



Rapid SQL Developer Debugger® 2.0 User Guide

Copyright © 1994-2008 Embarcadero Technologies, Inc.

Embarcadero Technologies, Inc.
100 California Street, 12th Floor
San Francisco, CA 94111 U.S.A.
All rights reserved.

All brands and product names are trademarks or registered trademarks of their respective owners.

This software/documentation contains proprietary information of Embarcadero Technologies, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

If this software/documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with Restricted Rights, as defined in FAR 552.227-14, Rights in Data-General, including Alternate III (June 1987).

Information in this document is subject to change without notice. Revisions may be issued to advise of such changes and additions. Embarcadero Technologies, Inc. does not warrant that this documentation is error-free.

Contents

Welcome to Rapid SQL Developer Debugger 2

 Overview 2

 Debugging SQL Code..... 4

 Displaying Variables 5

 Setting Breakpoints..... 6

 Debugging Stored Procedures 6

Using Rapid SQL Developer Debugger

Rapid SQL Developer Debugger is an Eclipse plug-in that provides the functionality to debug SQL code in the development environment prior to code execution.

This section is composed of the following sub-sections:

[Overview](#)

[Debugging SQL Code](#)

[Displaying Variables](#)

[Setting Breakpoints](#)

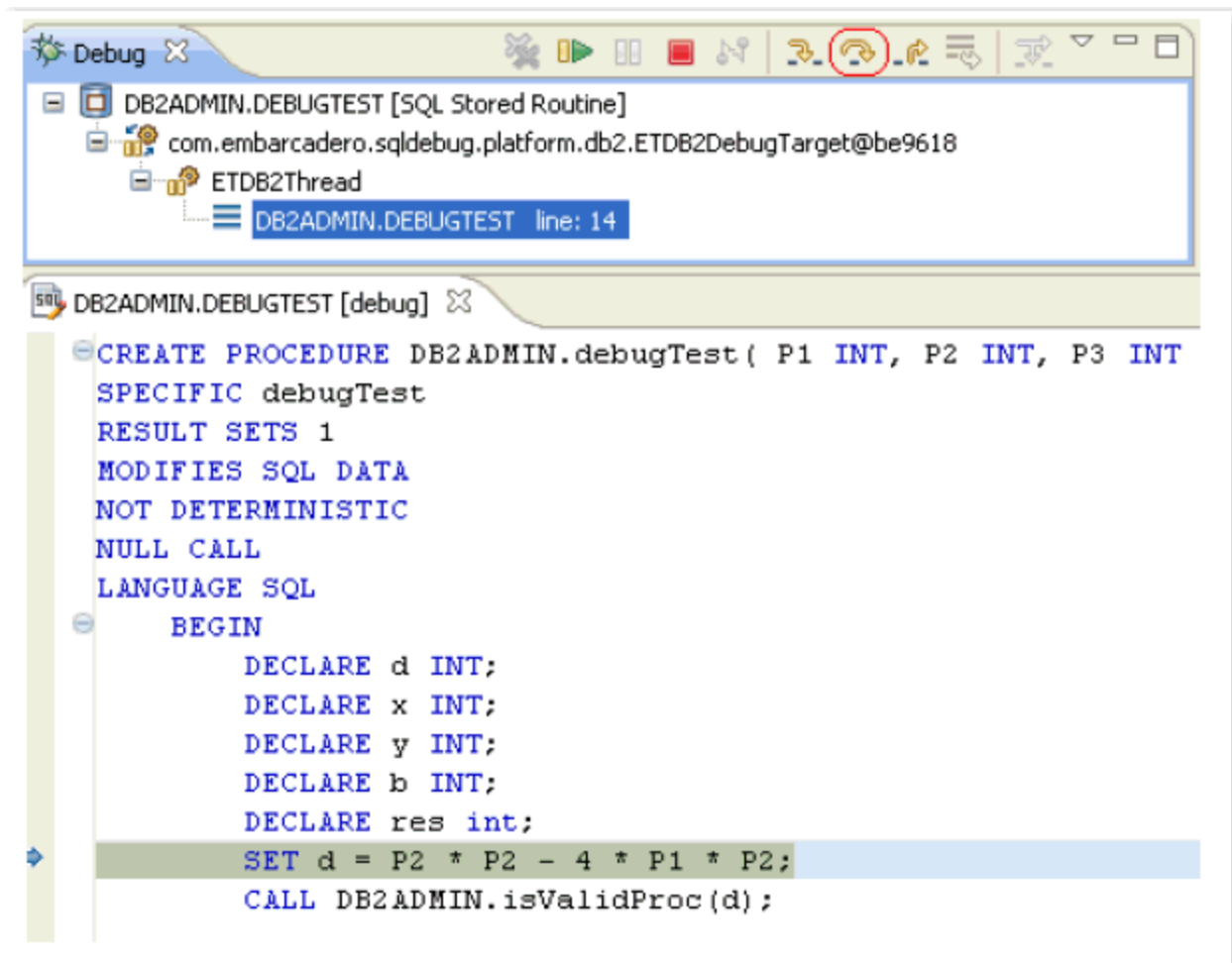
[Debugging Stored Procedures](#)

[Using Cross-Language Debugging](#)

[Configuring the Debugger](#)

Overview

The debug process is managed by the **Debug** view, and enables users to manage common debug functionality such as breakpoints and step over, as well as providing additional views and functionality to sort and handle variables, and compile dependencies.



Once the plug in is installed, you can debug code from **Data Source Explorer**. Expand the data source tree until you find the file that you want to debug, right-click on it, and then select **Debug As**, and specify how you want to debug it.

When the **Debug** view appears, you start the debug process via commands in the **Run** menu.

In addition to having the ability to run a debug process on code, the following additional commands are available, and represent common commands needed for debugging and other purposes:

- **Breakpoint Management Commands** (Toggle Breakpoint, Toggle Line Breakpoint, Toggle Method Breakpoint, Toggle Watchpoint, Skip All Breakpoints, Remove All Breakpoints)
- **Step Over Commands** (Step Into, Step Over, Step Return)
- **Execution Commands** (Debug History, Debug As, Open Debug Dialog)
- **Other Commands** (Debug History, Debug As, Debug Dialog)

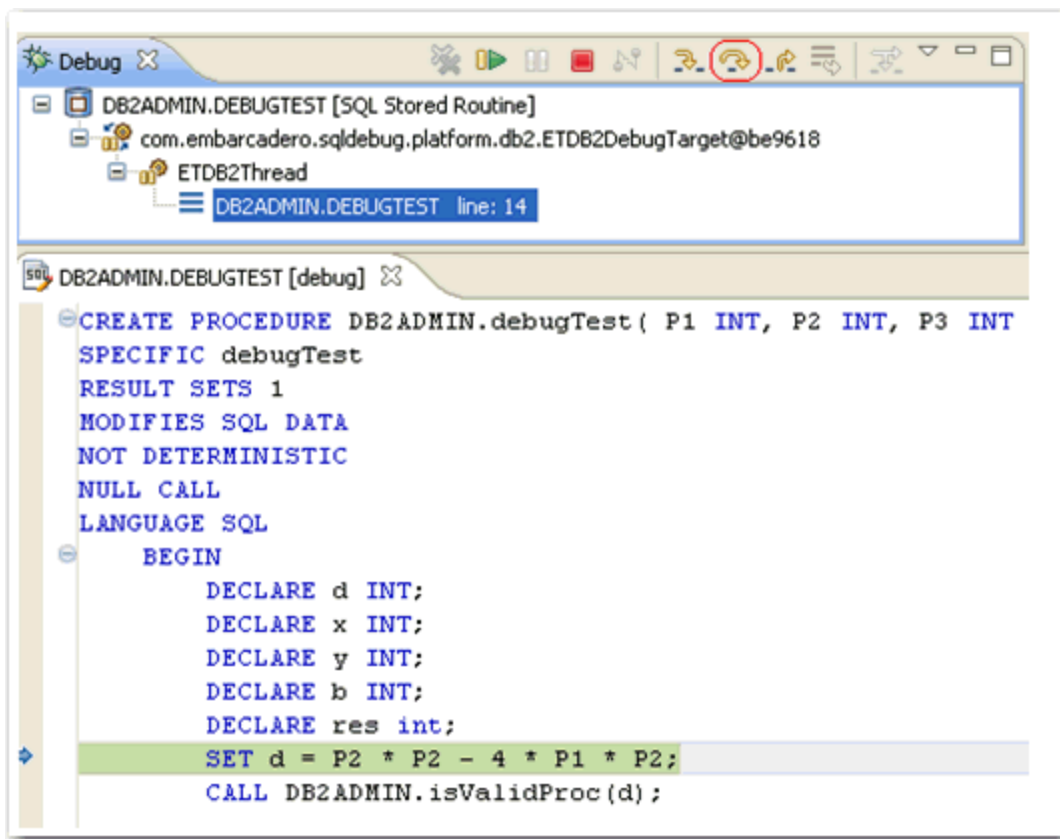
In addition to common commands, the debug feature also provides a number of supportive views to handle things such as variables, breakpoints, and dependencies in stored procedures.

- The **Variables** view is a complementary view to the **Debug** view. It displays the values of a specified variable during the debug process. For more information about how the debugger handles variables, see [Displaying Variables](#).

- A breakpoint is a placeholder within code undergoing the debug process. It notifies the process to temporarily suspend execution. Breakpoints can be used to halt the debug process so that modifications can be made to the code, or to change the debug execution point to only debug a certain section of a larger file. For more information on how the debugger handles breakpoints, see [Setting Breakpoints](#).
- When you debug a stored procedure, all routines that are referenced by the procedure appear in the **Dependency** view. If these routines have not yet been compiled via the debug option, the debugger will prompt you to compile those dependencies prior to executing the debug process. For more information on how the debugger handles dependencies, see [Debugging Stored Procedures](#).

Debugging SQL Code

The **Debug** view enables you to view and debug SQL code.



To debug SQL code:

In **Data Source Explorer**, expand the data source tree until you find the file you want to debug and right-click on it, then select the **Debug As** command and choose how you want to debug the code. You can debug stored routines such as procedures and functions, in addition to standard code.

When you start the debugging process, the **Debug** view appears. It displays the code in the file and enables you to execute debugging commands via the **Run** menu.

Additionally, the debug process contains a step over feature that suspends execution on the first line of a file. To resume the process, press **F5** or click the **Step Over** button located on the **Debug** view menu bar. **Step Over** can also be unlocked via the **Run > Step Over** command in the main menu.

Other commands available via the **Run** menu include common debug commands such as:

- Breakpoint management commands (**Toggle Breakpoint**, **Toggle Line Breakpoint**, **Toggle Method Breakpoint**, **Toggle Watchpoint**, **Skip All Breakpoints**, **Remove All Breakpoints**)
- Other step over commands (**Step Into**, **Step Over**, **Step Return**)
- **Resume** and **Terminate** commands
- Execution commands (**Run History**, **Run As**, **Open Run Dialog**)
- Other debug commands (**Debug History**, **Debug As**, **Open Debug Dialog**)

NOTE: The debugging function is supported on Oracle, Sybase, and DB2 platforms only.

See Also:

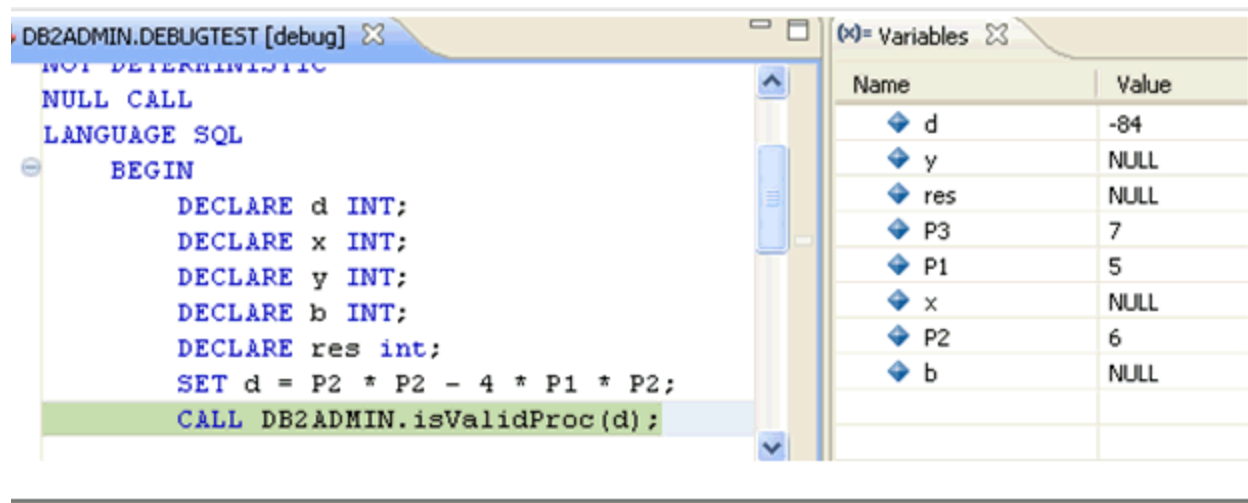
[Displaying Variables](#)

[Setting Breakpoints](#)

[Debugging Stored Procedures](#)

Displaying Variables

The **Variables** view displays the value of a variable during the debug process.

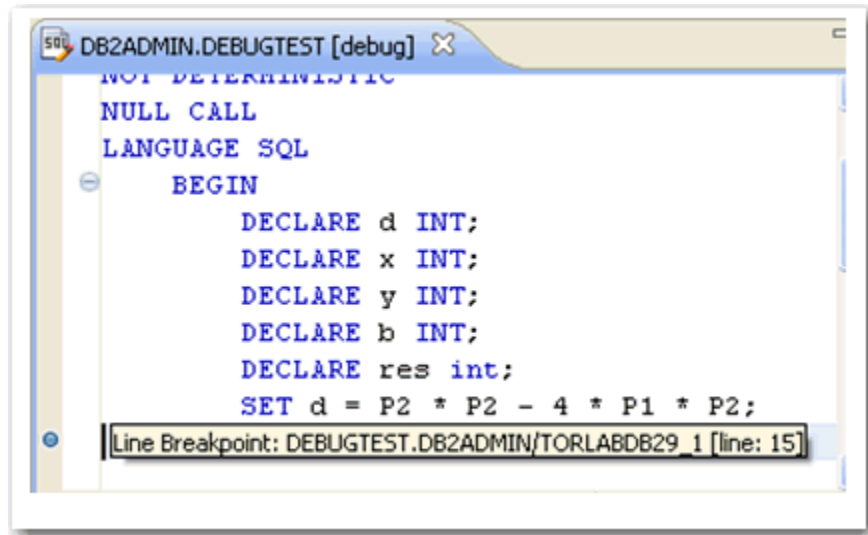


To view the value of a variable:

- Click on a variable in the **Debug** results view. Its value automatically appears in the **Variables** view.

Setting Breakpoints

A breakpoint is a placeholder within code undergoing the debug process that notifies it to temporarily suspend execution. Breakpoints can be used to halt the debug process so you can make modifications to the code (such as changing the value of a variable), or change the debug execution point to only debug a specific section of the total code.



To set a breakpoint:

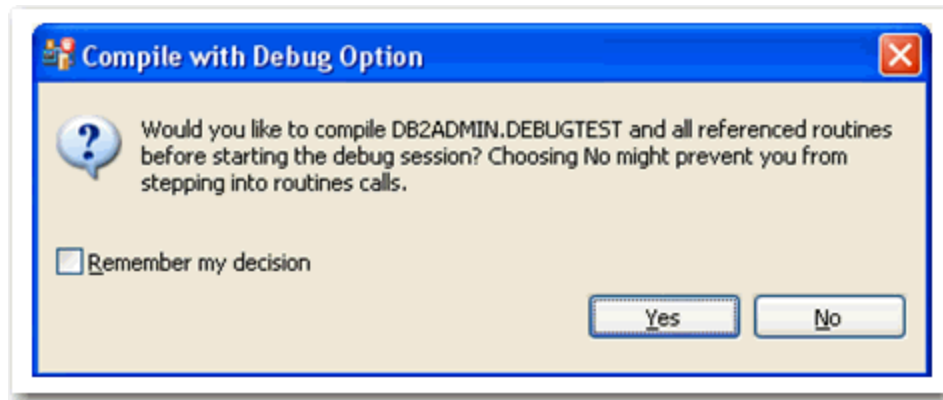
- Click the line in the code where you want to place a breakpoint. The breakpoint is indicated by a blue circle icon in the left-hand column.

To disable a breakpoint:

- To disable a breakpoint, right-click on the breakpoint icon and choose **Disable Breakpoint**. The blue circle is replaced with a white circle, indicating that the breakpoint no longer halts the debug process. You can enable a breakpoint again at any time via the **Enable Breakpoint** command in the context menu.

Debugging Stored Procedures

When you debug a stored procedure, all routines that are referenced by the procedure appear in the **Dependency** view. If these routines have not yet been compiled via the debug option, Rapid SQL Developer prompts you to compile the dependencies prior to executing the debug process.



If you select **No**, the debug process will execute regardless, but you will not be permitted to step into the uncompiled reference routines.

Using Cross-Language Debugging

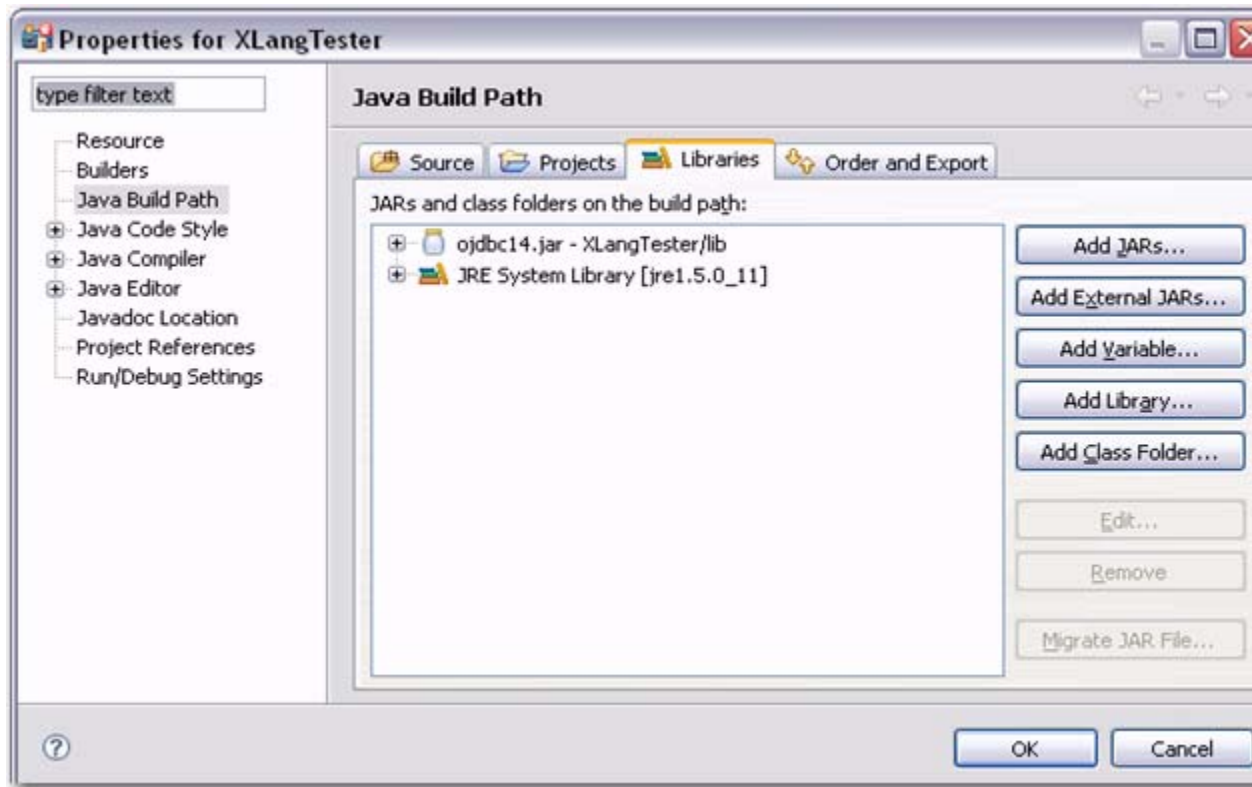
The Debugger features Java-to-SQL cross-language debugging. That is, debugging functionality in Java files that contain stored SQL routines.

In order for cross-language debugging to be enabled, you need to have Rapid SQL Developer installed as its Eclipse plug-in version. This provides you with the ability to add the Java application to the system prior to running the debug process. For more information on installing Rapid SQL Developer as an Eclipse plug-in, refer to the Installation Notes under [Technical Requirements](#).

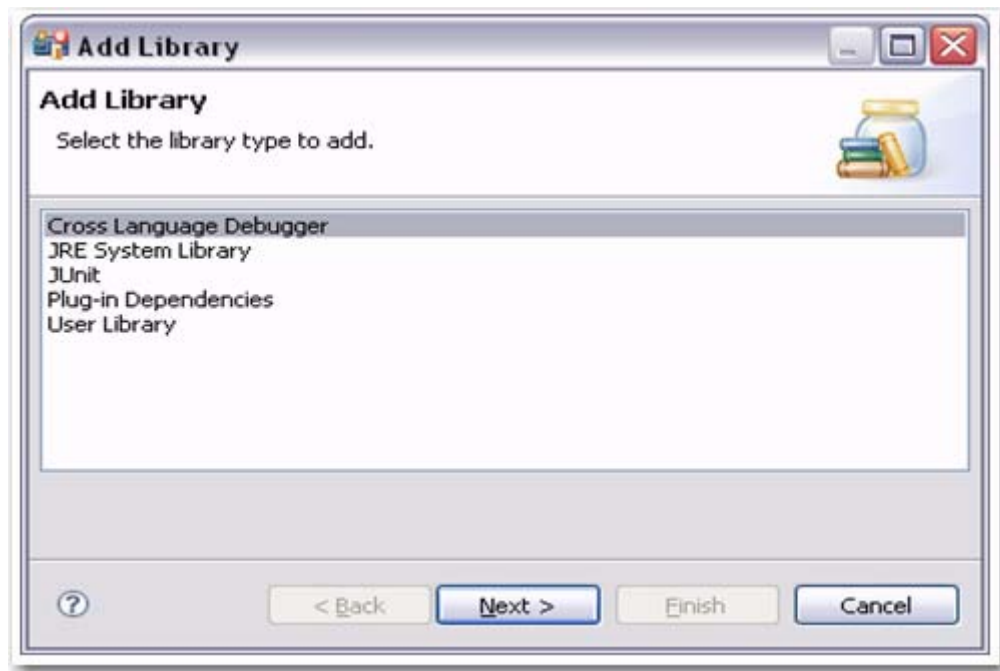
Additionally, once you have added a Java application to Eclipse, there are a number of configuration steps that must be performed in order for the cross-language debugging process to work correctly.

To configure a Java application for cross-language debugging:

- 1 Ensure the Java application includes the **Cross Language Debugger Library** in its classpath by right-clicking on it and choosing **Properties**. The **Properties** dialog appears.



- 2 Select **Java Build Path** and click **Libraries**. If the **Cross Language Debugger** does not appear in the list, click **Add Library**. The **Add Library** dialog appears.



- 3 Select **Cross Language Debugger** and click **Next** and then **Finish**. The **Cross Language Debugger** library is now associated with the Java application.

- 4 Add the following protocol to the **DriverManager**:

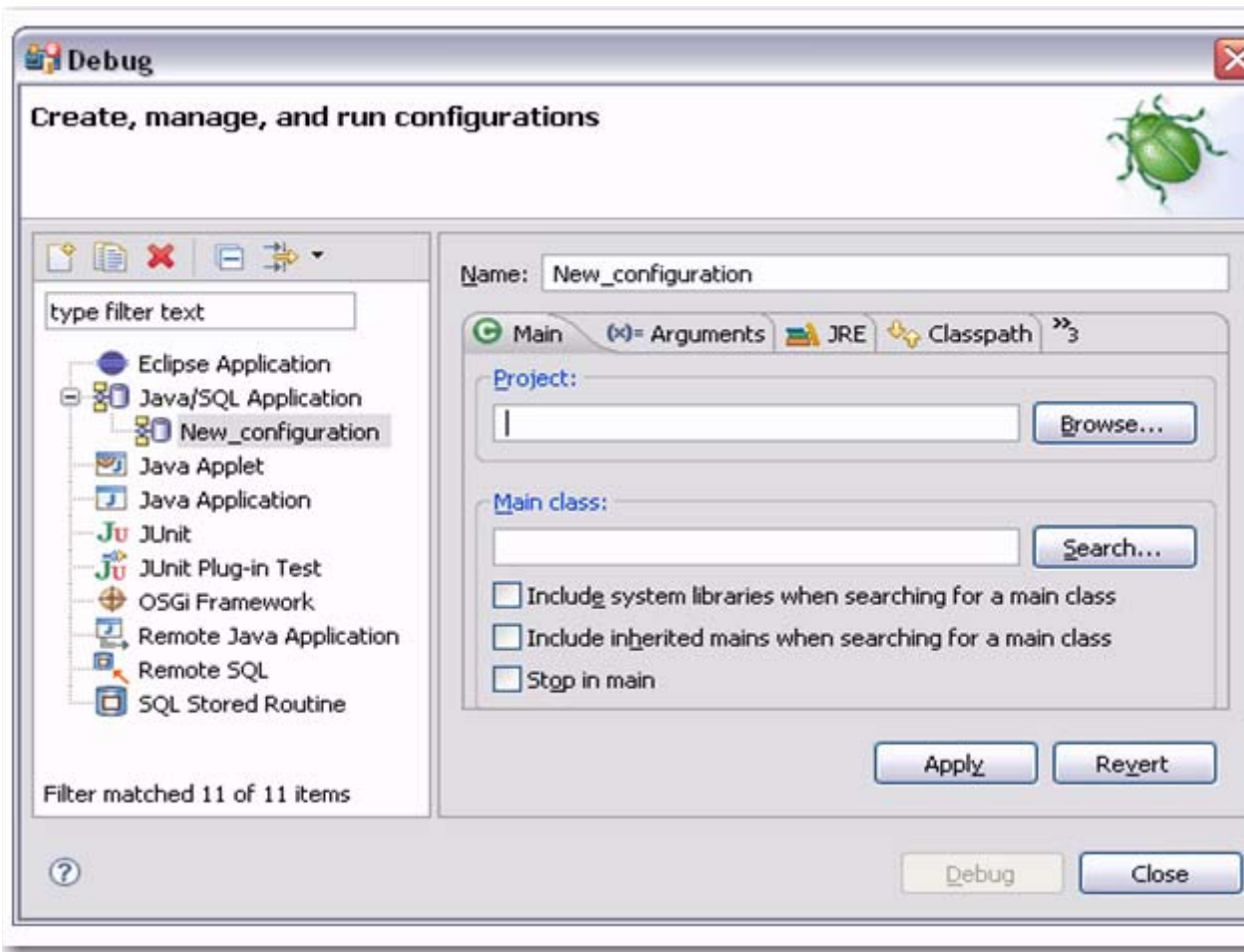
`etjdbc:xlangu:platform:@host:port:otherInfo`

- Where *platform* is **Oracle**, **Sybase**, **DB2**, or **MS SQL Server**
- Where *host* and *port* the address of the machine where the server resides.
- Where *otherInfo* represents any additional information required for connection purposes. For example, the SID on Oracle platforms.

- 5 Register the following driver:

`com.embarcadero.sqldebug.xlang.driver.DriverName`

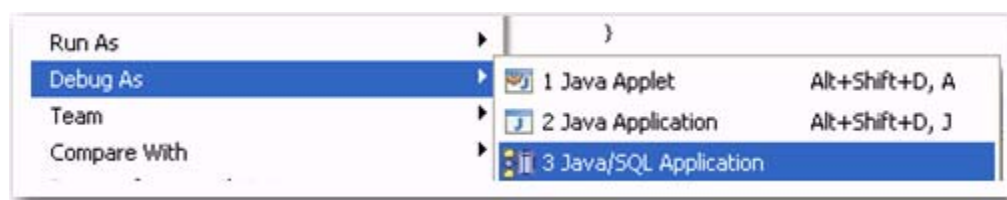
- Where *driverName* is **ETOracleDriver**, **ETSybaseDriver**, **ETDB2Driver**, or **ETSQLServerDriver**.
- 6 Create a new launch configuration named **Java/SQL Application**. Use the information as it pertains to the Java application you need to debug.



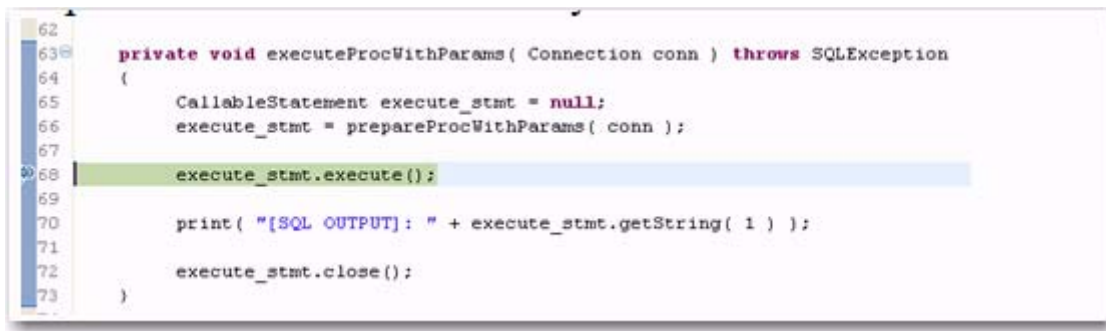
Once you have added and configured the Java application in Eclipse, select it and choose **Run > Debug As > Java/SQL Application** from the Main Menu.

To debug a java application that contains a SQL routine:

- 1 Select the Java application and choose **Run > Debug As > Java/SQL Application** from the Main Menu. The Debug dialog appears and the debugging process on the java code begins.



- 2 Navigate to the line of Java code that calls for the stored procedure, you can step into it by pressing **F5**.



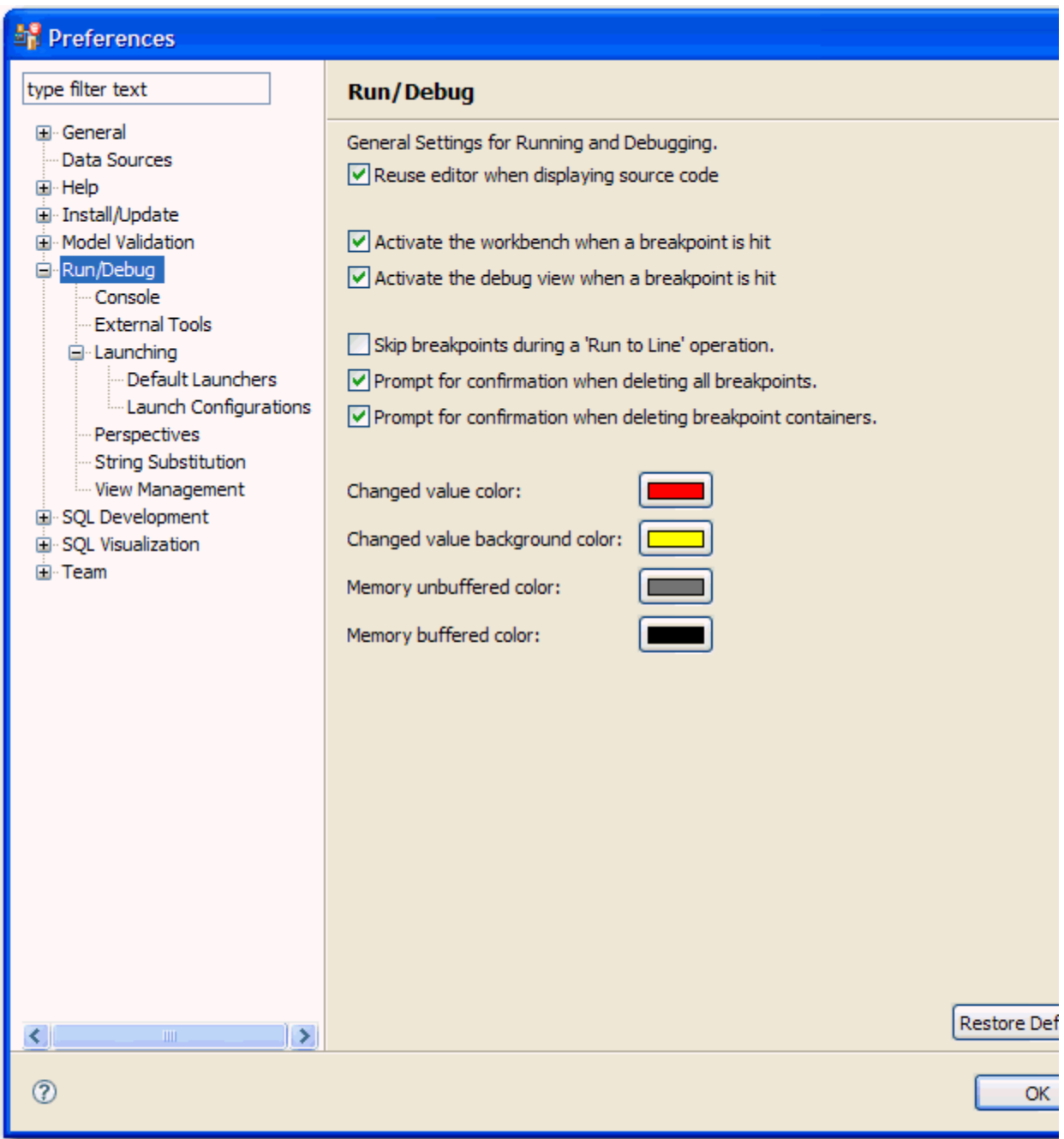
```
62
63 private void executeProcWithParams( Connection conn ) throws SQLException
64 {
65     CallableStatement execute_stmt = null;
66     execute_stmt = prepareProcWithParams( conn );
67
68     execute_stmt.execute();
69
70     print( "[SQL OUTPUT]: " + execute_stmt.getString( 1 ) );
71
72     execute_stmt.close();
73 }
```

- 3 The Debug process automatically isolates and suspends the stored procedure, and you can then initiate the SQL debug process, as normal.
- 4 When the process is finished, debugging on the Java code continues.

The Debug dialog appears. Navigate to the line of Java code that calls for the stored procedure, and step into the call by pressing **F5**.

Configuring the Debugger

To customize Debugger, select **Window > Preferences > Run/Debug**, and select a subnode, as required.



Change the configuration settings of the run/debug process, as appropriate:

- The general settings on the **Run/Debug** panel control how the workbench, views, and editor responds to various aspects of the debug and debug execution process. You can change the look and feel of text output, as well as specify how the interface handles breakpoints.
- The **Console** node controls aspects of the debug window, specifically. This includes buffer size, a fixed width option, and text colors for different console results.
- The **External Tools** node specifies how the Debugger handles project migration when executing from third-party tools.

- The **Launching** node and its corresponding sub-nodes (**Default Launchers** and **Launch Configurations**) determine the behavior of the Debugger when launching code, such as whether to automatically save editors prior to launching and how to filter or remove configurations from closed or deleted projects.
- The **Perspectives** node enables you to specify how the collection of debug windows behave when you run a debug session.
- The **String Substitution** node enables you to create and configure string substitution variables.
- The **View Mananagement** node enables you to specify what perspectives the **Debug** view is available in. By default, the **Debug** view is only associated with the **Debug** perspective.