it as a data source within the ColdFusion administrator.

Destination.cfc

```


```

```
<cffunction name="GetData" access="remote" returntype="query">
```

Configuring FusionDebug

Start up Eclipse with the FusionDebug Plug-in installed. Create a new Debug Configuration (Run->Debug Configurations...) for FusionDebug and configure the debugging port for your app server. In the IP Restrictions enter the IP address(es) of the server(s) that you want breakpoints to fire for. This means that when the request originates from one of the IP addresses you've entered it, the breakpoint should fire but for other requests the breakpoint will be ignored.

You then will have to navigate to the Source Code Lookup tab. Here you will simply need to select the project you want to debug and reference it's source in the directory. Add the lookup and you are ready to start debugging.

Start debugging by pressing the Debug button at the bottom of the debug configuration screen. Once the debugger is attached open the CFC and add some breakpoints where you want to stop execution of the code.

Now run your Flex application up and make a request from an IP address that is in the list of IP Restrictions. FusionDebug halts the activity for only the flash remoting requests whose IP addresses have been entered into the IP filter, so if you make a request for a different machine then the breakpoint doesn't fire. If you are using an app server on your local machine, CF8 and CF9 use loopback addressing and as a result uses the Ipv6 address 0:0:0:0:0:0:0:1 for localhost if you are using Windows Vista or 7 operating systems.

Whereas request from clients that are not in the IP restrictions list run to completion without the breakpoints firing! That's Non-Intrusive Debugging!

Messaging Gateways

Below is a short example of testing the non-Intrusive debugging functionality in regards to ColdFusion Gateway messages.

Create a Flex project with a main mxml that contains a script block with a function that creates a producer and consumer object along with event listeners and remote destinations.

MessagingGatewayTest.mxml

```
import mx.messaging.Consumer;
import mx.messaging.Producer;

private var publisher:Producer;
private var subscriber:Consumer;

private function createChat():void
{
    publisher= new Producer();
    publisher.addEventListener(MessageFaultEvent.FAULT,
producerFault);
    publisher.destination = "ColdFusionGateway";

    subscriber= new Consumer();
    subscriber.addEventListener(MessageAckEvent.ACKNOWLEDGE,
acknowledge);
    subscriber.addEventListener(MessageEvent.MESSAGE,
receiveMessage);
    subscriber.destination = "ColdFusionGateway";
    subscriber.subtopic = "test";
    subscriber.subscribe();
}
```

The you will need to write a function for the handling of the acknowledgement event from the consumer, functions for sending and receiving messages and a function for handling a fault event from the producer.

MessagingGatewayTest.mxml

```
private function producerFault(faultEvent:MessageFaultEvent):void
{
    output.text += "[Error: " + faultEvent.message.toString() + " ]
\n";
}
private function acknowledge(ackEvent:MessageAckEvent):void
{
    output.text += "[Contact with the Consumer has been established.]
\n";
}

private function sendChatMessage():void
{
    var msg:AsyncMessage = new AsyncMessage();
    msg.body = input.text;
    msg.headers["uname"] = "tester";
    msg.headers["gatewayid"] = "flexgateway";
    publisher.subtopic = "test";
    publisher.send(msg);
    input.text = "";
}

private function receiveMessage(msgEvent:MessageEvent):void
{
    var msg:AsyncMessage = AsyncMessage(msgEvent.message);
    output.text += msg.headers["uname"] + ": " + msg.body + "\n";
}
```

Within the content root of your CF app server (a.k.a wwwroot folder) you should place the CFC that you are using to interact with. This will contain a function that will return a message to the caller via the same destination and channel.

InterceptFlex.cfc

```
<cfcomponent output="false">
    <cffunction name="onIncomingMessage" returntype="any">
        <cfargument name="event" type="struct" required="true"
/>
        <cfscript>
            x = structNew();
            x.body = "Hello Friend";
            x.destination = "ColdFusionGateway";
            x.headers = structNew();
            x.headers['uname'] = "system";
            x.headers['DSSubtopic'] = "test";
            x.lowercasekeys = "yes";
        </cfscript>
```



```
<cfreturn x />
</cffunction>
</cfcomponent>
```

Some changes that may need to be made to CF8 and CF9 Application servers' configuration before Flash Gateway usage, the first being within the `corssdomain.xml` file, make sure the contents allows all domain policies, all domains and all ports. Obviously there would be security issues with the following changes, but since this is just an example of Flash Remoting we will allow all ports and domains, but in reality these should be appropriately restricted.

crossdomain.xml

```
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="all" />
  <allow-access-from domain="*" secure="false" to-ports="*" />
  <allow-http-request-headers-from domain="localhost" />
</cross-domain-policy>
```

Within the `messaging-config.xml` set the server properties as shown below within the destination "ColdFusionGateway" properties.

messaging-config.xml

```
<destination id="ColdFusionGateway">
  <adapter ref="cfgateway" />
  <properties>
    <server>
      < durable>false</durable>
      < allow-subtopics>true</allow-subtopics>
    </server>
  </properties>
  .....
</destination>
```

A subtopic is a property of both the producer and consumer components, within the xml you are configuring the destination "ColdFusionGateway" to allow the subtopic to be used as a message destination. Setting the server configuration durable to false allows for messaging without the JMS (Java Messaging Service) API. If you are running Flex server with ColdFusion comment out the "host=localhost" line within the `flex-messaging-gateway` config file. This is located within the root ColdFusion directory `\gateway\config\`, else if Flex is on this machine but not installed as part of ColdFusion specify localhost.

You can configure your app server gateways for CF8 and CF9 as shown below in order to create the appropriate gateway needed for this example:

The counter of your Gateway should be 0 before the test has been executed. After the test is run both the 'in' and 'out' counter should display one message each in their fields (The in-going message being the call for the CFC via the ColdFusion channel. The outgoing message being the response from the CFC I.E. the body of the `cfscrip`) When the program is first launched you should see the confirmation of contact from the Consumer.

Once some message has been entered, you should then see the message coming from the CFC.

FusionDebug halts the process of the CFC for specific users listed within the IP Restriction configuration options. All IP's not entered within the filtering box will simple run the app as normal.

CONCLUSION

The non-Intrusive debugging feature ultimately provides a development team with the ability to debug their code without affecting other team members. It could even be that you need to debug a request from a specific client because they are the only one that experiences

a problem. Non-Intrusive debugging can be used here to filter the requests that fire the breakpoints to only the problem client. This results in saving time and allowing bugs to be spotted earlier on within the development stage of production and can also lead to decreasing time consumed by debugging newly implemented components.

[FDS-130] FusionDebug: Feature Focus - Non-Intrusive Debugging

Non-Intrusive Debugging

Non-Intrusive debugging has been introduced in FusionDebug 3.5, which allows breakpoints to only fire when requests are coming from a specified IP address. This makes it possible for a developer to debug and set breakpoints against a central server, leaving the system untouched and unaffected for everyone else.

This article will cover how to configure and use Non-Intrusive Debugging, the underlying implementation and a few tips on what to do if your breakpoints don't fire when they are supposed to. The last section will also talk about the limitations of the Non-Intrusive Debugging feature in FusionDebug.

Configuration

Non-Intrusive Debugging can be enabled on any new or existent Debug Configurations in FusionDebug 3.5. This is done by entering the IP addresses to break on in the **IP Restriction** section available in the **Connect Tab** of the **Debug Configuration** to edit.

The IP Restriction feature in FusionDebug supports both IPv4 and IPv6 addresses, but there is no input validation so make sure these are entered correctly using standard IPv4 and IPv6 formats.

So for example say you only want breakpoints to fire when connecting locally, then you can enter 127.0.0.1 if connecting using IPv4 and 0:0:0:0:0:0:0:1 if using IPv6.

Multiple IP addresses can be specified by using a comma (,) to separate them, so in the previous example we can make sure that the breakpoints fire when connecting locally using either IPv4 or IPv6 by specifying them both like this: 127.0.0.1, 0:0:0:0:0:0:0:1

After the Non-Intrusive Debugging settings have been entered, click **Apply** to save the setting to the configuration. Please note that the debug session will have to be re-launched for the new IP Restriction properties to take effect.

Under the Hood

When a request is made to the server being debugged in FusionDebug the connected IP address is compared to the ones entered in the Debug Configuration of the launched session.

FusionDebug does this by evaluating the **CGI.REMOTE_ADDR** variable for the request with the ones specified in the Debug Configuration screen. If the `cgi.remote_addr` variable matches any of the ones entered the breakpoints will fire as usual, otherwise they will not.

It is important to understand that FusionDebug can only evaluate the value that the ColdFusion server has stored. For example if the connection to the server is going through a proxy, FusionDebug will compare the IP address of the proxy and not the end user's.

Also the entered IP addresses must exactly match the value of the `cgi.remote_addr` variable, no wildcards are allowed. For example you cannot enter 192.168.1.*, you will have to specify all the exact IP addresses to break on.

Adobe ColdFusion and Railo servers might also deal with different connection types differently and the value of `cgi.remote_addr` might differ slightly from server to server.

It is important to know that if the `cgi.remote_addr` variable cannot be evaluated, i.e. it has not been set properly, Non-intrusive Debugging will not function.

My Breakpoints don't fire

If you have enabled Non-Intrusive Debugging but the breakpoints do not fire or behave in the way you are expecting them to do, here are a few trouble-shooting tips.

IP addresses correctly entered

Make sure that the IP addresses to break on are specified correctly in the Debug Configuration using standard IP address notation and that no extra commas or dots have been introduced. Also make sure that the debug configuration settings have been saved.

IPv4 and IPv6

Make sure that you have not entered an IPv4 address when in fact the IP address reaching the server is an IPv6. Please note that different computers and operating systems will send different addresses depending on the network configuration. For example Microsoft Windows 7 has IPv6 enabled by default.

Also be aware that various servers will support different IP versions.

For example localhost can be specified as both an IPv4 and IPv6 address:

IPv4: 127.0.0.1

IPv6: 0:0:0:0:0:0:0:1

To be sure that the breakpoints fire on either the IPv4 or the IPv6 address both can be specified in the IP Restriction configuration, as in the image above.

Re-launch the debug session

It is not enough to just apply the new Non-Intrusive Debugging settings for the active Debug Configuration, but the actual debug session in FusionDebug has to be re-launched. This is because the IP Restriction properties are applied to the debug session at launch and cannot later be edited.

Inspect CGI.REMOTE_ADDR

If Non-Intrusive Debugging has been set up correctly but breakpoints still do not fire when they are expected to, a good thing is to actually check what value the `cgi.remote_addr` variable has been given.

One way to this is to have a CFML page containing this example code:

```
<cfoutput>remote_addr: #cgi.remote_addr#</cfoutput>
```

This will output the value of the `cgi.remote_addr` variable so that you can compare it to the ones entered in the Debug Configuration.

Another way is to inspect the value of `cgi.remote_addr` using FusionDebug. To do this you will have to disable Non-Intrusive Debugging by removing all the IP addresses in the **Addresses** field in the **IP Restriction** section of the Debug Configuration screen.

Once this has been done, apply the settings and re-launch the debug session and set a breakpoint on a page. Request this page from the machine you wish to enable Non-Intrusive Debugging for and the breakpoint should fire as normal.

You can now inspect the value of **CGI.REMOTE_ADDR** by selecting the **Variables** view tab in FusionDebug.

- **For Adobe ColdFusion:**

Select the **APPLICATION, REQUEST, SESSION, SERVER, CGI...** Variable scope.

In this scope select **CGI** and scroll down the list of variables and locate the variable called **REMOTE_ADDR**.

- **For Railo:**

Select the **cgi** Variable scope and scroll down the list of variables and locate the variable called **remote_addr**.

You can now copy the value of this variable and paste it in the **Addresses** field in the **IP Restriction** section of the Debug Configuration screen.

Proxy servers

If there is a proxy server between the user making the request and the server running ColdFusion, the value of the `cgi.remote_addr` variable will be set with the IP address of the proxy server and not the actual end user's. This can create problems if there are multiple machines behind a proxy server and you only wish to break on a specified local IP.

Clear existent breakpoints

If breakpoints still do not fire when they are supposed to or behave in an unexpected way, it is a good idea to go to the **Breakpoints** view tab and delete all of the existent breakpoints and re-add them.

Sometimes breakpoints from an older version of FusionDebug will not work properly in a newer version, also if there is one bad breakpoint it might cause issues for other enabled breakpoints.

Limitations

There are many ways a request can be made to a ColdFusion server, for example http, flash remoting, web services, messaging gateway and more. We have tested FusionDebug using several different requests types and there is a DevNet article available covering this in more detail, "Debugging Flash Remoting and Gateway Messaging".

As the Non-Intrusive Debugging feature in FusionDebug uses the value of the `cgi.remote_addr` variable for the request there will be cases when this variable is not being set and can therefore not be evaluated.

If you have specified IP addresses to filter breakpoints on and `cgi.remote_addr` doesn't actually contain a value the breakpoints will not fire as FusionDebug would not know where the request came from.

[FDS-118] OnDemand Seminar: FusionDebug Introduction (with Railo)

Using An Interactive Debugger To Increase Productivity

This recorded webinar reviews how to install FusionDebug and configure it for use with a Railo server in addition to discussing the many features FusionDebug offers.

Webinar Outline

- Installing FusionDebug
 - Connecting FusionDebug with Railo
- FusionDebug Feature Run-Down
 - Supported Platforms
 - ColdFusion 6, 7, 8, 9, 10, 11, 2016
 - Railo 3, 4
 - Lucee 4
 - Eclipse: 3.1 to 4.5
 - Conditional Breakpoints
 - Break on Runtime Exceptions
 - Run to Line
 - Advanced Stepping Technology
 - Source Code Lookups
 - View Variables and Scopes (including queries)
 - Expression Watcher
 - Multiple Request Support
 - Full Stack Traces (including function arguments)
- Summary

WATCH WEBINAR

1. FusionDebug Introduction (with Railo) - Part 1 of 6
2. FusionDebug Introduction (with Railo) - Part 2 of 6
3. FusionDebug Introduction (with Railo) - Part 3 of 6
4. FusionDebug Introduction (with Railo) - Part 4 of 6
5. FusionDebug Introduction (with Railo) - Part 5 of 6
6. FusionDebug Introduction (with Railo) - Part 6 of 6

GETTING STARTING WITH FUSIONDEBUG

- Download a free trial of FusionDebug

ABOUT FUSIONDEBUG

FusionDebug is the fastest interactive step debugger for CFML and supports multiple engines (ColdFusion, Railo and Lucee).

[FDS-125] Configure FusionDebug 3 for use with Railo 3.1 on Tomcat

INTRODUCTION

I ran into a situation while working on a MachII application running on Railo 3.1 where I REALLY need to see what was going on with the various variable scopes during the request cycle, so I decided tonight to see if I could get FusionDebug 3 Beta up and running with Railo and Tomcat.

For the past couple of months, I've been running my CFML server engines (yes engines, plural) on top of Tomcat on my local development environment. This offers me, as an independent developer that works on a number of different client projects, a great deal of flexibility in matching a particular client's production configuration. Also lately, I've been working on a couple of projects using Railo 3.1 as well as a project for a client that still uses CFMX 7. One of the things that I really missed when not developing with ColdFusion 8 is the step debugger that ships with CF8. I'd used FusionDebug some time ago with CFMX 7 when I was running it on top of JRun 4 but had never gotten around to getting it configured under my current, Tomcat-based setup.

Environment

Before we get started, I want to detail my particular setup and lay out some assumptions. My environment consists of the following components. Yours might be slightly different, but the basics should be the same.

- Apache 2.2.11 web server configured with one virtual host per client site
- Tomcat 6 installed into /opt/tomcat
- Apache forwards ColdFusion requests to Tomcat via AJP
- http://localhost web root is /users/dskaggs/Sites/localhost/htdocs
- WEB-INF folder from exploded Railo 3.1 WAR file resides in the web root folder

Now for some assumptions:

- You can browse to a CFML page (either by going directly to Tomcat `http://localhost:8080/someContext/test.cfm` or through Apache `http://localhost/test.cfm`)
- You've installed FusionDebug into Eclipse using the instructions in the "Install FusionDebug into Eclipse" section here.
- You've created a project in Eclipse that points to your site files
- You don't have Tomcat starting as a daemon or Windows service

Setting Up Tomcat

The first thing that you have to do is tell Tomcat to enable debugging on a specific TCP port. For this example the port number that we're going to use is 8000 (the same as the documentation on the FusionDebug site). You do this by adding the following line of code to the `catalina.sh` file in the `/bin` folder under the Tomcat install directory (in my case `/opt/tomcat/bin/catalina.sh`). I added it directly under the large comment block at the top of the file.

```
CATALINA_OPTS=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

If you're on Windows, this file is named `catalina.bat` and the format is slightly different:

```
set CATALINA_OPTS=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000
```

Save the file and launch Tomcat. In my case, I launched it in a terminal window so I could see exactly what was going on using the command:

```
cd /opt/tomcat/bin
sudo ./catalina.sh run
```

If everything worked correctly, you'll see the following line near the top of the output to the console:
Listening for transport dt_socket at address: 8000

That concludes the configuration necessary on the Tomcat side.

Configuring FusionDebug

Now we need to get FusionDebug configured to connect to that port. In Eclipse, go to `Window > Open Perspective > Other` and choose FusionDebug from the list that appears in the popup window.

Then you'll need to open up the Debug Configurations panel by going to `Run > Debug Configurations`. Find the entry for Fusion Debug in the left side, right click on it and choose "New" from the context menu. Enter a descriptive string in the Name: block at the top of the left side of the screen (I used simply Localhost). Then move to the configuration panel below with the 3 tabs (Connect, Source Code Lookup, and Common).

On the connect tab, enter localhost in the Host: block and 8000 in the Port: block.

On the Source Code Lookup tab, in the drop list that says "<All Projects>", choose the Eclipse project that contains your files. In the drop list just to the right of the Project select box you just clicked on, choose the folder that corresponds to the web root of your site. Finally, enter the full file system path that corresponds to the folder you just chose and click the Add button. Click the Apply button to save the configuration. You can then click the Debug button at the bottom of the screen to start the debug session. If all goes well, you'll see "FusionDebug (localhost:8000) (Connected)" in the Debug pane.

Testing

Now it's just a matter of creating a test.cfm file (or using an existing file), setting some breakpoints and browsing to the file and you should see the execution stop at your breakpoint and be able to browse through the list.

[FDS-121] FusionDebug: Feature Focus - Breakpoint Conditions & Hitcounts

Breakpoint Conditions & Hitcounts

FusionDebug breakpoints now have conditions and hitcounts, further controlling when they fire. This article will demonstrate these features.

Conditional Breakpoints

Introduced in FusionDebug 3.0, conditional breakpoints allow you to halt an application when a certain condition evaluates to true. For example, in a loop you could use a conditional breakpoint to halt when a variable equals a specified value. The condition used can be any form of CFML expression, anything that can be used in a `<cfif>` tag can be used as a condition. Advantages of conditional breakpoints include:

- Less time debugging; No need to step through long loops to get to a particular iteration.
- Prevents convolution of production code with debug code. To stop on an iteration or when a condition evaluates to true without conditional breakpoints you would need to add `<cfif>` tags to your code and set a breakpoint in the body of the tags. One of the advantages of a debugger is the alleviation of extra code for debugging purposes.

- Removes the need to disable breakpoints for some test cases, for example you might only want to stop a page on some specific input.

Example:

Given the following code:

```
<cffunction name="DoSomethingWithLastName">
    <cfargument name="lastname" type="string">
    <!--- Do something with last name arg --->
</cffunction>

<cfquery name="qry" datasource="test1">
    SELECT last_name FROM emp
</cfquery>

<cfloop query=query>
    <cfset DoSomethingWithLastName(qry.last_name)>
    <cfoutput> #qry.last_name# <br> </cfoutput>
    <cfflush>
</cfloop>
```

Suppose we knew there was a bug somewhere in the "DoSomethingWithLastName" function, and we knew it happened when the last name of "Smith" was used as an argument. We could add a breakpoint on the line of the call to the function and set the following breakpoint properties:

This means that the breakpoint will only fire when `qry.last_name` equals "Smith".

The page has stopped on row 9 of the query loop, where the `last_name` field equals "Smith". We can now step into the function and find the cause of the bug. Note that this is a relatively small query but could be thousands of rows. Using a non-conditional breakpoint in this scenario is time-consuming and unfeasible, plus you could end up resuming past the row that contains "Smith".

Other useful examples of breakpoint conditions:

- **`isdefined("url.length") and (url.length + 1) gt 10`**. This can be used to halt the page when certain url parameters are used, in this case `length > 10`.
- **`DateCompare(yourdate, CreateDate(2009, 01, 01)) LTE 0`**. This will halt the page when "yourdate" is before 1st January 2009.
- **`http.remote_addr eq "127.0.0.1" or http.remote_host eq "localhost"`**. These conditions will allow another machine to request the file without the breakpoint firing. Useful if you have to demo the file to another person but don't want to disable your breakpoints. Note, on Railo use **`cgi.remote_addr eq "127.0.0.1"`**.

Read more about Conditional Breakpoints

HitCounts

New to FusionDebug 3.0.1 are Breakpoint Hitcounts. Hitcounts are a specific type of condition for a breakpoint separate from an actual breakpoint condition. A breakpoint with a hitcount will only fire after a certain amount of hits. This is useful if you have no variables to use in a condition.

Hitcounts can be used along with conditions to fine tune the firing of the breakpoint. For example consider the following loop over a query variable:

```
<cfloop query="qry">
    ... use data from qry
</cfloop>
```

If a breakpoint was on line 2 we could use a hitcount of 15 and a condition of "qry.last_name eq 'Smith'".

This would stop the page on the 15th row that had the last_name field equal to "Smith"

Read more about Hitcounts

Conclusion

These two features will cut your debugging time down by letting FusionDebug work for you, evaluating conditions that would otherwise need to be carried out by yourself, watching the Variables or Expressions view for changes. Most modern debuggers support Conditional Breakpoints and Hitcounts because as applications become more complex, debugging them becomes more complex and in response debuggers must relieve some of this complexity. These two features and countless others do just that.

[FDS-122] FUSIONDEBUG: FEATURE FOCUS - AUTO-STEP

Auto-step

New to FusionDebug is the Auto-step feature. Interactively watch your code being executed at a specified speed.

The new Auto-step feature adds even more interactivity, enabling you to view execution paths of your application, variable / expression changes and even root out bottlenecks in your code.

Using Autostep

You can set an interval between step operations in the FusionDebug Configuration dialog

There you can set intervals for both auto-stepping into or over.

You control auto-stepping with two toggle buttons on the Debug View toolbar.

The button on the left is Auto-step Into, the right one is Auto-step Over. Only one can be toggled at any time so toggling on Auto-step Into when stepping over will cause Auto-step Over to be toggled off.

How Autostepping Helps

Auto-step can be used in a number of interesting ways:

- By setting a moderately high stepping interval you can visibly watch the path of execution your application takes. This is very powerful as many bugs are simply code you think is being executed when in actuality it isn't.
- By setting a high interval, say 500ms - 1000ms, you can watch the Variables / Expressions view and for any changes that should or should not be happening.
- By setting a low interval (lowest 20ms) you can visibly see what parts of your code are taking a long time to complete. If there are long pauses between steps (longer than the step interval you entered) you know that the line should be investigated, if need be.
- Auto-step could also be used for entirely other purposes than debugging. For example, introducing a new developer to an application and its code can be time consuming and consequently expensive. By using Auto-step with an appropriate interval, the developer can watch where the execution path goes and learn what is being done and where.