

How to use the PL/SQL Debugger In TOAD

By

Steve C. Chapman
TOAD Development Team
Quest Software, Inc.
Last Revised: Friday, July 16, 1999



**the Tool for Oracle
Application Developers**

This guide contains proprietary information, which is protected by copyright. The information in this guide is subject to change without notice and does not represent a commitment on the part of Quest Software. The software described in this guide is furnished under a license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of this agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

1999 Quest Software, Inc. All Rights Reserved.

Quest is a trademark of Quest Software, Inc.

Quest Software, Inc.

610 Newport Center Drive, Suite 1400
Newport Beach, CA 92660

USA

Tel. (949) 720-1434 / Fax. (949) 720-0426

support@quest.com

www.quest.com

TOAD™ is a trademark of Quest Software, Inc. Other trademarks and registered trademarks used in this guide are the property of their respective owners.



Table of Contents

What's New	1
Installation	2
TOAD Version.....	2
Quest Software Registration Key.....	2
Copying Files.....	3
Other Beta Notes.....	3
Minimum Oracle Database Requirements	3
Objects You Can Debug	4
Objects You Cannot Debug	4
Requirements to use the Debugger	4
Starting and Stopping The Debugger.....	4
Starting the Debugger	4
Stopping the Debugger	5
Menus, Toolbars, Windows, and GUI	5
The Debug Menu	5
The Debug Toolbar Buttons.....	6
The Dockable Windows Feature.....	7
The Status Panel Indicators.....	8
Debug Functions.....	9
Starting With or Without Argument Values	9
Setting Argument Values.....	9
Execute Procedure or Function	10
Execute Package	11
Execute Trigger: INSERT.....	11
Execute Trigger: UPDATE.....	12
Execute Trigger: DELETE	14
Viewing OUT Argument Values	15
Dockable Windows.....	16
Breakpoints	16
Setting or Adding Breakpoints.....	16
Editing a Breakpoint	16
Conditional Breakpoints	16
Pass Count Breakpoints	17
Conditional, Pass Count Breakpoints.....	18
Enabling or Disabling a Breakpoint.....	18
Deleting a Breakpoint	18
Breakpoints Right Mouse Menu	18
Watches.....	19
Adding a Watch	19
Editing a Watch	19
Enabling or Disabling a Watch	20
Deleting a Watch.....	20
Watches Right Mouse Menu.....	21
Call Stack.....	21
DBMS Output.....	21
Evaluate/Modify	22
Keyboard Shortcuts.....	23
General Debug Options.....	23
Preparing PL/SQL Code for Production	25

What's New

(1). “Trace Out” toolbar button and corresponding Debug menu item has been added to execute to the end of a called procedure back to the caller, to continue debugging the caller. <Shift>F8 is the shortcut keystroke to invoke Trace Out from the keyboard.

(2). “Compile Dependencies with Debug Information” toolbar button and function has been added to compile referenced procedures with debug information, so that when your procedure steps into another called procedure, the debug information will be available. Note that the arrow in the icon points down indicating the hierarchy, as opposed to the up arrow for the “Compile Dependencies” toolbar function.

Also, “Compile Used Objects” has been added to the procedure editor right click Debug submenu.

(3). Icon for the “Set Parameters” button changed from lightning bolt and plus sign to parentheses containing ellipses.

(4). Icon for “Compile Dependencies” changed to include up arrow indicating that procedures that call your procedure will be recompiled. This is needed because after you recompile your procedure, Oracle marks all other procedures that call your procedure as INVALID, forcing you to recompile them.

(5). On the “TOAD Options” dialog, Debugging tab, there is a collection of radio option buttons for the type of Compile Dependencies with debug information to choose from. As you start debugging a given procedure, you can select how to compile procedures called by your procedure.

Compile Dependencies Option	Result
Yes	Just before TOAD begins to debug your procedure, it automatically compiles all procedures called by your procedure with debug information.
No	The debugger will not compile procedures called by your procedure. Note that you will receive “[No Debug Information Available]” messages if you step into other procedures while you are debugging.
Prompt	Just before the debugging begins, you will be prompted whether or not you want all procedures called by your procedure to be compiled with debug information.

(6). “Program Reset” on the Debug menu changed to “Halt Execution”. Does the same thing.

(7). The init.ora parameter is BLANK_TRIMMING, not BLANK_TRIM.

(8). The minimum Oracle database version on Unix to support PL/SQL debugging is 7.3.4, not 7.3.3 as was previously reported.

(9). “Output OUT Args” check box enabled on the “Execute Procedure” dialog. Section on “Viewing OUT Argument Values” added to this document.

In the “Product Authorization” window, enter the software registration key given to you by Quest Software, in the “Enter authorization key:” text box. Click the <OK> button. Open a new “Stored Procedure Edit/Compile” window. The items on the Debug menu should now become enabled.

The software registration key is a string of numbers and dashes that looks like this:

0-00000-00000-00000-00000

Copying Files

If you have a working installation of TOAD Commercial or Evaluation, then make a backup copy of your TOAD.EXE file, and copy over it with the PL/SQL Debugger version of TOAD.EXE. There are no additional files required for the PL/SQL Debugger. All of the code is contained within the TOAD.EXE application file.

Other Beta Notes

This version also has a beta version of a true object browser to browse Oracle 8 objects. Thus, the window name “Object Browser” has been changed to “Schema Browser” to avoid confusion. To open the Object Browser, select “Object Browser” from the “Database” menu.

For example, a Pet-type object, PET_T, could be defined as a Pet tag number and a Pet name:

```
type pet_t as object (tag_no integer, name varchar2(60))
```

Minimum Oracle Database Requirements

For all databases, you must have the Oracle Probe API installed in order to debug PL/SQL using TOAD. Check for the existence of a package named DBMS_DEBUG in the SYS schema.

The debugger works on these database versions with the following caveats:

Database Version	Notes
Oracle 7.3.4 on Unix, Oracle 7.3.4 on NT	Inspect package variables by stepping into the package first, then add the watch on the package variable. For the Call Stack to display, you must set the BLANK_TRIMMING value to TRUE in the init.ora Oracle Initialization parameters file and restart your database. Otherwise, you will receive a load error invoking the Call Stack window. Refer to your Oracle documentation regarding the effects of the BLANK_TRIMMING setting.
Oracle 8	(No notes or known issues. The PL/SQL Debugger works just fine on Oracle 8 databases.)
Oracle 8I	(No notes or known issues. The PL/SQL Debugger works just fine on Oracle 8I databases.)

Objects You Can Debug

You can debug these kinds of Oracle objects:

- (1). Top-level Functions
- (2). Top-level Procedures
- (3). Package Functions
- (4). Package Procedures
- (5). Triggers

Objects You Cannot Debug

You cannot debug these kinds of Oracle objects:

- (1). Java classes
- (2). Oracle 8 object methods

Requirements to use the Debugger

You must have the “Enable compiling multiple objects from a single file” option unchecked on the TOAD Options dialog, Procedure Editor tab in order to use the debugger. You cannot debug a file containing multiple PL/SQL objects. There has to be a one-to-one correspondence from the lines of source in the editor to the lines of source in the database.

When you uncheck this option, the toolbar on the Stored Procedure Edit/Compile window will change. Toolbar buttons 2 (execute FROM the cursor position), 3 (execute TO the cursor position), and 4 (run current statement) will disappear. The multiple object dropdown list will disappear. Also, the 6 debug buttons (Execute, Set Parameters, Step Over, Trace Into, Halt, and Add Watch) will appear on the right side of the toolbar.

Starting and Stopping The Debugger

NOTE: The word “Procedure” refers to PL/SQL code including Procedures, Functions, Package Procedures, Package Functions, or Triggers.

Starting the Debugger

Open up the “Stored Procedure Edit/Compile” window, from the “Database|Stored Procedure Edit” menu item, or the “Open a new Procedure Edit Window” button on the main TOAD toolbar.



Load a PL/SQL procedure into the editor (either a file on disk or an existing object from the database), compile the procedure via F9 or the “Compile” button on the Stored Procedure editor toolbar, then press F7 (Trace Into) to start stepping through the code. This will automatically generate the symbol table required to obtain debug information for this procedure only.

If you intend to step into other procedures and view debug information in them, then you will need to click the “Compile with Debug” toolbar button before beginning the debug process. Optionally, you can set

TOAD to always compile these procedures, never compile these procedures, or be prompted whether or not to compile these procedures. See the “TOAD Options” dialog, “Debugging” tab.

Stopping the Debugger

To stop the debugger, select the Halt toolbar button, or the “Debug|Halt Execution” menu item. The status panel will reset from “Running” to “Idle”.

When you have finished debugging your PL/SQL code, you need to compile it once again via F9 to discard the symbol table.

Menus, Toolbars, Windows, and GUI

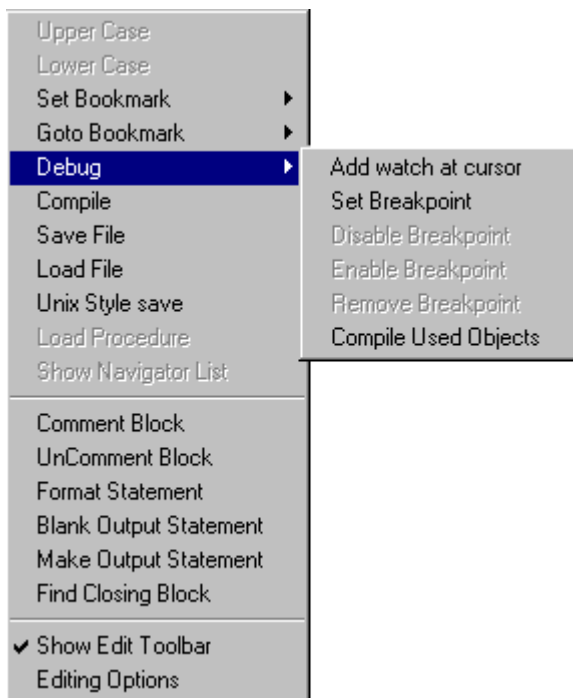
The Debug Menu

Debug	
Execute current source	Shift+F9
Set parameters	Ctrl+F9
Run (continue execution)	F11
Run to Cursor	F12
Step Over	F8
Trace Into	F7
Trace Out	Shift+F8
Halt Execution	
Add Breakpoint at cursor	F5
Evaluate/Modify	Ctrl+Alt+E
Breakpoints	Ctrl+Alt+B
Call Stack	Ctrl+Alt+S
Watches	Ctrl+Alt+W
DBMS Output	Ctrl+Alt+D

Function	Description
Execute current source	Starts debugging and runs to the next breakpoint or end of procedure with the current argument settings (set using “Set Parameters”).
Set Parameters	Presents a dialog to set the IN argument values, and, in the case of a Package, allows you to select which package procedure or package function to debug.
Run (continue execution)	Once you are stepping through the code, this function runs to the end of the procedure or to the next breakpoint, whichever it encounters first.
Run to Cursor	Once debugging has begun, runs to the cursor location as if it were a breakpoint, and stops.
Step Over	Executes one line of code at a time, bypassing a procedure or function call.
Trace Into	Executes one line of code at a time, stepping into other procedures as they are called. NOTE: only the top-level procedure will have debugging information available for it. If you step into another procedure and want to view debugging information, use the SQL Edit window and [alter procedure proc_name compile debug]. Otherwise the message, “no debug information available” will appear in the watch window.
Trace Out	Executes to the bottom of the called procedure, returning to the caller to continue debugging the caller.

Halt Execution	Stops stepping through the code, retaining watch and breakpoint settings.
Add Breakpoint at cursor	Adds or removes a breakpoint at the cursor location.
Evaluate/Modify	Brings up a window where you can, on the fly, inspect and/or change values of variables and continue execution with the new values.
Breakpoints	Brings up a dockable window of the currently set breakpoints, allowing you to add, edit, delete, enable or disable breakpoints.
Call Stack	Brings up a dockable window of the current procedure or function call stack (which procedures called which other procedures). This list is meaningful only during execution, as indicated by the “Running” light in the status panel. The Call Stack window is blank when you are at “Idle”.
Watches	Brings up a dockable window of the current variables being watched, allowing you to add, edit, delete, enable or disable watches.
DBMS Output	Brings up a dockable window for displaying DBMS Output generated from the procedure code. Note that the DBMS_OUTPUT content is not released from the database (and therefore not displayed) until all procedures have finished, or you force it to stop via the Halt button or Program Reset menu item.

There is also a Debug menu in the Stored Procedure Edit/Compile window. Right click over the editor, select “Debug” from the menu, and several Debug functions will appear on the submenu.



The Debug Toolbar Buttons

The Debug toolbar, contained on the right half of the “Stored Procedure Edit/Compile” toolbar, looks like this:

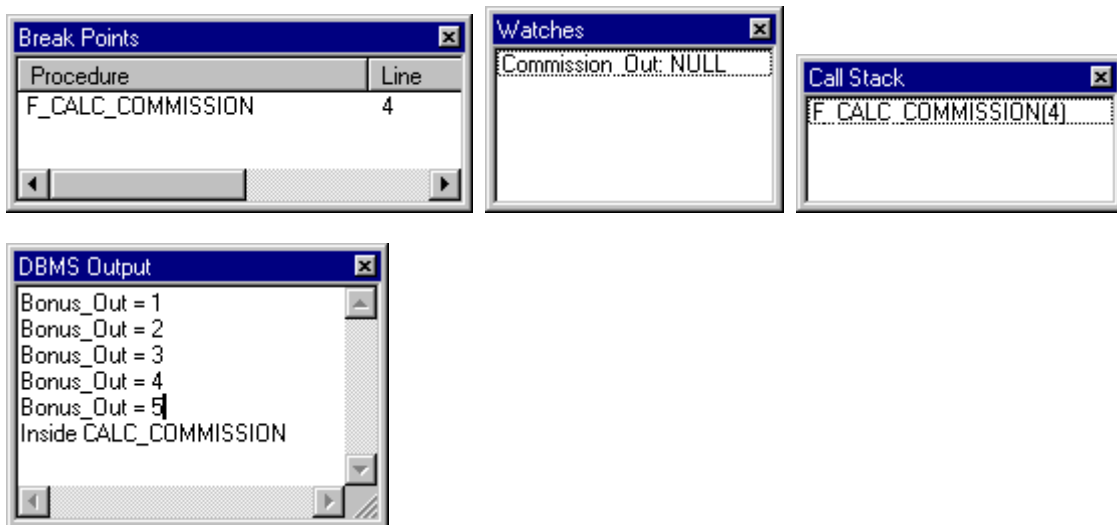


The functions are:

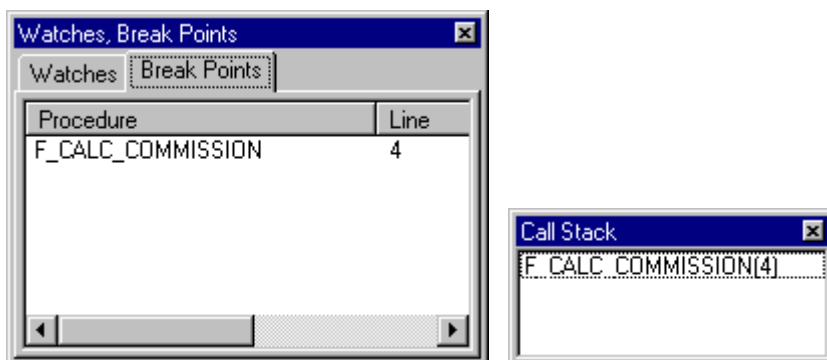
Execute Procedure without setting parameters
 Set Parameters
 Step Over
 Trace Into
 Trace Out
 Halt
 Add Watch
 Compile Dependent procedures with Debug Information

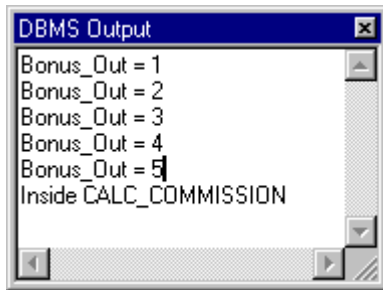
The Dockable Windows Feature

There are 4 stay-on-top windows for Breakpoints, Call Stack, Watches, and DBMS Output. Any of these 4 windows can be docked together into one window (or combinations of multiple docked windows) by dragging the window title bar of one window and dropping it on another window. This will create a tabbed interface to the separate panels.

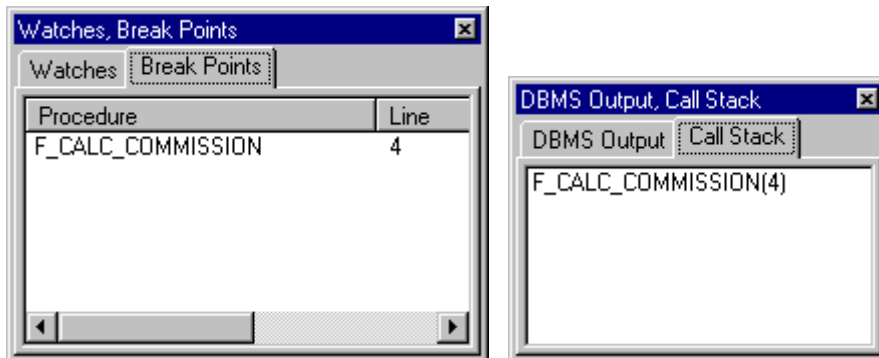


into this:

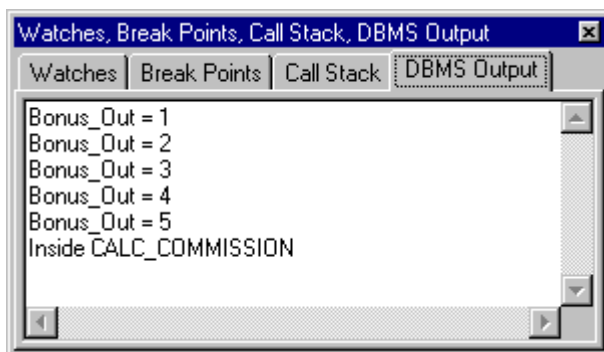




or perhaps this:



or combined all together into this:



There is an option in the TOAD Options dialog (see “View|Options”), on the “Debugging” tab, where you can force the opening of all four debug windows when any of them are opened. Check on “Automatically show all debugging windows when debugging”.

The Status Panel Indicators

While debugging PL/SQL code, the word “Running” will be displayed in a segment of the “Stored Procedure Edit/Compile” window status panel at the bottom of the window, or “Idle” if you are not currently debugging a procedure.



In order to finish running the debugger, click the “Continue execution to breakpoint” button in the toolbar, or select “Run (continue execution)” from the Debug menu.

Debug Functions

Starting With or Without Argument Values

To start a debug session with or without any argument values, select menu item “Debug|Execute Current Source”, or click the toolbar button:



or press the shortcut keys, <SHIFT>F9. Debugging will begin, and it will stop on the breakpoints as appropriate, or run to the end otherwise.

If your procedure contains any IN or IN/OUT argument values, and those argument values have not been set yet (via the “Execute Procedure” dialog), these will be set to NULL.

```
FUNCTION F_CALC_BONUS (Salary_In IN Number) RETURN NUMBER IS
Bonus_Out NUMBER;
BEGIN
  /* Set Bonus earned equal to 10 percent of the employee's salary. */
  Bonus_Out := Salary_In * 0.10;
  DBMS_OUTPUT.PUT_LINE ( 'Bonus_Out = ' || to_char(Bonus_Out));
  Return Bonus_Out;
END F_CALC_BONUS;
```

In this case, “Salary_In” would be set to NULL. This would not be very useful for functions that depend on the argument values, but is useful for checking branching logic, etc.

If you set any argument values in the “Execute Procedure” dialog (see below for details), then those values and settings will be used.

In the case of debugging a package, you must select which package procedure or package function to start debugging. This is selected in the “Execute Procedure” dialog. Once you have selected the package procedure or function to execute, the “Execute” toolbar button will be enabled.

If you are debugging a trigger, then you have to go through the “Execute Procedure” dialog in order to set up the anonymous PL/SQL block that will invoke the trigger.

Setting Argument Values

To set IN or IN/OUT argument values, select a Package Procedure to execute, or set up a trigger for debugging, select menu item “Debug|Set Parameters”, or click the toolbar button:

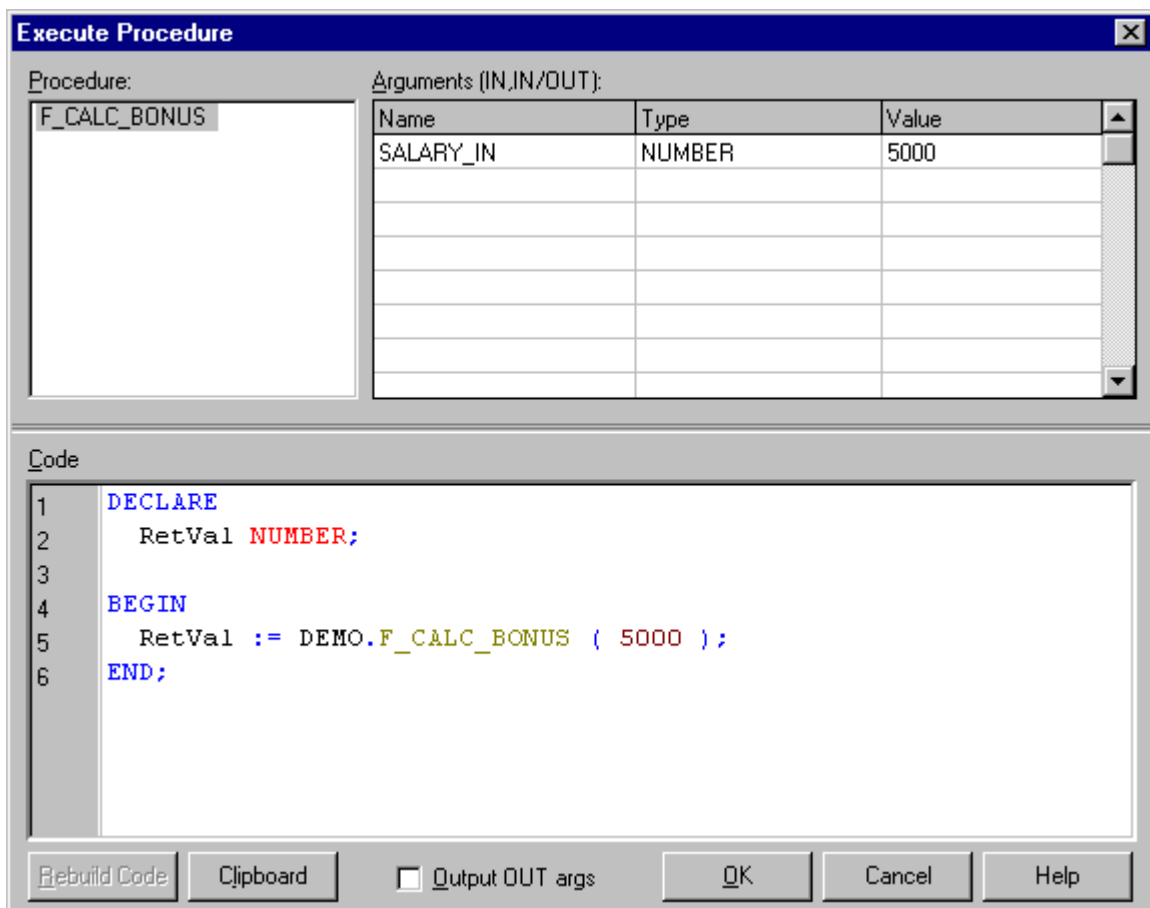


or press the shortcut keys, <CTRL>F9. This does NOT execute the procedure or start debugging.

There are different uses for the “Execute Procedure” dialog depending on the type of PL/SQL object to debug: Procedures, Functions, Package Procedures, Package Functions, or Triggers.

Execute Procedure or Function

You will be presented with this dialog to input values for the procedure or function arguments:



The dialog box is titled "Execute Procedure". It has a "Procedure:" field on the left containing "F_CALC_BONUS". To the right is a table for "Arguments (IN,IN/OUT):".

Name	Type	Value
SALARY_IN	NUMBER	5000

Below the table is a "Code" section with a text area containing the following PL/SQL code:

```
1 DECLARE
2   RetVal NUMBER;
3
4 BEGIN
5   RetVal := DEMO.F_CALC_BONUS ( 5000 );
6 END;
```

At the bottom are buttons: "Rebuild Code", "Clipboard", a checkbox for "Output OUT args" (unchecked), "OK", "Cancel", and "Help".

Enter the desired values in the “Value” column, and click the <OK> button when ready to invoke the procedure.

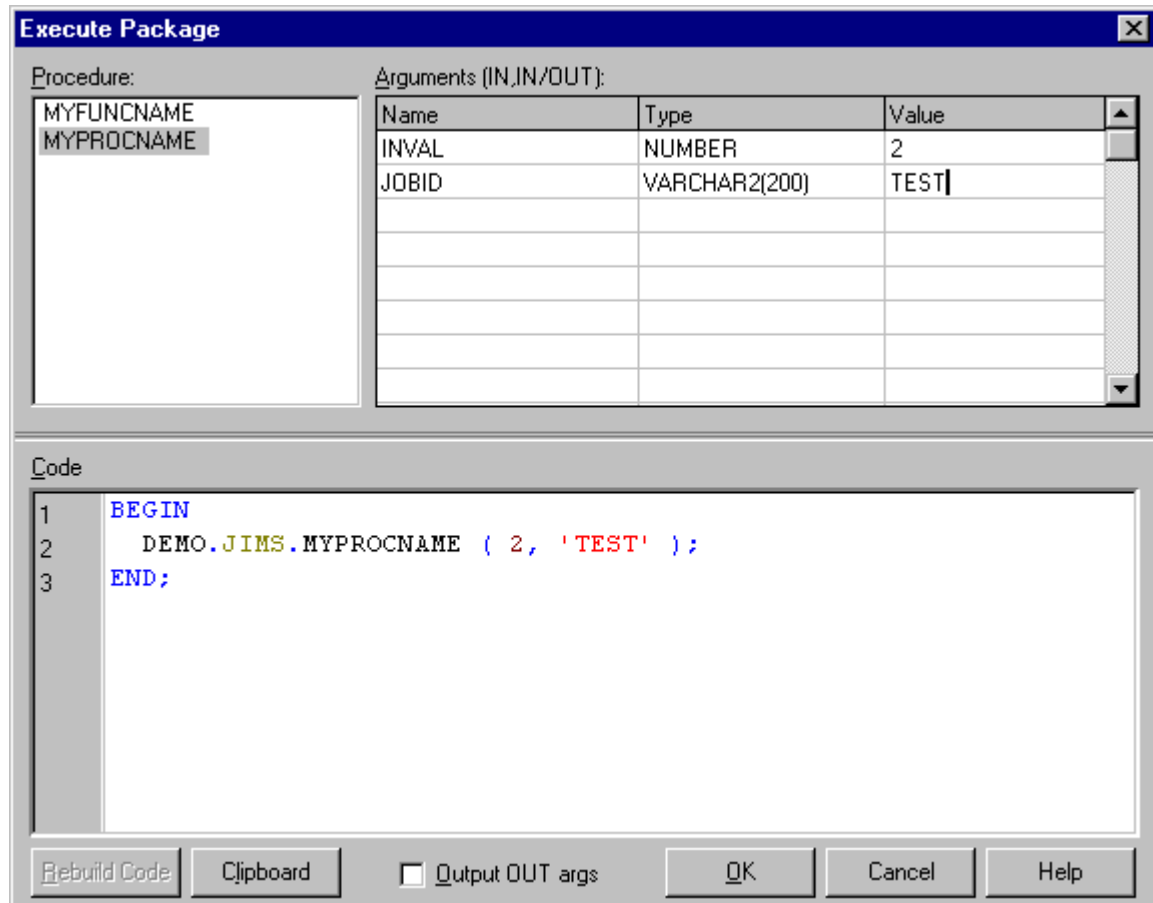
Notice how TOAD debugs the given PL/SQL procedure via an anonymous PL/SQL block. As you enter values, the anonymous PL/SQL block code will be updated automatically.

You can also directly edit the anonymous PL/SQL code block. If you want to resynchronize the anonymous PL/SQL block with the values entered in the grid, then click the <Rebuild Code> button. The

<Rebuild Code> button starts out disabled. If you make manual changes in the anonymous PL/SQL block, then the button becomes enabled.

Execute Package

In the case of debugging packages, a list will be presented for you to select which package procedure or package function to execute. Enter your values in the “Value” column, if desired.



The "Execute Package" dialog box is shown. It has a title bar with a close button. The main area is divided into two sections. The top section is labeled "Procedure:" and contains a list box with two items: "MYFUNCNAME" and "MYPROCNAME", with "MYPROCNAME" selected. To the right of this is a section labeled "Arguments (IN,IN/OUT):" which contains a table with four columns: "Name", "Type", and "Value". The table has three rows of data: "INVAL" with type "NUMBER" and value "2", "JOBID" with type "VARCHAR2(200)" and value "TEST", and an empty row. The bottom section is labeled "Code" and contains a text area with the following PL/SQL code:

```
1 BEGIN
2   DEMO.JIMS.MYPROCNAME ( 2, 'TEST' );
3 END;
```

At the bottom of the dialog are five buttons: "Rebuild Code", "Clipboard", a checkbox labeled "Output OUT args" which is unchecked, "OK", and "Cancel".

Name	Type	Value
INVAL	NUMBER	2
JOBID	VARCHAR2(200)	TEST

Execute Trigger: INSERT

Debugging triggers are a little different then debugging procedures or functions. The values entered are for the column values, not argument values. You must go through the “Execute Trigger” dialog to set up the proper anonymous PL/SQL block to invoke the trigger, at which point the “Execute” toolbar button will become enabled.

In the case of debugging an INSERT trigger, the values will be used as the values to insert. Note that the inserted record will be rolled back so that no changes are made to the database during debugging. The “INSERT INTO...” code is not valid until you enter the column values.

Execute Trigger

Trigger: MONEY_TEST_TRIG

Column Values: ☐ WHERE clause values

Name	Type	Value
ID	NUMBER	2
MONEY_VALUE	NUMBER	2

Code

```

1  -- Modify this anonymous block so that it will
2  -- cause the trigger to fire.
3  BEGIN
4      INSERT INTO DEMO.MONEY_TEST (ID, MONEY_VALUE) VALUES (2, 2);
5      ROLLBACK;
6  END;

```

Rebuild Code Clipboard ☐ Output OUT args OK Cancel Help

Execute Trigger: UPDATE

In the case of debugging an UPDATE trigger, you have to enter values for the “SET...” clause AND the “WHERE...” clause. With the “Where Clause Values” check box unchecked, enter the “SET...” values. With the “Where Clause Values” check box checked, enter the “WHERE...” values. Note that the updated record will be rolled back so that no changes are made to the database during debugging. The “UPDATE TABLE...” code is not valid until you enter the column values.

Entering the “SET...” values (“WHERE clause values” checkbox is unchecked):

Execute Trigger

Trigger: MONEY_TEST_TRIG_UPD...

Column Values: ☐ WHERE clause values

Name	Type	Value
ID	NUMBER	1
MONEY_VALUE	NUMBER	4

Code

```

1  -- Modify this anonymous block so that it will
2  -- cause the trigger to fire.
3  BEGIN
4      UPDATE DEMO.MONEY_TEST SET ID = 1, MONEY_VALUE = 2
5      WHERE ID = 3 and MONEY_VALUE = 4;
6      ROLLBACK;
7  END;

```

Rebuild Code Clipboard ☐ Output OUT args OK Cancel Help

Entering the “WHERE...” values (“WHERE clause values” checkbox is checked):

Execute Trigger

Trigger: MONEY_TEST_TRIG_UPD...

Column Values: ☒ WHERE clause values

Name	Type	Value
ID	NUMBER	3
MONEY_VALUE	NUMBER	4

Code

```

1  -- Modify this anonymous block so that it will
2  -- cause the trigger to fire.
3  BEGIN
4      UPDATE DEMO.MONEY_TEST SET ID = 1, MONEY_VALUE = 2
5      WHERE ID = 3 and MONEY_VALUE = 4;
6      ROLLBACK;
7  END;

```

Rebuild Code Clipboard ☐ Output OUT args OK Cancel Help

Execute Trigger: DELETE

In the case of debugging a DELETE trigger, you have to enter values for the “WHERE...” clause. With the “Where Clause Values” check box checked, enter the “WHERE...” values. Note that the deleted record will be rolled back so that no changes are made to the database during debugging. The “DELETE FROM...” code is not valid until you enter the column values.

Execute Trigger

Trigger: MONEY_TEST_TRIG_DELE...

Column Values: ☒ WHERE clause values

Name	Type	Value
ID	NUMBER	1
MONEY_VALUE	NUMBER	2

Code

```

1  -- Modify this anonymous block so that it will
2  -- cause the trigger to fire.
3  BEGIN
4      DELETE FROM DEMO.MONEY_TEST WHERE ID = 1 and MONEY_VALUE = 2;
5      ROLLBACK;
6  END;

```

Rebuild Code Clipboard ☐ Output OUT args OK Cancel Help

NOTE: Once a value is entered into the Value column of the grid, in order to make it “NULL” again, type in the word “NULL”. Otherwise, the value will be the empty string, ‘’.

NOTE: In the case of multiple BEFORE or AFTER actions, INSERT takes priority over UPDATE, and UPDATE takes priority over DELETE.

Viewing OUT Argument Values

If you have OUT or IN OUT arguments in your procedure, you can elect to view their values during debugging in the Debug DBMS Output window. To accomplish this, check the “Output OUT Args” check box on the “Execute Procedure/Package/Trigger” dialog. TOAD will automatically add DBMS_OUTPUT.Put_Line statements at the end of the anonymous PL/SQL block used to invoke your procedure. Turn on the Debug DBMS Output window from the “Debug” menu, or press <CTRL><ALT>D.

Procedure To Debug:

```

CREATE OR REPLACE PROCEDURE IN_OUT_ARG_TEST
    (in_arg      IN      number,
     out_arg     OUT     number,
     in_out_arg  IN OUT  number) IS
tmpVar NUMBER;
BEGIN
    tmpVar := 0;

```

```
END IN_OUT_ARG_TEST;  
/
```

Resulting Anonymous PL/SQL Block to Invoke the Procedure:

```
DECLARE  
    OUT_ARG NUMBER;  
    IN_OUT_ARG NUMBER;  
  
BEGIN  
    OUT_ARG := NULL;  
    IN_OUT_ARG := 2;  
  
    DEMO.IN_OUT_ARG_TEST ( 1, OUT_ARG, IN_OUT_ARG );  
  
    DBMS_OUTPUT.Put_Line('OUT_ARG = ' || TO_CHAR(OUT_ARG));  
    DBMS_OUTPUT.Put_Line('IN_OUT_ARG = ' || TO_CHAR(IN_OUT_ARG));  
  
END;
```

Dockable Windows

Breakpoints

Setting or Adding Breakpoints

You can single click in the Procedure Editor gutter to set or reset a breakpoint, which will be indicated by a “Stop” sign in the gutter. It is recommended that you set your gutter width to 35. See “Edit|Editor Options” menu dialog to change the gutter width.

Also, pressing the F5 key will set or reset a breakpoint on the current line in the editor.

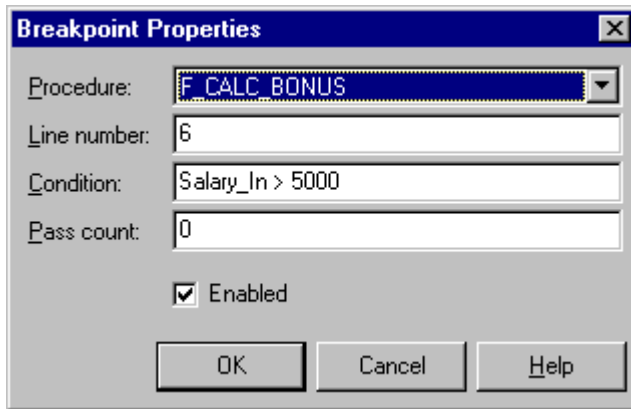
From within the Breakpoints window, you can also right click and select the “Add Breakpoint” menu item to add a breakpoint, or press <CTRL>A.

Editing a Breakpoint

You can change an existing breakpoint by double-clicking the breakpoint in the Breakpoints window, or single click to select it, click the right mouse button to present the menu, and select the “Edit Breakpoint” menu item, or press <CTRL>E. This will bring up the “Breakpoint Properties” dialog.

Conditional Breakpoints

You can set breakpoints that **ONLY** break if a certain condition is met. Select a breakpoint in the Breakpoints window, right click, Edit Breakpoint. Enter the condition for the breakpoint, e.g, “salary_in > 5000”. When running, the debugger will stop on the breakpoint **ONLY** if the condition is met.



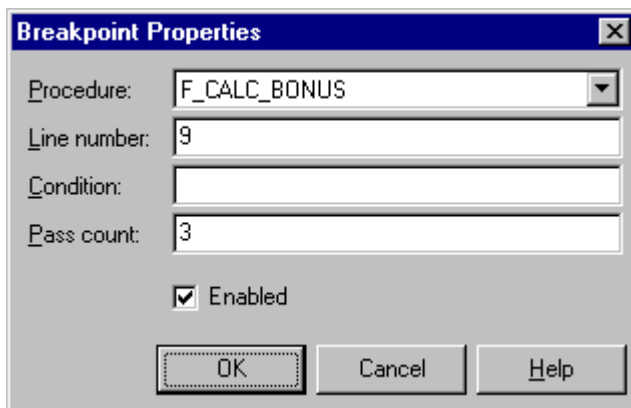
The image shows a 'Breakpoint Properties' dialog box. It has a title bar with a close button. Inside, there are four input fields: 'Procedure:' with a dropdown menu showing 'F_CALC_BONUS', 'Line number:' with the value '6', 'Condition:' with the text 'Salary_In > 5000', and 'Pass count:' with the value '0'. Below these fields is a checked checkbox labeled 'Enabled'. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

The format for “Condition” is Variable Operator Value. The currently supported operators are:

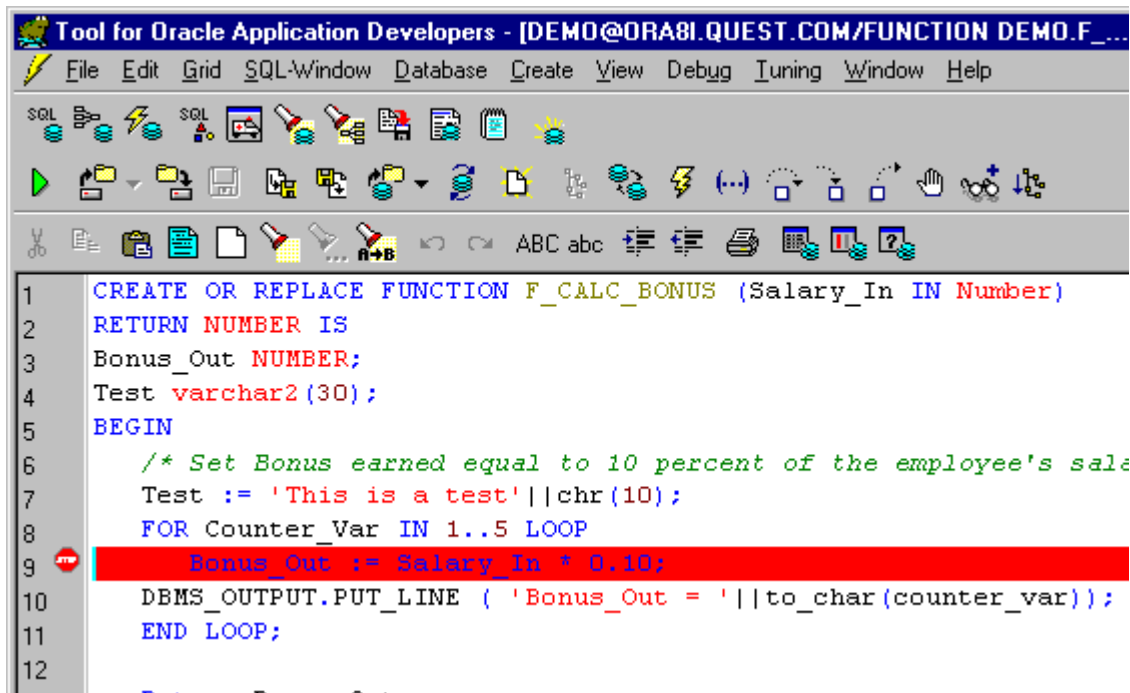
- <= Less then or equal to
- <> Does not equal
- >= Greater then or equal to
- < Less then
- > Greater then
- = Equal

Pass Count Breakpoints

You can set breakpoints that break **ONLY** after a certain number of passes in a loop have occurred.



The image shows a 'Breakpoint Properties' dialog box. It has a title bar with a close button. Inside, there are four input fields: 'Procedure:' with a dropdown menu showing 'F_CALC_BONUS', 'Line number:' with the value '9', 'Condition:' which is empty, and 'Pass count:' with the value '3'. Below these fields is a checked checkbox labeled 'Enabled'. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.



In this case, the debugger will **NOT** stop on line 9 of the code until just before “Bonus_Out := ...” executes for the third time. The pass could be a FOR loop, DO WHILE loop, IF/END IF, etc. It is not dependent on the COUNTER_VAR in this example.

Conditional, Pass Count Breakpoints

If both Condition and Pass Count are specified, the break will **ONLY** occur the nth time the condition is met.

Enabling or Disabling a Breakpoint

Once a breakpoint is set, you can temporarily disable it by double-clicking the breakpoint and unchecking the “Enabled” check box, or select the breakpoint, click the right mouse button, and select the “Disable Breakpoint” menu item.

Disabled breakpoints will be grayed out in the Breakpoints window.

Deleting a Breakpoint

You can delete a breakpoint by selecting it in the Breakpoints window and pressing the <Delete> key, or mouse right click and select “Delete Breakpoint” from the menu, or press <CTRL>D. Alternatively, you can press F5 in the procedure editor on the line containing a breakpoint to toggle it.

Breakpoints Right Mouse Menu

From the Breakpoints window, there is a right mouse menu for working with breakpoints:

E ^{dit} Breakpoint...	Ctrl+E
A ^{dd} Breakpoint...	Ctrl+A
E ⁿ able Breakpoint	
D ⁱ sable Breakpoint	
D ^e lete B ^r eakpoint	Ctrl+D
V ⁱ ew Source	
Enable All Breakpoints	
Disable All Breakpoints	
Delete All Breakpoints	
✓ Stay on T ^{op}	
✓ D ^{ock} able	

Watches

Adding a Watch

You can double-click to select a variable in the editor, click the “Add Watch” button in the toolbar, and the variable will be added to the list of watches.

Also, from the editor, you can click the right mouse button and select the “Debug|Add Watch at Cursor” menu item to add a Watch.

Also, from the Watches window, you can click the right mouse button and select the “Add Watch” menu item, or press <CTRL>A to add a watch.

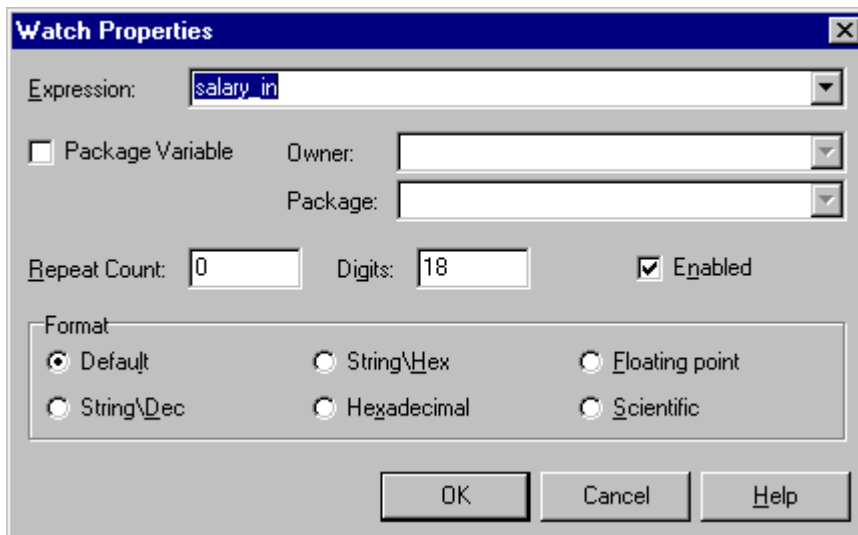
NOTE: you cannot watch a trigger :new.column or :old.column value. The Oracle Probe API does not support it.

NOTE: because of limitations in the Oracle Probe API, you cannot watch implicitly defined variables. For example, this code is correct, but you cannot watch the Counter_Var variable as it loops. A workaround to this would be to explicitly declare a local variable and copy the contents as it is changed, then add the watch to the local variable.

```
CREATE OR REPLACE FUNCTION F_CALC_BONUS (Salary_In Number) RETURN
NUMBER IS
Bonus_Out NUMBER;
BEGIN
    /* Set Bonus earned equal to 10 percent of the employee's salary. */
    for counter_var in 1..5 loop
        Bonus_Out := Salary_In * 0.10;
    end loop;
    Return Bonus_Out;
END F_CALC_BONUS;
/
```

Editing a Watch

You can change an existing watch by double-clicking the watch in the Watches window, or single click to select it, click the right mouse button to present the menu, and select the “Edit Watch” menu item, or press <CTRL>E. This will bring up the “Watch Properties” dialog.



If the variable you want to watch is a package variable, check the “Package Variable” checkbox, and select the owner and package name. Otherwise, a watch variable is assumed to be within the current scope of the package procedure or package function.

In addition to the usual data types that you can watch, e.g., date, number, varchar2, you can also watch array values and record types. If you have an array, e.g., MyArray(1..10), and set up a watch on MyArray(1), then you could also set a “Repeat Count” setting of 3 to examine MyArray(1), MyArray(2), and MyArray(3) all at the same time.

Digits is for the number of significant digits to be displayed.

If you prefer to see the watch value formatted differently than the default, then select from the format options, e.g., floating point, scientific, etc. Non-printable characters (ASCII 0-31) embedded in strings can often cause confusing errors and are hard to debug because most fonts are unable to render them in a meaningful way. “String\Dec” will display non-printable characters, e.g., CR and LF, in decimal format, e.g., “This is a test.\013\010” “String\Hex” will display those non-printable characters in hexadecimal format, e.g., “This is a test.\\$D\\$A”.

Enabling or Disabling a Watch

Once a watch is set, you can temporarily disable it by double-clicking the watch and unchecking the “Enabled” check box, or select the watch, click the right mouse button, and select the “Disable Watch” menu item.

Disabled watches will be grayed out in the Watches window, and marked with “<disabled>”.

You may want to disable some watches to improve the performance of the debugger. As each line of code is executed, each watch has to be evaluated. The fewer the watches to evaluate, the faster it will run.

Deleting a Watch

You can delete a watch by selecting it in the Watches window and press the <Delete> key, or <CTRL>D.

Watches Right Mouse Menu

From the Watches window, there is a right mouse button menu for Watch specific commands:



Call Stack

The Call Stack window displays the chain of functions and procedures as they are called. The most recent functions or procedures are listed on the top, least recent on the bottom. At the end of each procedure name is the current line number in that procedure. So, if procedure A called procedure B in line 5, then the call stack would look like this: "Procedure B(1)" followed by "Procedure A(5)".

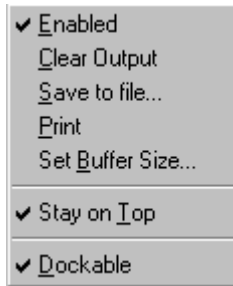
You can navigate among multiple procedures via the Call Stack window either by double-clicking the procedure name in the Call Stack window, or by selecting the procedure, click the right mouse button, and select the "View Source" menu item.



DBMS Output

The DBMS Output window displays the results of "DBMS_OUTPUT.PUT_LINE()" statements in the editor. Note that output only comes out after the procedure has completed execution, not while you are single stepping through the code. In the case of nested procedure calls, all procedures must have run to completion before any DBMS Output content is displayed.

There is a right mouse button menu for DBMS Output specific commands:

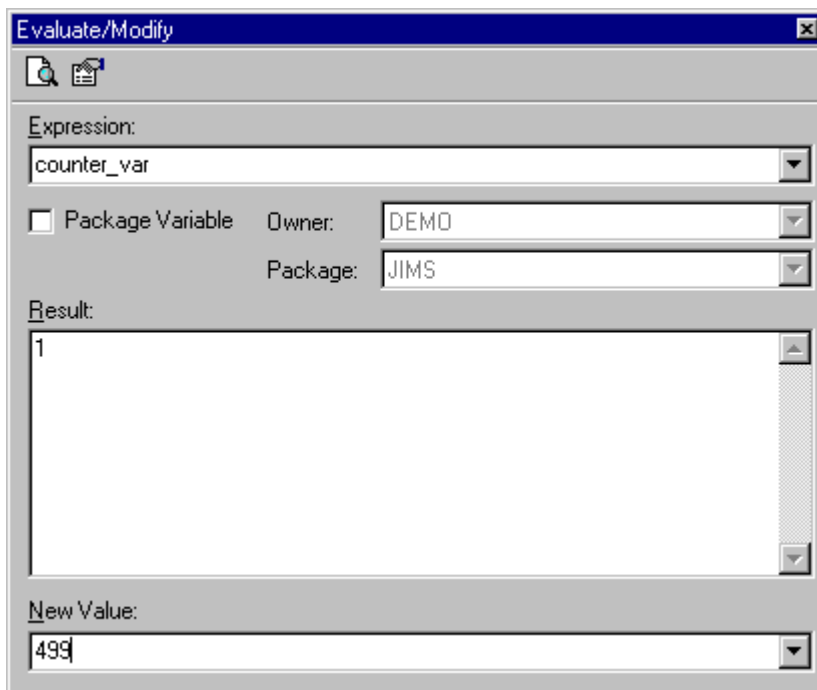


Buffer Size defaults to 20,000 bytes. The maximum size is 1,000,000 bytes.

You can edit the DBMS Output content, if you wish, to make comments, delete specific lines of output, etc. Also, the standard copy, cut, and paste keys work in the DBMS Output text box.

Evaluate/Modify

The Evaluate/Modify window allows you to view the value of a variable on the fly, without having to set a watch. Also, it permits you to change the value of a variable and continue execution. This is useful for advancing a loop variable to the end of a “FOR COUNTER_VAR IN 1..500 LOOP” loop. In this case, evaluate Counter_Var, and set its new value to 499. That will save you from having to step through the loop the extra 498 times.



Check the “Package Variable” checkbox if the variable to evaluate is a package level variable, and not a local variable.

The “Evaluate/Modify” window is not dockable with the rest of the debug windows.

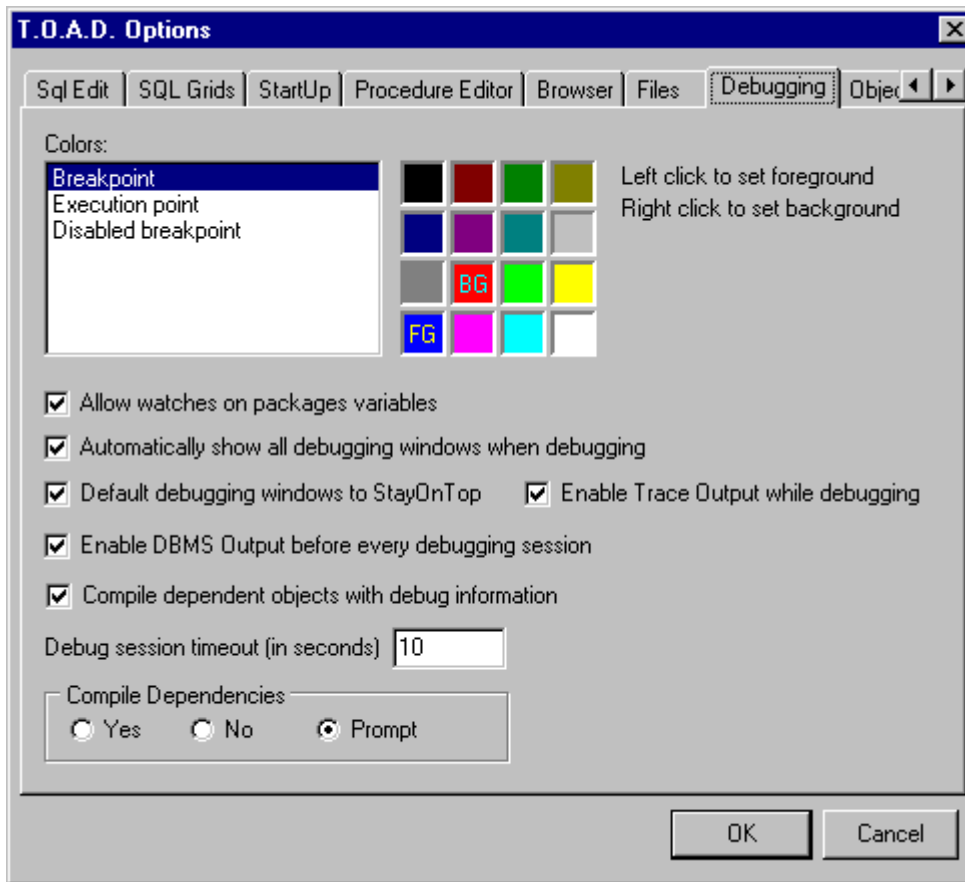
Keyboard Shortcuts

Here is a list of keyboard shortcuts for debugging purposes:

Keyboard Shortcut	Function
F5	Set or Delete a Breakpoint on the current line
F7	Trace Into
F8	Step Over
<Shift>F8	Trace Out
F9	Compile without Debug information
<Shift>F9	Execute Current Source
<Ctrl>F9	Set Parameters
F10	Display mouse right-click popup menu
F11	Run (continue execution)
F12	Run to Cursor
<Ctrl><Alt>B	Display Breakpoints
<Ctrl><Alt>D	Display DBMS Output
<Ctrl><Alt>E	Evaluate/Modify
<Ctrl><Alt>S	Display Call Stack
<Ctrl><Alt>W	Display Watches

General Debug Options

In the “View|TOAD Options” dialog, “Debugging” tab, there are several settings you can select.



To set the colors of a breakpoint, current execution point, and any disabled breakpoints, select the breakpoint type from the list at left. Then move the mouse pointer over the color selectors, click the left mouse button to select a foreground color (e.g., the code text color), and the right mouse button to select a background color. The letters “FG” will appear for Foreground color, and “BG” for Background color.

“Allow Watches on Package Variables” is provided because the Oracle Probe API call for watching package variables acts differently on Oracle 7 and Oracle 8 databases. On Oracle 7 databases, you have to step into the procedure **BEFORE** adding a watch on a package variable. On Oracle 8, you can set up the watch on the package variable before or after stepping into the procedure. If you do not want to inspect package variables, then uncheck this option.

“Automatically Show All Debugging Windows when Debugging” will bring up the docked window with all of the four windows: Breakpoints, Watches, Call Stack, and DBMS Output if any of the four are opened. With this option unchecked, then each window is activated separately, undocked.

“Default Debugging Windows to StayOnTop” will create the Breakpoints, Watches, Call Stack, or DBMS Output window as a Stay-On-Top window when activated. Otherwise, they will be hidden underneath TOAD when the Stored Procedure Editor window gets focus.

“Enable Trace Output while debugging” is for creating trace information while the debugger is running, which will help debug the debugger interactions with the database. This is normally unchecked and is for tech support or DBA use.

“Enable DBMS Output before every debugging session” will turn on DBMS Output before the debugging begins to capture any output from DBMS_OUTPUT statements. Otherwise, the DBMS Output is turned on

ONLY when you open the DBMS Output window, and will not display any previously executed output statements.

“Compile dependent objects with debug information” will cause debug symbol tables to be created for the procedure you are debugging, and all procedures called by your procedure. This will eliminate the “[No Debug Information]” messages when you step into other procedures.

“Debug session timeout (in seconds)” will limit the amount of time that the debugger will wait for the database to respond with debug information.

“Compile Dependencies Yes/No/Prompt” will conditionally compile procedures called by your procedure with debug information just before debugging begins.

These settings are saved in TOAD2.INI and restored the next time TOAD is invoked.

Preparing PL/SQL Code for Production

Once you have finished debugging your PL/SQL code, compile it one last time, which will recompile it without the debug symbol tables. This will make it smaller and faster to run.