

PLATINUM



SQL-Station Debugger

User Guide

Version 2.0

Title and Publication Number

PLATINUM Publication Number: SSD-O-200-UG00-00

Printed: January 20, 1997

Information in this guide is subject to change without notice and does not constitute a commitment on the part of PLATINUM *technology, inc.* It is supplied on an "as is" basis without any warranty of any kind, either explicit or implied. Information may be changed or updated in this guide at any time.

Copyright Information

SQL-Station Debugger is ©copyright 1/20/97 by PLATINUM *technology, inc.* and its subsidiaries. This guide is ©copyright 1/20/97 by PLATINUM *technology, inc.*, and its subsidiaries and may not be reproduced in whole or in part, by any means, without the written permission of PLATINUM *technology, inc.* and its subsidiaries.

Names marked TM or ® and other company and product names may be trademarks or registered trademarks of their respective vendors or organizations.

Mailing Address

PLATINUM *technology, inc.*
1815 South Meyers Road
Oakbrook Terrace, Illinois
60181-5235

Cover Photo

Photo by Daniel J. Cox/Tony Stone Images/Seattle

The animal featured on the cover of this guide is a symbol of PLATINUM's corporate commitment to preserving endangered and threatened species around the globe.

Table of Contents



Preface

Philosophy	ix
Contacting Technical Support	x
About This Guide	xi
Conventions	xi
Related Publications	xii

1 • Installation

System Requirements	1-2
ORACLE Requirements	1-2
Microsoft Windows Requirements	1-2
Installing SQL-Station Debugger	1-2

2 • Getting Started

SQL-Station Debugger Features	2-3
What It Means to Debug	2-4
Connecting to a Server	2-5
Creating a Connection	2-6
A Login Shortcut	2-7
Group Logins	2-8
Working in SQL-Station Debugger	2-9
Catalog Browser	2-10
Debugging	2-11
The Edit Window	2-13
Executing Server Objects	2-13

Ending a Session	2-15
Setting Preferences	2-15
Setting Defaults from the Edit Menu	2-15
Setting Overrides and Changing Defaults	2-16
Debugger Preferences	2-16
Edit Window Preferences	2-18
Oracle Options Preferences	2-20
Procedure Execution Preferences	2-22
General Preferences	2-22

3 • Tutorial

Introduction	3-2
Logging in	3-3
Exploring the Catalog Browser	3-3
Debugging a Procedure	3-5
Changing Values	3-8
Working with the Call Tree	3-9

4 • The Catalog Browser

Displaying the Catalog Browser	4-2
The Contents of the Catalog Browser	4-4
Catalog Browser Object Hierarchy	4-4
Object Types and Components	4-5
Function and Procedure Names	4-7
Debug Versions	4-8
Arranging the Catalog Browser Display	4-8
Filtering the Catalog Browser Display	4-9
Executing Server Objects	4-10
Entering Input Parameters	4-10
Executing and Committing	4-14

Displaying Tables	4-15
Displaying Table Contents in Grid Format	4-15
Displaying Table Structure	4-15
Displaying Object Code in an Edit Window	4-16
Dropping Objects	4-16

5 • Using the Debugger

Using the Debugger	5-3
Beginning the Debug Process	5-3
To Debug a Server Object	5-3
To Debug a SQL Text File	5-4
The Debugging Environment	5-4
Debugger Settings	5-5
Displaying Packages	5-6
Steps in Debugging	5-7
Excluding Modules	5-10
Working with Breakpoints	5-11
Setting Breakpoints	5-12
To Disable a Breakpoint	5-13
To Reactivate a Breakpoint	5-13
To Delete a Breakpoint	5-14
Cycling Breakpoint States	5-14
Starting the Debug Session	5-14
Entering Parameters	5-15
Using the Watch Pane	5-17
Adding a Variable	5-17
Evaluating Expressions	5-18
Changing Variable Values	5-18
Deleting Lines	5-19
Watching the Flow	5-20
Continue to Next Breakpoint	5-20

Step Into	5-20
Step Over	5-21
Step Out Of	5-22
Ending the Debugging Session	5-22

6 • Using the Edit Window

The Edit Window	6-3
Displaying an Edit Window	6-3
Using Multiple Windows	6-4
Edit Window Preferences	6-5
Setting the Result Display	6-6
Command Echoing	6-7
Executing SQL Code	6-7
Commit Options	6-8
Halting Execution	6-9
Display Options	6-9
Show Plan	6-9
Displaying DBMS Output	6-9
Editing Query Code	6-10
To Find and Replace Text	6-11
To Replace Text	6-11
Saving Text	6-12
Spooling Sessions to a Log File	6-13
To Stop Spooling	6-14

7 • Object Maintenance

About Debug Versions	7-2
Debug Version Preferences	7-2
Deleting Debug Versions	7-3
Trigger Maintenance	7-4

Debug Naming Conventions	7-5
Debug Object Status	7-6
Session Maintenance	7-7

8 • Tips and Troubleshooting

Background Information	8-2
Tips and Techniques	8-2
Granting Privileges on Debug Versions	8-2
Putting an Object on the Exclusion List	8-3
Debug Version Naming and Maintenance	8-3
Advantages of Debugging File Objects	8-3
Disadvantages of Debugging File Objects	8-4
Hung Sessions	8-4
Locking Issues	8-5
Trigger Issues	8-5
Overloaded Objects	8-6

Index



Preface

Philosophy

PLATINUM *technology, inc.*, is the leading vendor of open enterprise systems management (OESM) products, which help organizations manage all the hardware and software components of the multiplatform, multi-operating system, multivendor environment called the open enterprise environment (OEE).

By leveraging its expertise in relational technology, PLATINUM offers products and services that increase the efficiency of individual computing systems and databases, as well as the interoperability of these systems and databases in distributed environments.

Contacting Technical Support

You can contact us with any questions or problems you have. You will be directed to an experienced software engineer familiar with the product in question

For product assistance or information, contact:

USA or Canada, toll free	800-442-6861
Illinois	630-620-5000
FAX	630-691-0708 or 630-691-0406
Internet	info@platinum.com
World Wide Web	http://www.platinum.com

To send Email to PLATINUM Technical Support, use:

Internet	techsup@platinum.com
IBM MAIL Exchange	USRWNPSN

To contact PLATINUM Technical Support, use:

USA or Canada, toll free	800-833-PLAT (7528)
IBM Software Mail	PLATSM4
CompuServe	GO PLATINUM

Our Mailing Address is:

PLATINUM *technology, inc.*
1815 South Meyers Road
Oakbrook Terrace, IL 60181-5235

About This Guide

The *PLATINUM SQL-Station Debugger User Guide* explains how to use *PLATINUM SQL-Station Debugger* to its fullest capabilities.

This guide assumes that the appropriate *PLATINUM SQL-Station Debugger* components have been installed at your site. The instructions for installing the product are in the Installation Guide.

Chapter Number	Chapter name	Description
1	<i>Installation</i>	Introduces you to the functions and features of SQL-Station Debugger
2	<i>Getting Started</i>	The basics of SQL-Station Debugger
3	<i>Tutorial</i>	Guides you through some basic Debugging procedures.
4	<i>The Catalog Browser</i>	An introduction to the Catalog Browser
5	<i>Using the Debugger</i>	In-depth explanation of the Debugger
6	<i>Using the Edit Window</i>	Explores the functionality available in the Edit Window
7	<i>Object Maintenance</i>	Maintaining objects using the Debugger's functionality
8	<i>Tips and Troubleshooting</i>	Tips for working with the Debugger

Conventions

The following notational conventions are used throughout this manual:

- Each manual is divided into chapters. A chapter is a main division that describes a subject matter. Each chapter contains topics that are the major sections of the chapter. Each topic may contain subtopics that further break down the topic.
- The documentation's chapter number and page number appear on each page's footer. The page numbers restart with each chapter.
- The ■ symbol denotes a set of bullet points or instructions.
- NOTE text and WARNING text are designed for easy identification.

Related Publications

As you use this *PLATINUM SQL-Station Debugger User Guide*, you might find it helpful to have these additional books available for reference:

- PLATINUM SQL-Station Coder User Guide

Installation

This chapter lists the system requirements for running SQL-Station Debugger and details the installation process.

System Requirements	1-2
ORACLE Requirements	1-2
Microsoft Windows Requirements	1-2
Installing SQL-Station Debugger	1-2

System Requirements

This section lists software and hardware requirements for installing and running PLATINUM SQL-Station Debugger.

ORACLE Requirements

The following are the minimum requirements for your Oracle system:

- 32 bit SQL*Net 1.0 or greater.
- ORACLE 7.1 or greater.

Microsoft Windows Requirements

The following are the minimum requirements for your Windows system:

- Windows NT or Windows 95.
- 5 MB of client-side disk space.
- 300K of server-side Oracle table space.
- 8 MB of RAM.
- IBM PC, 486DX or higher.

Installing SQL-Station Debugger

Note • Installation of SQL-Station Debugger is part of an installation process that includes the entire SQL-Station product suite. This section covers only the portion of the installation that pertains to SQL-Station Debugger. See the installation chapter in the *SQL-Station Coder User Guide* and the *Plan Analyzer for Oracle Installation Guide* for detailed information on installing those products.

Installing SQL-Station Debugger

SQL-Station Debugger installation includes both client-side and server-side components. The server-side installation needs to be performed only once for each server and must be performed by the `SYS` account. If you choose server-side installation during the install process, SQL-Station Debugger first copies the client-side files and then presents a login dialog. Debugger installs the server files on the server designated in the connect string. If you login as a user other than `SYS`, the server-side install fails. You can perform the server-side install at any time after the client installation is complete by double-clicking the SQL-Station Debugger Server Install icon.

Note • We strongly recommend that you exit all programs other than Program Manager.

- 1 Insert Disk 1 in your 3.5" drive, or insert the CD-ROM into your CD-ROM drive, and run `SETUP.EXE`. After a few seconds, you see a Welcome screen. Read its contents and choose **Next**.
- 2 In the User Information window, enter your name, the company name, and choose **Next**.



Install then displays the component selection dialog.

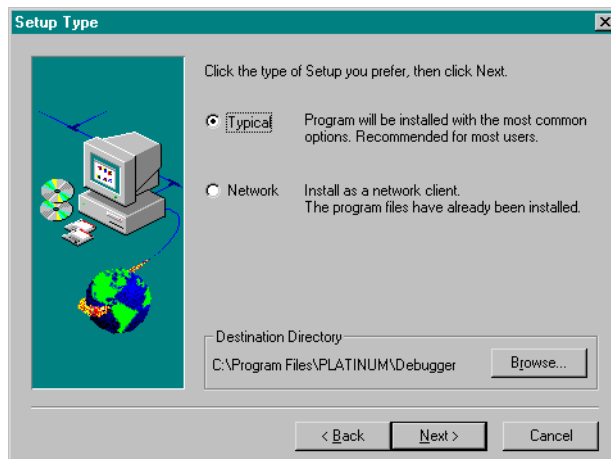
Installing SQL-Station Debugger

- 3 Install prompts you to choose which SQL-Station components you want to install. SQL-Station Coder is a prerequisite for installing Debugger. For this release, you must choose to install SQL-Station Coder and SQL-Station Debugger.

SQL-Station Coder will be the first product that installs in this process. For details on the installation of Coder, see the *SQL-Station Coder User Guide*.

Once Coder is installed, Install prepares for the installation of Debugger. After a brief pause, the Setup Type dialog is displayed.

- 4 Install prompts you to select either **Typical** or **Network** installation. Choose **Typical** for a standard client-side installation. Choose **Network** if you want the ability to invoke the client-side software via a network.

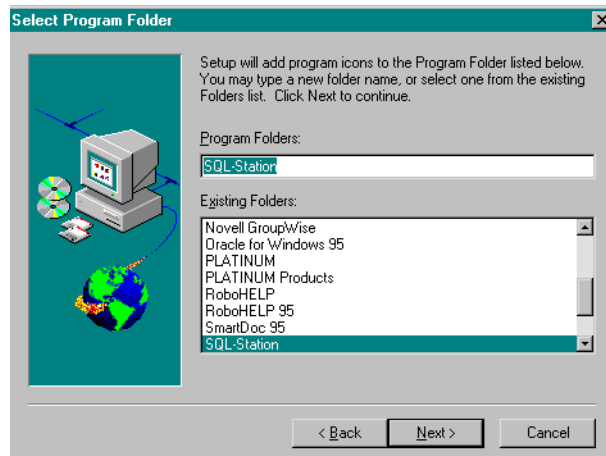


If the Destination Directory is incorrect, click **Browse** and select the proper destination directory.

Click **Next** and Install displays the Select Program Folder dialog.

- 5 Select the Program Folder where you want Install to place icons for the program.

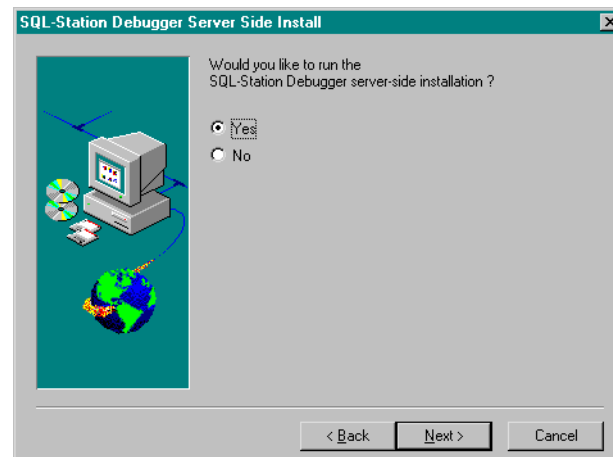
Installing SQL-Station Debugger



Type in a new folder name, select one from the list, or click **Next** if the default is satisfactory.

Install displays the Server-Side Installation dialog.

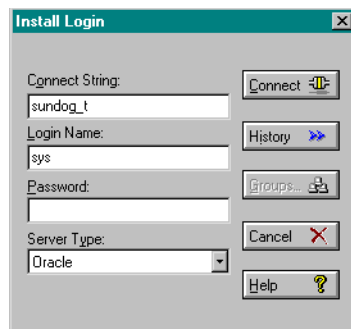
- 6 Install prompts you whether or not to run the server-side install in addition to the client-side install.



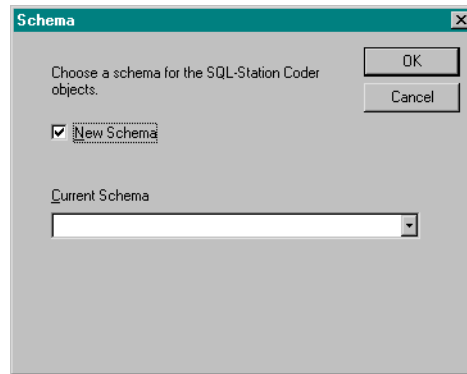
If you choose **No** to a server-side install, click **Next** and continue to step 7.

Note • In order to perform the server-side portion of the install, you must be able to login as SYS.

- 7 If you select **Yes** and then choose **Next**, you are asked if you wish to install the tutorial. The tutorial occupies less than 80K on the server. Select **Yes** or **No** for the tutorial install and then choose **Next**.
- 8 Install prompts you to review the installation information before copying the files. If the information is correct, choose **Next** and when prompted, insert additional disks.
- 9 **Client-side install only:** If you are requesting a client-side install only, Install copies all the files for the client-side portion of Debugger to the designated location. When copying is complete, it posts the Installation Complete window which prompts you to read the Readme file. If you choose **OK**, the Readme file displays in a Notepad window.
- 10 **Server-side install:** If you are requesting a server-side install, Install first copies the client-side files and then posts a login dialog. Enter a connect string to the server where you want the server files to be placed. The **Login Name** must be **SYS**. Supply the appropriate password, and click **Connect** to start the server installation process.

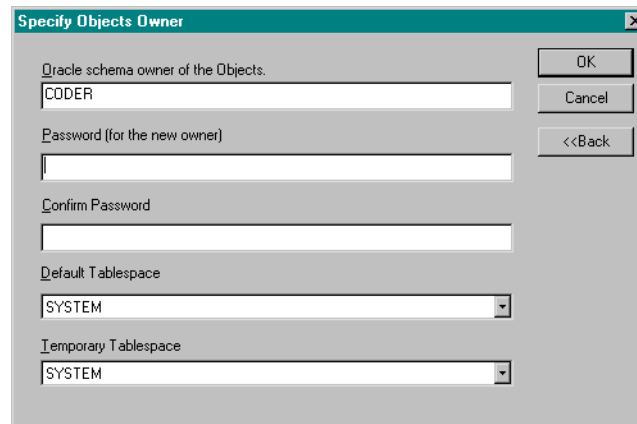


Install prompts you for information on the schema where the server objects will be placed.



Check New Schema to create a new schema for the server objects, or choose a currently existing schema name from the list box. Click OK when you've chosen.

- 11 Install displays the Specify Objects Owner dialog, requesting an account name and password.



The account name you supply will be the owner of all the common objects that Debugger needs to create debug versions of server objects. It will also own the tutorial objects if you chose to install the tutorial. Install creates the account and gives it SYS privileges. If you give an existing account name, the existing account is overwritten. The default owner name is `SQLDEBUG`.

Installing SQL-Station Debugger

- 12 Install copies the required files to the server. Any error messages appear above the gang thermometer in the progress dialog.
- 13 If this is the first time that Debugger has been installed on the client system, a dialog displays requesting the license number. Look at the sticker on your License Agreement sheet and enter the License Password.

If Debugger has been previously installed on the client system, Install does not post this dialog because it reads the Validation number from the `PLDBG.INI` file in the `WINNT35` or `WIN95` directory.

- 14 When installation is complete, Install displays the Installation Complete dialog.

Getting Started

This chapter discusses the role of debugging in the application development cycle and points out the benefits of server-side debugging. It describes the functionality of SQL-Station Debugger, charts the debugging process, and discusses setting preferences.

SQL-Station Debugger Features	2-3
What It Means to Debug	2-4
Connecting to a Server	2-5
Creating a Connection	2-6
A Login Shortcut	2-7
Group Logins	2-8
Working in SQL-Station Debugger	2-9
Catalog Browser	2-10
Debugging	2-11
The Edit Window	2-13
Executing Server Objects	2-13
Ending a Session	2-15
Setting Preferences	2-15
Setting Defaults from the Edit Menu	2-15

Setting Overrides and Changing Defaults	2-16
Debugger Preferences	2-16
Edit Window Preferences	2-18
Oracle Options Preferences	2-20
Procedure Execution Preferences	2-22
General Preferences	2-22

SQL-Station Debugger Features

PLATINUM SQL-Station Debugger provides a flexible and efficient environment for code testing, editing, and debugging. The three work environments facilitate a wide range of activities and provide the tools you need to work effectively. The following is a list of Debugger's highlights:

The Catalog Browser

- Displays a catalog of objects on all currently connected servers, arranged by owner or by object type.
- Shows all component modules of each object, including objects which reference it and objects which it references, and privileges on the object.
- Permits you to invoke debugging on any selected procedure, function, package procedure, package function, or trigger.
- Permits you to execute any selected procedure, function, package procedure, or package function.
- Displays the reverse-engineered code for any server object in a separate Edit window.
- Displays a table's contents, structure, and code.

The Debugger

- Includes a Call Tree showing the component modules of the selected object, a Source pane that displays the code for the selected node of the Call Tree, a Variable Tree that displays the current variables, a field for entering expressions for evaluation, a Watch pane for viewing parameter values, and a pane for package headers.

What It Means to Debug

- Reverse-engineers PL/SQL database objects into debug versions that transparently contain the necessary debug information; the original objects are never altered.
- Tracks the execution sequence of a selected procedure, function, trigger, package function, or package procedure in precise detail through the use of breakpoints; includes the ability to step into, out of, or over any local module.
- Displays the values of variables and expressions in the Watch pane at each step of the debugging process and permits manipulation of them to test different scenarios.
- Permits evaluation of complex expressions.
- Executes on the server, minimizing network traffic and providing a real-world picture of what's happening as code executes.

The Edit Window

- Provides an environment for checking, editing, and executing code.
- Displays the results of each command in a separate Result pane.
- Provides comprehensive error messages to facilitate syntax checking.
- Permits checking and editing of code for all server objects displayed in the catalog and for any SQL text file in the system.

What It Means to Debug

Debugging is a crucial phase in the development cycle of an application. In the initial phase, you specify the desired behavior of the function, procedure, package, or trigger. You then write code and test it until it executes. At that point, you've got functioning logic. But is it the *right* logic? Does its behavior match that described in the original specification? How do you locate the precise spot where it goes wrong, identify the nature of the problem, identify a solution, and implement it?

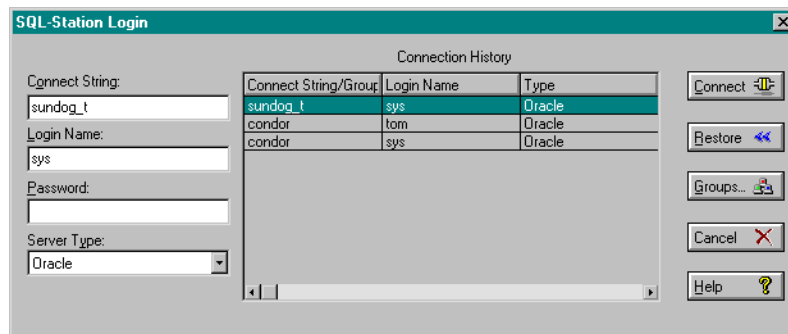
This is the point in the development cycle where a good debugging tool can ease your task as a developer, speed up the development cycle, and increase the reliability of the finished product. SQL-Station Debugger is a graphical debugging environment that lets you:

- Examine and compare the structure of database objects.
- Examine and modify SQL code for any object.
- Set breakpoints.
- See the flow of execution.
- See changes in the values of variables as you move through the code.
- Step into, over, or out of any program module.
- Test different scenarios by entering different parameter and variable values.
- See results and outputs.

The following sections are a quick outline of typical work flow in SQL-Station Debugger. Later chapters provide details.

Connecting to a Server

In general, the first step in using SQL-Station Debugger is to connect to a server that contains the database objects you are debugging.



Connect String

Enter either a SQL*NET I, SQL*NET II, or Personal Oracle connect string. If you are using SQL NET II, you can also enter a TNS alias connect string to a server containing a database. If your network supports it, you can also enter an alias.

Login Name

Enter the user account name that you wish to use for the session.

Password


Enter the password associated with the user account.

Server Type

This version supports only Oracle servers.

Creating a Connection

- 1 Select the SQL-Station icon from your Start menu. You will access the Debugger from within SQL-Station. SQL-Station opens and displays the Login dialog.

If SQL-Station is currently running, click the Connect  toolbar button or choose **Database, Open Connection** to display the Login dialog.

- 2 In the **Connect String** box, enter a string that connects to a server containing a database. Debugger also accepts aliases in place of actual connect strings.

Example: p:pendragon:orcl (string) or moondog (alias)

- 3 In the **Login Name** box, supply the user account for the session.

- 4 Enter the password for the specified user account in the **Password** box.
- 5 Choose the server type from the drop down list


Note • The first release of SQL-Station Debugger connects only to Oracle servers.

- 6 Click **Connect** to create the connection and close the login window.

A Login Shortcut

If you want to use a combination of account and connect string that you have used in the past, SQL-Station Debugger provides a shortcut for you:

- 1 Select the SQL-Station icon from the Start menu to start SQL-Station. You will access Debugger from within SQL-Station. SQL-Station opens and displays the Login dialog.

If SQL-Station is currently running, click the Connect  toolbar button or choose **Database, Open Connection** to display the Login dialog.

- 2 Check the **Connection History** spreadsheet for the login information you used previously.
- 3 Click on the row containing the connection that you want to re-establish. Debugger fills in the **Connect String**, **Login Name**, and **Server Type** fields for you.
- 4 Supply the correct password in the **Password** field, and click **Connect**.

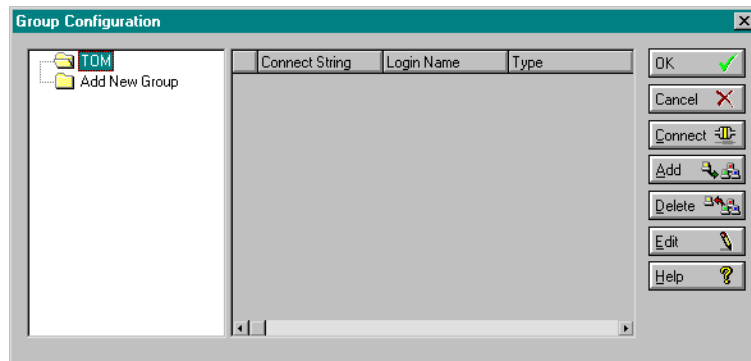
Note • SQL-Station Debugger allows concurrent sessions: one user can login to several servers and multiple users can login simultaneously to one server.

Group Logins

You can create a group of logins and then login to the whole group by entering the password for each group member when prompted.

Creating a New Login Group

- 1 In the Debugger Login dialog, click the **Groups**  button. The dialog changes to the Group Configuration dialog.



- 1 To create a new group, either double-click on the folder **Add New Group** or highlight **Add New Group** and click **Add**.
- 2 In the Add New Group dialog, type the name of the new group and click **OK**.

If an existing group is highlighted when you click **Add**, you can create a new group by choosing **Primary Group** in the Insert Element Into List dialog.

Adding Logins to a Group

To add individual logins to a specified group, do the following:

- 1 Highlight the group name and click **Add**.

- 2 Choose **Group Item** in the Insert Element Into List dialog and click **OK**.
- 3 In the New Item dialog, enter a connect string and a login name in the appropriate fields and click **OK**. You can add as many logins as you wish in this manner.

Subgroups

You can add subgroups to an existing group by doing the following:

- 1 Highlight the group name, click **Add**, choose **Group Item**, and click **OK**.
- 2 In the New Item dialog, enter a group name in the Connect String/Group Name field. Leave the Login Name field empty. Click **OK**.

SQL-Station Debugger adds the named group as a subgroup of the selected primary group. If the named group does not already exist, SQL-Station Debugger creates it as both a Primary Group and as a subgroup.

Logging In as a Group

To login as a group, specify the group name in the Connect String field of the Login dialog and click **Connect**. (If you are working in the Group Connect dialog, highlight the group name and click **Connect**.)

Debugger prompts you to enter the password for each login that is part of the group, including logins within associated subgroups. If you want to skip a particular login, press **Esc** or click **Cancel** when asked for the password.

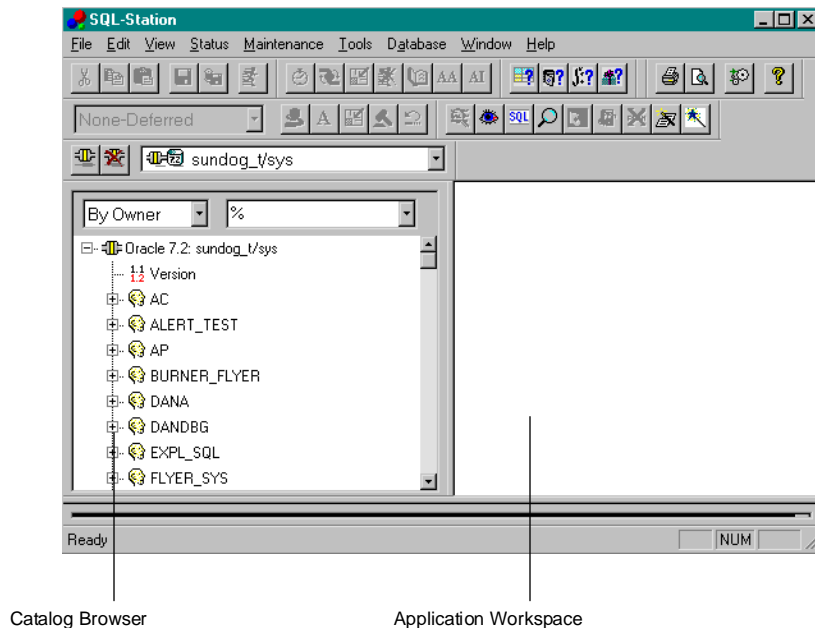
Working in SQL-Station Debugger

The following sections describe a typical sequence of activity in SQL-Station Debugger. See the *Tutorial* chapter for more detailed help in learning your way around Debugger.

Catalog Browser

This section describes a typical sequence within the Catalog Browser.

- 1 When you start Debugger, the Login dialog displays. Connect to one or more database servers. SQL-Station is opened with the Catalog Browser on the left and the Application Workspace on the right.




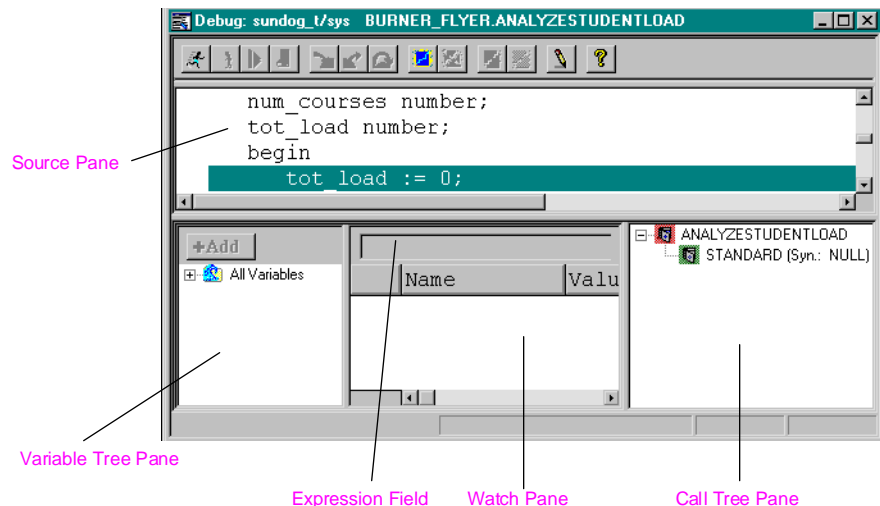
The Catalog Browser is your entry point to SQL-Station Debugger's functions. Here, you select the objects you want to debug or edit. You can have multiple instances of the Catalog Browser open. When you choose to debug an object, the Debug window will open in the Application Workspace area.

- 2 Choose from the list box in the Catalog Browser whether to display the objects grouped by **Owner** or by **Object Type**. Examine the list of available objects in the Catalog Browser. You can expand each object icon to see icons for its code, its component objects, the objects it references, and the objects that reference it. Each component in the catalog is further expandable as long as there is a plus symbol (+) to the left of its icon.

Debugging


This section describes a typical sequence within the Debugging Window.

- 1 To debug an object, highlight it and choose **Debug** from the right mouse-button menu or click the **Debug**  toolbar button. The Debugger displays in the Application Workspace with the object's code displayed in the Source pane. The Call Tree pane shows the component modules of the object.



- 2 Set breakpoints within the Source pane by double-clicking the appropriate lines of code. To set breakpoints in child modules, click the node in the Call Tree to display the code in the Source pane.

Note • Some server objects may be excluded from debugging. In this case, they do not appear on the Call Tree.

- 3 Choose **Debug, Go**, from the main menu, or click the Debug  toolbar button to start the debugging session.

The session runs to the first breakpoint. The variables that are in scope display in the Variable Tree and all local variables are automatically displayed in the Watch pane.


Each component in the Variable Tree is expandable as long as there is a plus symbol (+) to the left of its icon.

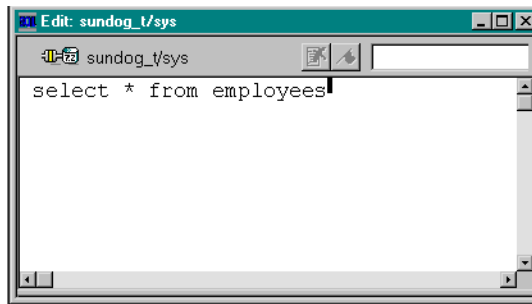
- 4 At this point, you can add additional variables to the Watch pane. Expand the Variable Tree so that the variable you are interested in is visible. Click on the variable and do one of the following:
 - Click the **Add** button directly above the Variable Tree.
 - Choose **Debug, Add Variable** from the menu bar.
 - Right-click the mouse and select **Add Variable**
- 5 Use the controls provided to execute the code a step at a time, run to the next breakpoint, or continue to the end. You can choose to step into or over any child module, and step out of a child module back to the calling module. You can also return to the beginning of the code and restart the debugging session.
- 6 You can watch the values of all variables that are in scope, and assign values to variables and input parameters to test different scenarios. SQL-Station Debugger even allows you to evaluate complex expressions at each step. Changes that you make to variables and expressions in the Debugger are runtime changes only. To make permanent changes to the object, modify and execute the code in the Edit window.
- 7 Use these debugging tools and options until you have determined the nature of the problem and identified a possible solution.

The Edit Window

This section describes a typical sequence within the Edit window.

- 1 If your debugging session suggests changes to the object's code, you will then want to work in the Edit window.

To display the object's code in the Edit window, click the **Edit Code**  toolbar button on the Debug window toolbar. SQL-Station Debugger reverse-engineers the object and displays the code in an Edit window. You can then edit and modify this code. Choosing **SQL, Execute** executes the SQL statements.




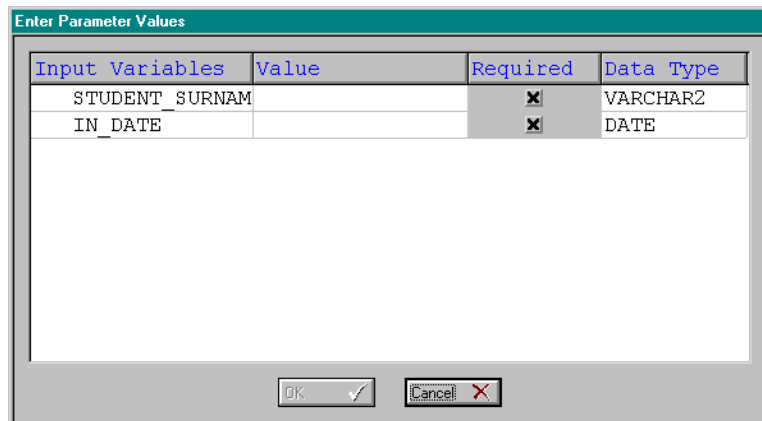
Because you can have multiple Edit windows open, you can compare code for different objects and you can copy and paste code within and between windows. You can display reverse-engineered code for compiled objects—described in the previous step—or display the contents of any text file containing SQL code.

Note • If multiple Edit windows are open, each one is numbered separately under **Window** in the **SQL-Station Debugger** menu bar.

Executing Server Objects

This section describes a typical sequence for executing server objects.

- 1 To execute an object, highlight the object's icon in the Catalog Browser and choose **Execute** from the right mouse-button menu or from the toolbar, or click the **Execute**  toolbar button.
- 2 If the object requires input parameters, SQL-Station Debugger displays the Enter Parameter Values window. Enter the required input values and click **OK**.



The dialog box titled "Enter Parameter Values" contains a table with four columns: "Input Variables", "Value", "Required", and "Data Type". There are two rows of data. The first row has "STUDENT_SURNAM" in the first column, an empty text box in the second, a required checkbox (checked) in the third, and "VARCHAR2" in the fourth. The second row has "IN_DATE" in the first column, an empty text box in the second, a required checkbox (checked) in the third, and "DATE" in the fourth. Below the table is a large empty text area. At the bottom are "OK" and "Cancel" buttons.

Input Variables	Value	Required	Data Type
STUDENT_SURNAM		<input checked="" type="checkbox"/>	VARCHAR2
IN_DATE		<input checked="" type="checkbox"/>	DATE

After execution, the output values appear in a separate Results window. If the code contained more than one command, the output from each command appears in a separate tab. If you select **Database, Options, DBMS Output** from the main menu, the contents of any DBMS_OUTPUT statements appear in the message pane at the bottom.

A Sample Scenario

A user in accounting notices that a certain combination of data causes a problem when he runs the hourly consolidation routines. He refers the problem to the DBA who passes it on to a programmer. Working on development versions of the objects, the programmer identifies the error by running the procedure in the debugger. He steps through individual lines of code and identifies the exact line where the calculated value


becomes incorrect. He then displays the code in the Edit window, modifies it, and redeploys the object. The DBA returns the object to the production server.

Ending a Session

SQL-Station Debugger allows concurrent sessions. You can terminate a particular session at any time without affecting other sessions.

- 1 In the Connection list box, choose the session that you want to disconnect



Click the Disconnect  toolbar button or choose **Database, Disconnect**.

SQL-Station Debugger terminates the selected session. All other sessions remain in their current state.

Setting Preferences

You can use the Preferences dialog to alter the behavior of the Debugger and the Edit window.

Setting Defaults from the Edit Menu

You can access the Preferences dialog from the **Edit** menu to set default behaviors.

- 1 Choose **Edit, Preferences** to display the Preferences dialog.
- 2 Choose a tab: Debugger, SQL Server Options, Catalog Browser, Procedure Execution, Edit Window, Oracle Options, or General.
- 3 Make the settings and click **Save As Default**.

Note • The new defaults take effect in any new Edit or Debugging windows. They do not affect windows that are open at the time you make the change.

The settings are described in *Debugger Preferences* and in *Edit Window Preferences*, below.

Setting Overrides and Changing Defaults

You can override the default settings for either the Debugger or the Edit window. An *override* is a change that affects only the current session. You can also choose to make these overrides be the new defaults.

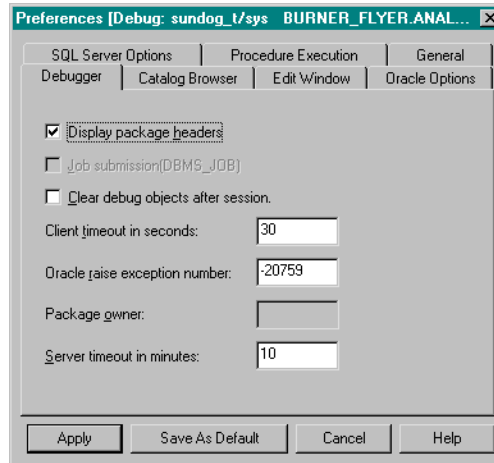
- 1 To override settings, choose **Edit, Preferences**.

Note • The **Debug** menu item is available only when a Debugger window is active. The **SQL** menu item is available only when an Edit window is active.

- 2 Choose the appropriate tab and make the desired settings. The settings are described in the following pages.
- 3 To apply the settings to only the current session, choose **OK**. To change the default settings, choose **Save As Default**.

Debugger Preferences

Use the Debugger tab of the Preferences dialog to customize the behavior of debugging sessions:

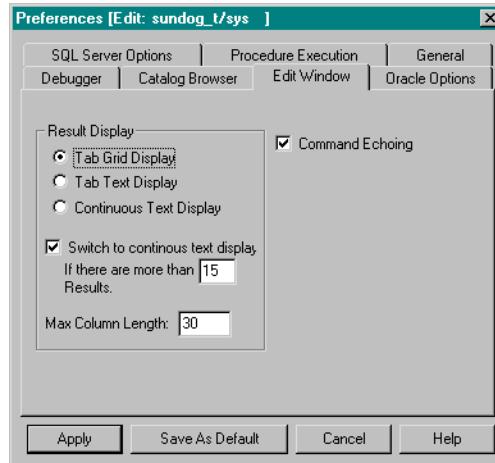


Field	Description
Display package Headers	When this preference is displayed in the Source pane, SQL-Station Debugger displays the package header in an additional pane at the top of the Debugger.
Job submission	When this is enabled the job is submitted to the database management system.
Clear debug objects after session	When this is enabled, SQL-Station Debugger deletes the debug versions of objects when the debugging session ends. If it is not enabled, the debug versions are saved and reused in future debugging sessions. SQL-Station Debugger checks to be sure they are current at the beginning of any debugging session where they are needed and deletes them if they are out of date.

Field	Description
Client timeout in Seconds	During a debugging session, SQL-Station Debugger waits for the specified number of seconds for some message to be returned from the procedure being debugged. If no message is returned, SQL-Station Debugger posts an error message.
Oracle raise exception number	By default, SQL-Station Debugger uses message number 20759 for the error message. If this number is already in use in your system, you can specify any other valid user-defined exception number.
Package owner	When you are connected to more than one server, you may need to specify the location of the SQL-Station Debugger schema
Server timeout in minutes	A procedure running on the server waits for a command from the Debugger for the amount of time specified in this field before terminating execution.

Edit Window Preferences

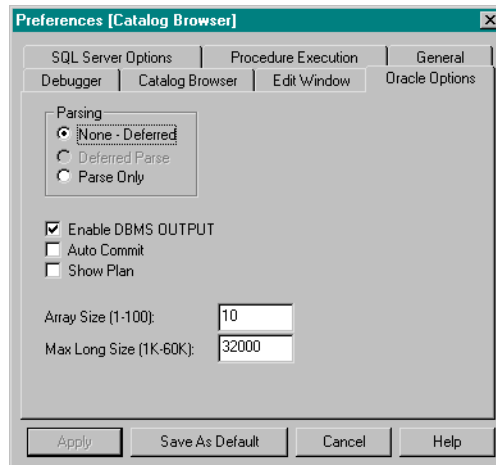
Use the SQL Window tab of the Preferences dialog to customize the behavior of SQL editing sessions:



Fields	Description
Tab Grid Display	Query results display in grid form if this option is checked. This is the default setting.
Tab Text Display	Query results display in text form.
Continuous Text Display	This displays the results in text without the table format.
Switch to continous text display if there are more than n results	This will cause the display to switch to continuous text if the number of results returned exceeds the number you specify in the field.
Max Column Length	This sets the maximum length of the columns returned.
Command Echoing	Displays the command associated with each result tab in a separate pane with time and session information.

Oracle Options Preferences

Choosing the Oracle Options tab of the Preferences dialog displays the following dialog:

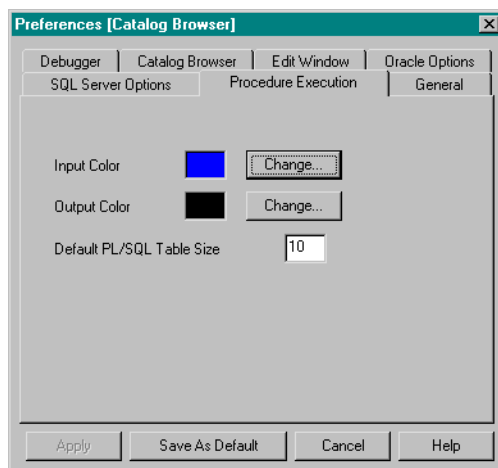


Field	Description
None-Deferred	The statement in the Edit Window will be parsed on-line in a separate step from the execution.
Deferred Parse	The statement is not parsed on-line, but the parse will be combined with the execution in order to enhance performance.
Parse Only	An on-line parse will be performed with no execution. However, parsing will execute all DDL statements immediately.
Enable DBMS OUTPUT	When enabled, includes the contents of any DBMS_OUTPUT commands in the execution results. (This option can also be set using the button in the Edit Window toolbar).

Field	Description
Auto Commit	SQL-Station Debugger issues a commit after each statement runs in an Edit Window. Rollbacks are not possible when this option is enabled. Auto Commit affects the SQL editing environment, not object execution from the Browser.
Show Plan	This displays the execution plan of a statement in the message pane of the Results window.
Array Size	The Edit Window takes advantage of Oracle's array binding to enhance performance. This sets the preferred array size.
Max Long Size	The Edit Window can select long data types up to 60K. This truncates this length to conserve memory and increase performance.

Procedure Execution Preferences

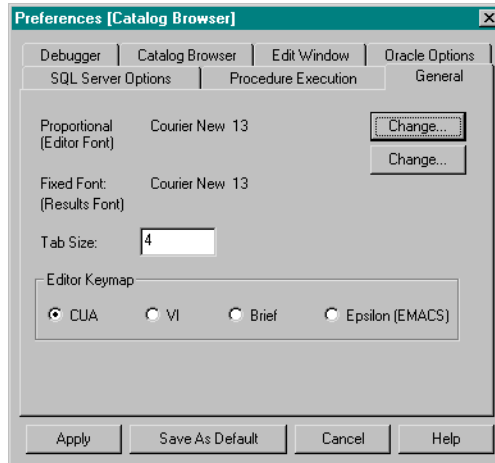
Use the Procedure Execution tab of the Preferences dialog to set input and output options when executing procedures and functions:



Field	Description
Input Color	Sets the color used for column headers and input parameter values in the Input parameter grid.
Output Color	Sets the color used for column headers and output variable values in the Results grid.
Default PL/SQL Table Size	For the database object execution tool: This sets the default size for tables that are input parameters or outputs for object execution.

General Preferences

Use the General tab of the Preferences dialog to set display characteristics:



Field	Description
Proportional (Editor Font)	Sets the font used in the Edit Window query pane. Clicking the top Change button brings up a font change dialog.
Fixed Font (Results Font)	Sets the font used in the Result text pane. Clicking the second Change button brings up a font change dialog.
Tab Size	Sets the size of the tab (in characters) in the Edit Window.
Editor Keymap	Sets the keymapping for the Edit Window to match one of the popular text-editing keymaps you choose.

Tutorial

This chapter takes you step-by-step through a number of basic SQL-Station Debugger activities. In it, you practice debugging a procedure, a function, and a trigger. In each, you set breakpoints and step through the object, stepping into and out of local modules.

Introduction	3-2
Logging in	3-3
Exploring the Catalog Browser	3-3
Debugging a Procedure	3-5
Changing Values	3-8
Working with the Call Tree	3-9

Introduction

In this tutorial, you learn a number of techniques for debugging server objects and file objects, as well as some basic maintenance activities.

In order to perform the steps in this chapter, the following must be installed:

- The tutorial objects must be installed on a server that you can connect to.
- You must have the client-side portion of SQL-Station Debugger installed on your system.

The server-side installation of SQL-Station Debugger includes an option for installing the tutorial objects on the server. If the administrator did not choose this option when installing the rest of SQL-Station Debugger, the tutorial can be installed later by selecting the SQL-Station Debugger Tutorial Install from your Start menu or running the `DBGTUTOR.SQL` file.

To check whether the tutorial has been installed on a server, look for a procedure named `AnalyzeStudentLoad` and a function named `CheckGenderEquality`. The complete list of tutorial objects is:

- **3 tables:** `ptsql_students`, `ptsql_courses`, `ptsql_student_registration`.
- **1 package:** `StudentStatistics`.
- **2 stand-alone procedures:** `AnalyzeStudentLoad`, `CourseGenderDistribution`
- **1 stand-alone function:** `CheckGenderEquality`.
- **1 trigger:** `after_reg_insert`.

Logging in

Prior to beginning these steps, you need to know a connect string to a server that contains the tutorial objects. You also need to know the account name that owns these objects and the password for that account.

- 1 Select the SQL-Station Debugger icon from your Start menu to run SQL-Station Debugger.

SQL-Station Debugger opens and displays the Connect dialog.

- 2 Enter a connect string to a server that contains the tutorial objects.
- 3 The **Login Name** must be the owner of the tutorial objects. If the tutorial was installed at the same time as the SQL-Station Debugger product, the owner of the tutorial objects is the account that was specified in the Specify Debug Objects Owner dialog during installation.

You can also run the SQL-Station Debugger Tutorial Install from the Program Manager or run the `DBGTUTOR.SQL` file and install the objects in your own schema.

- 4 Enter the password for the login account. The **Server Type** must be Oracle.
- 5 Click **Connect**.

Exploring the Catalog Browser

The Catalog Browser displays a hierarchical list of server objects for each connection.

Notice that the visible portion of the Catalog Browser begins with an Owner icon for the account that you logged in as. Beneath the Owner icon is an *object type* icon for each different type of server object that can be owned: functions, indexes, packages, procedures, sequences, synonyms, tables, triggers, and views.

This section walks you through the various levels of the Catalog Browser.

- 1 Double-click the Procedure icon to see a list of all the procedures that your login account owns.

Double-clicking the Procedure icon displays the list of procedures in the Catalog Browser. you can also click on the + next to the Procedure icon in the Catalog Browser. These are the ways to *expand* an icon.

- 2 Now expand the `AnalyzeStudentLoad` icon by clicking the + next to it.

The icon expands to show its *components*. For procedures, the components are Code, Info, References, Referenced By, and Privileges. Other types of objects have different component icons.

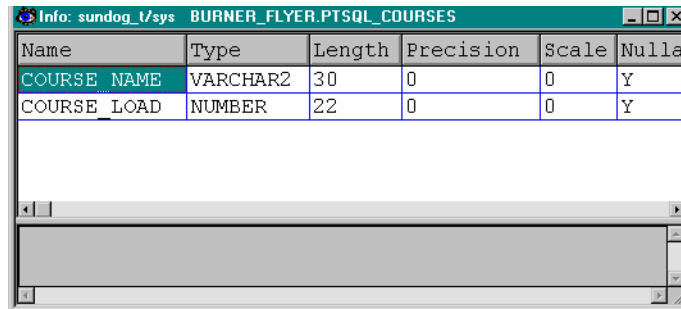
- 3 To display SQL code for `AnalyzeStudentLoad` in a separate Edit Window, single-click the Code icon.

The Edit Window appears and displays the reverse-engineered SQL code for the `AnalyzeStudentLoad` procedure.

- 4 To see what objects reference `AnalyzeStudentLoad`, return to the Catalog Browser and click the + next to the References icon, then click the + next to the Tables icon and next to the Packages icon. You can see that two tables and a package reference the `AnalyzeStudentLoad` procedure.


- 5 Click the + next to the `PTSQL_Student_Registration` table icon to display the component icons for that table.

- 6 Display the column descriptions for the table by clicking the Info icon. SQL-Station Debugger displays the **Name**, **Datatype**, **Length**, **Precision**, **Scale**, and **Nullability** for each column. You may need to scroll to the right to see the remaining columns of information.




Name	Type	Length	Precision	Scale	Nulla
COURSE_NAME	VARCHAR2	30	0	0	Y
COURSE_LOAD	NUMBER	22	0	0	Y

Click the **Data** icon to display the contents of the `PTSQL_Student_Registration` table in the right pane.

- 7 Scroll to the top of the Catalog Browser window. There is a **Version**  icon immediately below the **Connection** icon. To see information about the version of Oracle that is currently running, click the **Version** icon.


Debugging a Procedure

In these steps, you select a procedure for debugging, set two breakpoints, enter input parameters, and employ **Continue** and **Step Over** to move through the procedure.

- 1 In the Catalog Browser, single-click the `AnalyzeStudentLoad` icon to highlight it, then click the **Debug**  button or right-click the `AnalyzeStudentLoad` icon and choose **Debug** from the mouse-button menu.

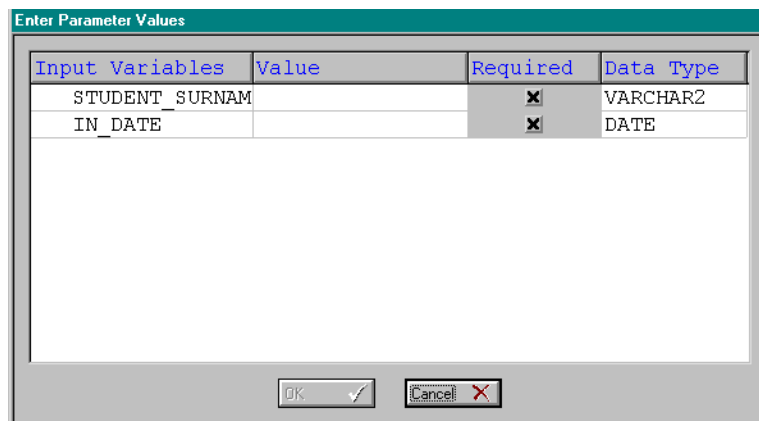
SQL-Station Debugger displays the Debugger in a separate window. The top pane of the Debugger is a **Source** pane that contains the SQL code for the `AnalyzeStudentLoad` procedure. The first executable line is highlighted.

- 2 To provide more room to work, click the **Maximize** button in the upper right-hand corner of the Debugger.

- 3 Set two breakpoints. The first breakpoint is two lines below the highlighted line. Double-click the line that reads `open get_course_names` to set the first breakpoint. Move down 8 lines and double-click to set a breakpoint at the line that reads `end loop`.
- 4 Click the Run  toolbar button.

SQL-Station Debugger displays the Enter Parameter Values dialog so that you can supply the necessary inputs.

- 5 Enter **FARLEY** (in all caps) for the `STUDENT_SURNAME` and **3-MAR-96** for the `IN_DATE`, then press **Tab**, **Enter**, or the down-arrow key after entering the date.



The dialog box titled "Enter Parameter Values" contains a table with the following data:


Input Variables	Value	Required	Data Type
STUDENT_SURNAM		<input checked="" type="checkbox"/>	VARCHAR2
IN_DATE		<input checked="" type="checkbox"/>	DATE


At the bottom of the dialog are "OK" and "Cancel" buttons.

- 6 Click OK. SQL-Station Debugger executes the code to the first breakpoint (the location is marked by a yellow Position arrow).

At this point, all variables within scope are added to the Variable Tree. Each component in the Variable Tree is expandable as long as there is a plus symbol (+) to the left of its icon.



By default, all local variables are also added to the Watch Pane to the right of the Variable Tree. You may need to resize the Watch pane to view the variables. To do this, position the mouse cursor between the

Watch pane and the Call Tree pane, so that the mouse pointer changes to this: . Then drag to the right until you reach the desired width.

Click the Continue  toolbar button to run to the next breakpoint. This executes one iteration of the loop. Notice that the value for `crse_name` variable reads LAW. Now click the Continue button again to run one more iteration of the loop. Notice that the `crse_name` variable now says ART.


- 7 You can clear breakpoints and set new ones during a debugging session. To deactivate the breakpoint at `end loop`, double-click the line. The solid circle becomes a hollow circle and the breakpoint is no longer active. The hollow circle reminds you where the breakpoint was, in case you want to set it again.

If you want to get rid of this reminder, double-click the line one more time. There is no difference in functionality between a line marked by a hollow circle and a line with no circle at all.


- 8 To practice moving through a debugging session, click the Step Over  toolbar button until the yellow Position arrow returns to `end loop`. Watch the values of the variables in the Watch pane after each step. Notice that you now see the variable values for ENGLISH.
- 9 Click Continue. The procedure runs to completion and SQL-Station Debugger displays the Results window which shows the value of the output parameters.
- 10 To widen the **Output Variables** column, position the mouse cursor over the right-hand side of the column header, so that the mouse pointer changes to this: . Then drag until the column is the desired width.
- 11 Notice that the value of the `LOAD_PER_COURSE` variable is 8.666.... Click OK to dismiss the Results dialog, but leave your debugging session as it is now, because you will work with it again in the next part of this tutorial.


Changing Values

In the following steps, you restart the debugging session that you ran in the previous steps. Then you change a variable value and notice the effect on other variables.

- 1 Remove the breakpoints at the lines `open get_course_name` and `end loop` by double-clicking the lines.
- 2 Set a breakpoint at the line `load_per_course := tot_load / num_courses`, which is the second line from the bottom.
- 3 Click the Run Debugger  toolbar button.

SQL-Station Debugger displays the Enter Parameter Values dialog.

- 4 In the `IN_DATE` field's **Value** column enter **3-SEP-96**. Press **Tab**, **Enter**, or the down-arrow key and click **OK**. When the session runs to the first breakpoint, look at the variable values. At this point, `tot_load = 24` and `num_courses = 3`.
- 5 In the Variable Tree, click on the + next to `Out`. Select `load_per_course` and click **Add**, directly above the Variable Tree. This adds its parameter value to the Watch pane. The value for `load_per_course` is null.
- 6 In the Watch pane, click in the **Value** field next to `tot_load` and enter **30**. Press **Enter**, **Tab**, or the down arrow key to enter the value.
- 7 Click the Step Over  toolbar button once and notice that `load_per_course = 10`, indicating that the assignment has taken place.
- 8 Click the Step Over toolbar button again to run the procedure to the end. SQL-Station Debugger displays the Results dialog. Notice that `total_load = 24`, `num_courses = 3`, and `load_per_course = 10`. (The variable `total_load` was set to `tot_load` prior to the reassignment. As a result, `total_load` is still equal to 24.)

- 9 In the Debugger window, click the Close  button to return to the Catalog Browser.


Note • Be sure to click the Debugger Close button. Clicking the SQL-Station Close button will close the application.

Working with the Call Tree

When you are debugging an object that has dependent modules, these modules appear in the Call Tree. (If they have been placed on the Global Exclusion List, they do not appear on the Call Tree, however. See the section *Excluding Modules* in the chapter *Using the Debugger*.) You can display the code for any of these dependent modules, place breakpoints in them, and choose whether or not to step into them as you execute the code of the parent object.

If one of the child modules is a package, you can choose whether to display the package header by setting a preference.

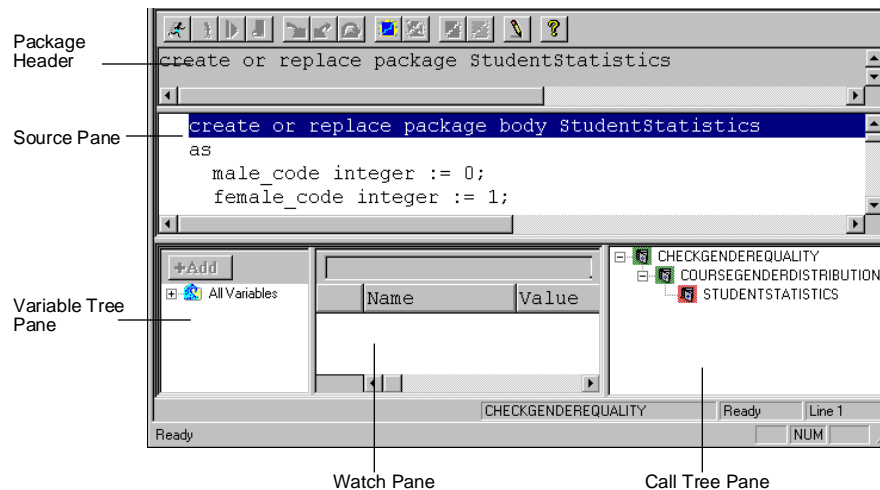
The next item that you're going to debug is a function that has two child modules. One of these is a package, so you begin by setting a Debugger Preference so that the package header will display.


- 1 Choose **Edit, Preferences**. The Debugger tab displays by default. Click the box next to **Display package headers** so that it is checked. Close **Preferences** by clicking Save as Default.
- 2 In the Catalog Browser, expand the Function icon under your login account icon to display the list of functions. Click `CheckGenderEquality` to select it, and then choose **Debug** from the right mouse button menu or click the **Debug**  toolbar button.


The `CheckGenderEquality` function checks one course at a time and reports what percent of the students registered for the course are males.

Working with the Call Tree


- 3 Click the maximize arrow in the upper right corner of the Debugger to maximize the Debugger window. This is optional, but gives you more room to work.
- 4 Look at the Call Tree pane and notice that there are two generations of dependent modules. The code for the parent module is displayed in the Debugger Source pane. Click the COURSEGENDERDISTRIBUTION module in the Call Tree and notice that its code is now displayed in the Source pane.
- 5 Click the STUDENTSTATISTICS module in the Call Tree. Its code displays in the Source pane. Since this is a package, and you enabled Show Package headers, a Package Header pane displays above the Source pane, showing the contents of the package header





In the Call Tree, click the parent module (CHECKGENDEREQUALITY) to display its code in the Source pane. Set a breakpoint at the line that reads `CourseGenderDistribution (this_course, reg_date, tmp_percent)`. (It's the 17th line.) You can set the breakpoint by double-clicking the line, by clicking the Set Breakpoint  toolbar button, or by choosing **Breakpoints, Set**.

- 6 In the Call Tree, click the `COURSEGENDERDISTRIBUTION` module to display its code in the Source pane. Set a breakpoint at the line that reads `percent_male := num_male / num_tot`. It's the third line from the end.
- 7 Click the Run  toolbar button. SQL-Station Debugger displays the Enter Parameter Values dialog. Click the **Value** cell next to `REG_DATE` and enter **3-MAR-96**. Press **Tab**, **Enter**, or the down arrow key and then click **OK**.

SQL-Station Debugger runs the function to the first breakpoint and adds the local variables that are in scope to the Watch pane. Notice that the value of `this_course` is `Chemistry`. (You may need to resize the Watch pane by dragging the border between the Watch pane and the Call Tree pane to the right.)



- 8 Click the Continue  toolbar button. SQL-Station Debugger continues to the next breakpoint. In this case, the next breakpoint is in the `COURSEGENDERDISTRIBUTION` module, so SQL-Station Debugger displays the code for that module in the Source pane.

Notice that different variables are listed in the Variable Tree.

- 9 Click Continue again to execute to the next breakpoint. Since the loop that the breakpoint is in is not completed, the next breakpoint is the first one you set (`CourseGenderDistribution (this_course, reg_date, tmp_percent)`); the yellow Position arrow once again points at this breakpoint. Notice that the value of `course` is now `Biology`.
- 10 Now click the Step Into  toolbar button. SQL-Station Debugger “steps into” the called module and displays its code in the Source pane. This time, however, the Position arrow is at the first executable line in the called module, rather than at the breakpoint.
- 11 Click the Step Out Of  toolbar button. SQL-Station Debugger steps out of the called module and returns to the calling module at the line following the call.

Working with the Call Tree

- 12 At this point, you can continue to explore the debugging session by experimenting with the Continue, Step Into, Step Over, and Step Out Of options.

To terminate the session at any time, click the Stop Debugging  toolbar button. Then click the Close  button to return to the Catalog Browser.

Note • Be sure to click the Debugger Close button. Clicking the SQL-Station Close button will close the application.

This completes the SQL-Station Debugger tutorial.

The Catalog Browser

This chapter describes the functionality available in the Catalog Browser.


Displaying the Catalog Browser	4-2
The Contents of the Catalog Browser	4-4
Catalog Browser Object Hierarchy	4-4
Object Types and Components	4-5
Function and Procedure Names	4-7
Debug Versions	4-8
Arranging the Catalog Browser Display	4-8
Filtering the Catalog Browser Display	4-9
Executing Server Objects	4-10
Entering Input Parameters	4-10
Executing and Committing	4-14
Displaying Tables	4-15
Displaying Table Contents in Grid Format	4-15
Displaying Table Structure	4-15
Displaying Object Code in an Edit Window	4-16
Dropping Objects	4-16

Displaying the Catalog Browser

The Catalog Browser is a versatile environment that permits you to view the contents of a database server and examine the structure, dependencies, code, and behavior of the listed objects. You can then execute any function or procedure, debug any function, procedure, or trigger, or display an object's code in the Edit window.

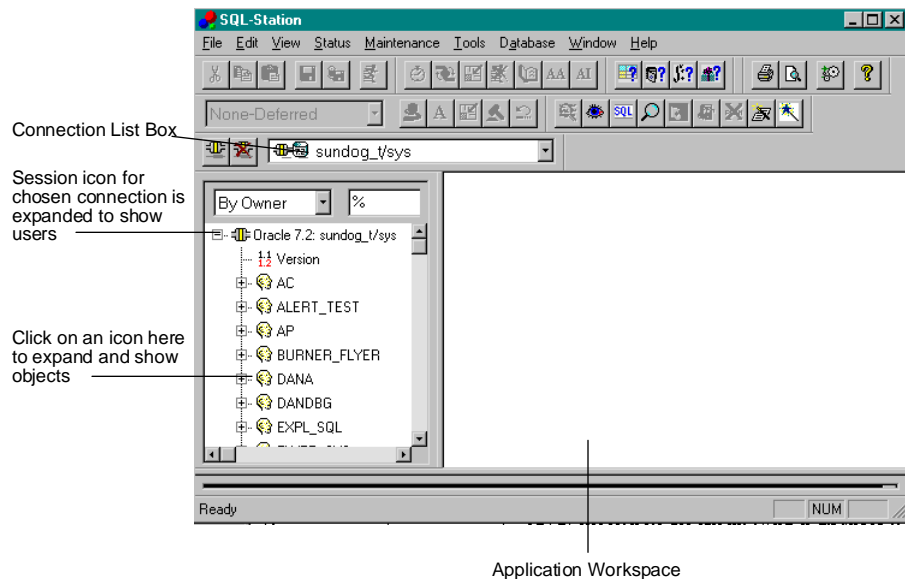
To display the Catalog Browser, follow these steps:

- 1 Connect to a server (refer to the section *Connecting to a Server* in the *Getting Started* chapter for additional information). SQL-Station Debugger automatically opens a Catalog Browser.


The Catalog Browser contains a Session  icon for the active connection, which is expanded to show owner icons. The objects that are visible under each owner depend on the permissions of that owner.

Each additional connection is represented by a Session icon.

Displaying the Catalog Browser



In the Catalog Browser, the icon for the session highlighted in the Connection list box is expanded to show the list of owners on the server. The icon for the current owner expands to show the list of object types for that owner. Each session icon, when expanded, shows only those objects on the specified server for which that owner has permissions.

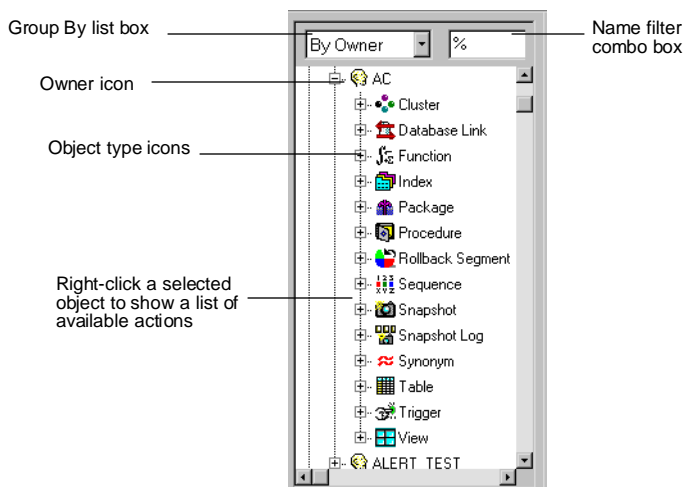
- 2 SQL-Station Debugger supports multiple Catalog Browsers. To open additional Catalog Browsers, click the New Catalog Browser  toolbar button or choose **Tools, Catalog Browser**.

SQL-Station Debugger displays an additional Catalog Browser.

- 3 Use the **Window** menu to tile, cascade, or switch between these windows. Click the Maximize button in the upper right-hand corner to maximize a Browser.

The Contents of the Catalog Browser

When you first display a Catalog Browser, it shows an icon for each server connection. The icon for the connection that is selected in the Connection list is expanded to show icons for each owner, and the icon for the current owner is expanded to show icons for each type of owned object—the *contents* of the owner icon.



Catalog Browser Object Hierarchy

The Catalog Browser displays a hierarchical list of server objects for each connection. When the display is sorted by owner (the default), the hierarchy is as follows:

Session

Owner

Object type (functions, procedures, tables, etc.)

Specific object

Components (Code, References, Referenced By)


Tables, views, and packages contain additional components.

Working with Icons in the Catalog Browser

The following techniques are available for working with icons in the Catalog Browser:

- Click a ⊕ to show the contents of the associated icon in the Catalog Browser.
- Single-click an icon to select it. Right-click a selected icon to display a menu of options.
- Double-click an icon to select it and display its contents.

Version, User, and Role Icons

At the level immediately following the session icon you will also find a Version  icon which displays information about the Oracle version currently running on the server. Following the owner icons you will find a User icon and a Role icon. These allow you to view the creating code and privileges for each user and role defined on the server.

Object Types and Components

The Catalog Browser displays the following object types:

- Clusters
- Database Links
- Functions
- Indexes
- Packages
- Procedures
- Rollback Segments




The Contents of the Catalog Browser

- Sequences
- Snapshots
- Snapshot Logs
- Synonyms
- Tables
- Triggers
- Views





Each object type icon can be expanded to show icons for specific objects of that type. Each specific object icon can then be expanded to show its components.

Most objects have component icons for Code, References, Referenced By, and Privileges. Tables, views, and packages have additional components.

Clicking a component icon has different effects depending on which type of icon you choose. The following table lists the results of clicking each type of component icon:

Component	Icon	Description
Code		Displays the SQL Code for the selected item in a separate Edit Window.
Data		Displays the contents of a table or view in grid form in a Data Window in the Application Workspace.
Info		Displays column information for a table or view in grid format.


The Contents of the Catalog Browser

Component	Icon	Description
References		Displays the objects that the selected object references.
Referenced By		Displays the objects that reference the selected object.
Privileges		Shows which users or roles have privileges on the selected object.
Comments		Displays any comments attached to a table or its columns in a separate dialog.

Function and Procedure Names

The Catalog Browser lists the input and output parameters of functions and procedures immediately following the name. The parameter name is followed by IN, OUT, or IN OUT, indicating whether it is an input parameter, output parameter, or both. This is followed by the datatype. If the Browser is arranged by Type rather than Owner, the owner of the object is listed after the parameters.

For functions, the input datatypes are followed by a colon and the output datatype.

 CALLEDPROC (OP1 IN NUMBER, OP2 IN NUMBER, OP3 IN OUT NUMBER)

Procedure

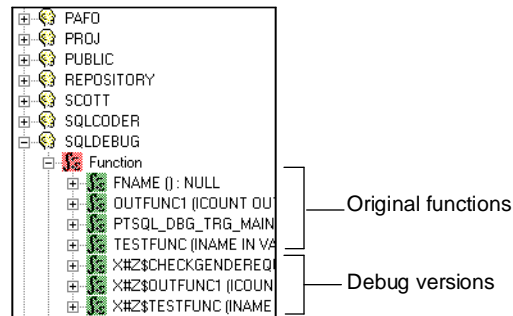
 TESTFUNC (INAME IN VARCHAR2) : VARCHAR2

Function

Debug Versions

The first time you debug a server object, SQL-Station Debugger creates a copy of the object, called a *debug version*. These debug versions have names beginning with x# and appear in the Catalog Browser.

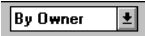
The illustration below shows a list of functions in the Catalog Browser with several debug versions at the end of the list.



For more information about debug versions of objects, refer to the chapters on *Using the Debugger* and *Object Maintenance*.

Arranging the Catalog Browser Display

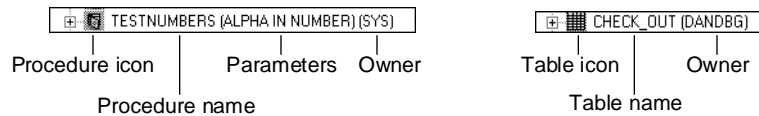
You can choose to group objects in the Catalog Browser by owner or by object type. The default grouping is by owner.

1 To change the grouping of objects in the Browser, click on the Group By list box ().


2 Click on **By Type** or **By Owner**.

The order of database objects displayed in the Browser changes to reflect your choice. When objects are arranged by type, the owner is listed in parentheses after the object's icon. The following figure shows examples of how two objects—a procedure and a table—appear in the Catalog Browser when the listing is By Type:


Filtering the Catalog Browser Display

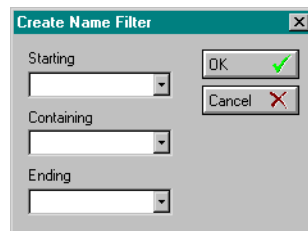


Filtering the Catalog Browser Display

You can use standard SQL wildcards to limit the list of objects. The current filtering specifications are displayed in the Name Filter combo box () at the top of the Catalog Browser. There are two ways to change the filtering specifications:

Using the Create Name Filter Dialog

- 1 Click the Edit Name Filter  toolbar button to display the Create Name Filter dialog.
- 2 Enter specifications into the boxes using standard SQL wildcards (% for any number of characters, _ for a single character).
- 3 Click **OK**.



Using the Name Filter Combo Box


- 1 Type the specification directly into the Name Filter combo box, using the standard SQL wildcards.
- 2 Press **Enter**.

Executing Server Objects

You can execute any procedure, function, package public procedure, or package public function. If the object requires inputs, SQL-Station Debugger displays its input and output parameters in the right pane of the Browser. You can then enter input values, run the code, and optionally commit it or roll it back.

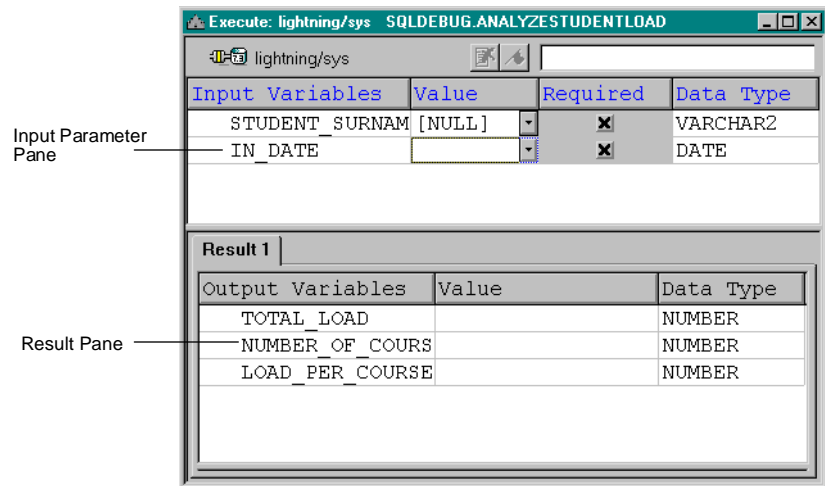
Entering Input Parameters

To display the input and output parameter grids for a procedure, function, or package public procedure or function, follow these steps:

- 1 Click the object's icon in the Catalog Browser so that the icon and name are highlighted.
- 2 Click the Execute  toolbar button. You can also right-click and choose **Execute** from the pop-up menu.

The right pane of the Catalog Browser displays an input parameter grid in the top subpane and a Results subpane under it.

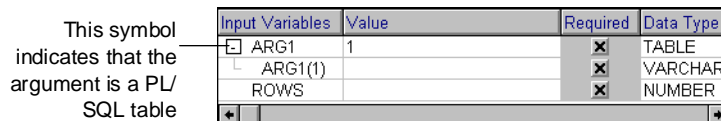
- 3 Enter a value for each input parameter and press **Enter** or the down arrow key.
 - An **x** in the **Required** box means that there is no default and you are required to enter a value.
 - To enter a Null value, click the down arrow at the right of the **Value** cell, and choose **NULL** from the drop-down list.
 - The drop-down list for boolean values contains the following options: **NULL**, **TRUE**, **FALSE**, and **USE DEFAULT**.



PL/SQL Table Inputs

SQL-Station Debugger supports PL/SQL tables as input parameters.

When you execute an object that has a table as an argument, the argument appears in the Input Parameter pane with a box symbol to its left.



If You Know the Array Size

- 1 Enter the array size in the Value cell of the table argument.

SQL-Station Debugger opens a table array with the specified number of lines.

- 2 Enter a number in each Value cell and press Tab, Enter, or the down arrow key after each one.

After the last number is entered, the table array closes, and you can enter other arguments, if there are any. You can enter a large number and then leave some lines of the table array blank. SQL-Station Debugger removes them before execution.

To Enter an Indeterminate Number of Inputs

Use the following technique if you don't know exactly how many values you will be inputting:

- 1 Don't type anything in the Value cell of the table argument. Leave the default value of 1.
- 2 Type the first value in the Arg*n*(1) Value cell and press **Tab**, **Enter**, or the down arrow key.

SQL-Station Debugger generates a new line for the next input value.

- 3 After entering the final value, press **Tab**, **Enter**, or the down arrow key *twice*.

The final blank line in the table array is removed before execution.

PL/SQL Table Output Size

If the output is a table, the size defaults to the PL/SQL table size specified in Preferences. If the output table is bigger than the default size, you will get an error when you execute the server object. In such cases, you must manually set the desired size of the output table so that it is big enough to hold the results.

Supported Parameter Types

The following table lists parameter types that are currently supported by SQL-Station Debugger:

Scalar Types Supported	Composite Types Supported
number	arrays of number

varchar	arrays of varchar
varchar2	arrays of varchar2
char	arrays of char
rowid	arrays of character
date	arrays of date
boolean	arrays of string
dec	arrays of dec
decimal	arrays of decimal
double precision	arrays of double precision
float	arrays of float
integer	arrays of integer
int	arrays of int
long	arrays of numeric
numeric	arrays of real
real	arrays of smallint
smallint	
string	
character	



The following table lists parameter types not currently supported by SQL-Station Debugger:


Scalar Types Not Supported	Composite Types Not Supported
mlslabel	records
raw	arrays of boolean
long raw	arrays of rowid
binary_integer	arrays of mlslabel
natural	arrays of raw
positive	arrays of long raw
	arrays of binary_integer
	arrays of natural
	arrays of positive

Executing and Committing

Once you have entered the input parameters, you can run the procedure, observe the new output values, and commit or roll back the changes.

Follow these steps to change input parameter values, run the procedure or function, and commit the changes:

- 1 **To execute the procedure**, click the Execute  toolbar button or choose **Tools, Procedure Execution**. SQL-Station Debugger displays the new output values in the Result pane.
- 2 **To commit changes immediately**, click the Commit  toolbar button or choose **Database, Commit**. The Auto Commit option does not affect object execution.

- 3 To roll back changes, click the Rollback  toolbar button.
 - 4 To reset all parameter values to zero, choose **Edit, Clear Arguments**.
- If you neither commit nor roll back the transaction, changes will be committed when you close the connection.
 - After execution, a message pane displays at the bottom of the Results window. It contains any error messages that may have been generated, or a message saying that execution was successful.
 - If **DBMS Output** is enabled in Preferences, the message pane also contains the DBMS output if there is any.

Displaying Tables

You can display the contents of any table in grid format. You can also display the column structure of any table. (To display a table's code, refer to *Displaying Object Code in an Edit Window* in this chapter.)

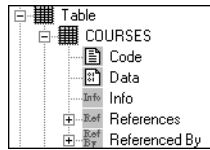
Displaying Table Contents in Grid Format

- 1 In the Catalog Browser, expand the table icon so that its component icons are visible.
 - 2 Click once on the table's Data icon.
- SQL-Station Debugger displays the table contents in grid format in the right Catalog Browser pane.

Displaying Table Structure

- 1 In the Catalog Browser, expand the table icon so that its component icons are visible.

Displaying Object Code in an Edit Window



- 2 Click once on the table's Info icon.

The table column definitions display in grid format in the right pane of the Application Workspace.

Displaying Object Code in an Edit Window

From the Catalog Browser, you can display the reverse-engineered code for a selected object in a separate Edit Window using the following steps:


- 1 In the Catalog Browser, expand the icon for the object you want to display so that the Code icon is visible.
- 2 Single-click the Code icon.

SQL-Station Debugger displays an Edit Window containing the reverse-engineered code for the selected server object.

See the chapter on *Using the Edit Window* for details on how to work in the Edit Window environment.

Dropping Objects

You can drop any object that is displayed in the Catalog Browser.

- 1 Click the icon for the object you want to drop, so that the icon and name are highlighted.
- 2 Click the Drop  toolbar button (at the top of the Catalog Browser). You can also right-click and choose **Drop** from the pop-up menu.

The object is dropped from the database.

Using the Debugger

This chapter describes SQL-Station Debugger's Debugging environment. It covers the debugging process, and tells you how to set breakpoints, step through a debugging session, evaluate expressions, and change variable values.

Using the Debugger	5-3
Beginning the Debug Process	5-3
To Debug a Server Object	5-3
To Debug a SQL Text File	5-4
The Debugging Environment	5-4
Debugger Settings	5-5
Displaying Packages	5-6
Steps in Debugging	5-7
Excluding Modules	5-10
Working with Breakpoints	5-11
Setting Breakpoints	5-12
To Disable a Breakpoint	5-13
To Reactivate a Breakpoint	5-13

To Delete a Breakpoint	5-14
Cycling Breakpoint States	5-14
Starting the Debug Session	5-14
Entering Parameters	5-15
Using the Watch Pane	5-17
Adding a Variable	5-17
Evaluating Expressions	5-18
Changing Variable Values	5-18
Deleting Lines	5-19
Watching the Flow	5-20
Continue to Next Breakpoint	5-20
Step Into	5-20
Step Over	5-21
Step Out Of	5-22
Ending the Debugging Session	5-22

Using the Debugger


The Debugger provides an environment for tracking the execution sequence of selected SQL code in precise detail through the use of breakpoints. It provides the following facilities:

- You can step through the code one command at a time or run to the next breakpoint.
- You can step into, over, or out of any module.
- At each point, you can see the value of all variables that are in scope and watch these values change as you move through the procedure.
- You can test scenarios by changing variable values or input parameter values and watching the runtime results.

Beginning the Debug Process

You can debug a function, procedure, or trigger, as well as functions and procedures that are part of a package body. SQL-Station Debugger will debug both server objects and SQL text files.

To Debug a Server Object

- 1 Highlight the object's icon in the Catalog Browser.
- 2 Perform one of the following three actions: choose **Debug** from the right mouse-button menu, choose **Tools, Debug** from the SQL-Station main menu, or click the **Debug**  toolbar button.

SQL-Station Debugger reverse-engineers the selected object and displays a Debugger in the Application Workspace to the right of the Catalog Browser. The object's code is in the Source pane at the top of the window. The first executable line is highlighted.

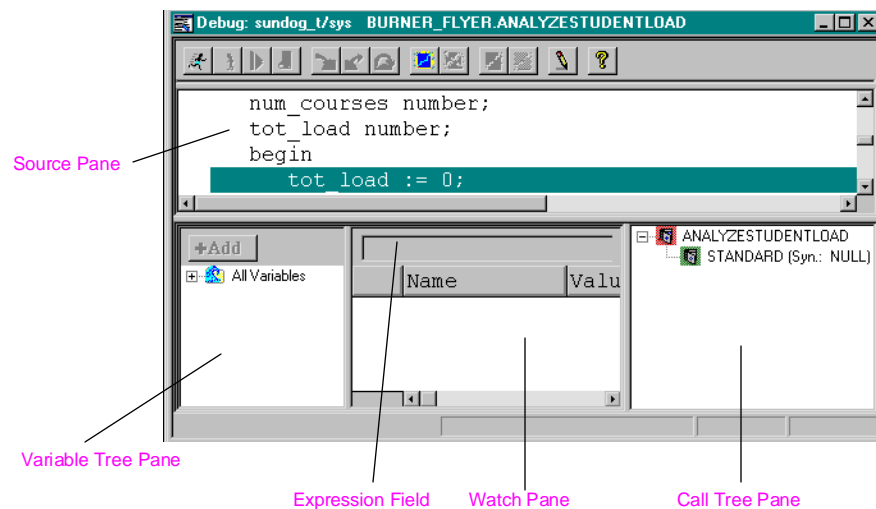
To Debug a SQL Text File

- 1 Choose **File, File Debug**.
- 2 Navigate to the directory that contains the file and double-click or highlight the file. Choose **OK**.

SQL-Station Debugger displays the file's SQL code in the Source pane of the Debugger. The first executable line is highlighted.

The Debugging Environment

The illustration below shows the Debugger as it appears when you first invoke it on an object:



Source Pane

Shows the code for the selected node in the Call Tree

Expression Field

When you enter an expression in this field and press Enter, SQL-Station Debugger evaluates the expression and displays the result in the Watch pane.

Variable Tree

Displays the variables that are in scope.

Watch Pane

Displays the values of variables that you select from the Variable Tree. Also displays expressions entered in the Expression field.

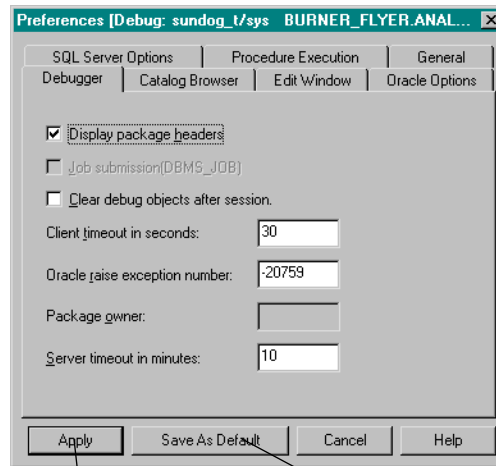
Call Tree

Shows the component modules of the object being debugged. The code for the selected node displays in the Source pane.

Debugger Settings

You can set a number of behaviors in the Debugger. SQL-Station Debugger allows you to set default behaviors as well as temporarily override settings during an active Debugger session.

You can set the default behavior for the Debugger by choosing **Edit, Preferences** from the SQL-Station main menu, then choose the **Debugger** tab. See the section on *Debugger Preferences* in this chapter for more information on setting these preferences.



To override the defaults for the current session, choose Apply.

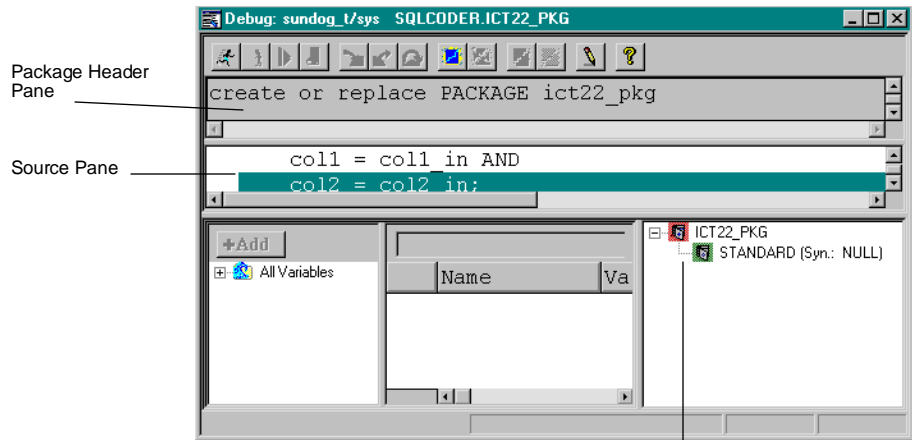
To change the Default Settings, choose Save as Default.

To override the default behavior or change the defaults in an active Debugger session, follow these steps:

- 1 Choose **Edit, Preferences**, and then select the **Debugger** tab to display the Debugger preferences dialog.
- 2 Make the changes you desire. The settings are discussed in the *Debugger Preferences* section in Chapter 2. The **Display package headers** setting is discussed in the next section, *Displaying Packages*.
- 3 To override the default settings for the current session only, choose **Apply**. To change the defaults, choose **Save As Default**.

Displaying Packages


When a package is selected in the Call Tree, SQL-Station Debugger optionally provides an extra pane in the Debugger to display the package header. This Package Header pane disappears when a non-package object is selected. The Package Header pane can be enabled or disabled by toggling a check box in the Debugger tab of the Preferences dialog.



When a package procedure or a package is selected in the Call Tree, its header is displayed in the Package Header Pane.

Steps in Debugging

The following steps provide an overview of the debugging procedure. The remaining topics in this chapter elaborate on various aspects of the debugging process.


- 1 To debug a server object, select a function, procedure, package function, package procedure, or trigger, and invoke the Debugger. To invoke the Debugger, do one of the following:
 - If selecting the object in the Catalog Browser, click the **Debug**  toolbar button, choose **Debug** from the right mouse-button menu, or choose **Tools, Debug**.
 - For a SQL file object, choose **File, Debug File** and select the file name.

Note • The debugging of dependent procedures and functions is supported from the Catalog Browser only.

A Debugger displays in a separate window. The SQL code for the object appears in the Source pane at the top and the first executable line is highlighted.

- 2 Set one or more breakpoints. Click on nodes in the Call Tree to display the associated code in the Source pane so that you can add breakpoints wherever you need them.

If any of the child modules have been excluded from debugging by being placed on the Global Exclusion List, they do not appear on the Call Tree. The Global Exclusion List is maintained by the DBA. Refer to the next section, *Excluding Modules*, for additional information.

- 3 Start the debug session by clicking the Run  toolbar button or by choosing **Debug, Go**. If the procedure requires input parameters, SQL-Station Debugger displays a dialog that lists the input parameters and requests values for each. See the section *Entering Parameters* later in this chapter for more information on entering values in this dialog.


SQL-Station Debugger executes the code up to but not including the line where the first breakpoint is set. The yellow Position arrow points to the line that will execute next

Note • Each time you click Run or Restart, or select **Debug, Go** or **Debug, Restart** during a debugging session, SQL-Station Debugger checks the datestamp of the original object. If the datestamp has changed since the creation of the debug version, a message box displays giving you the option of continuing with the current debug version or recompiling the debug version to reflect any new changes in the object.

- 4 At this point, you can select specific variables to be displayed in the Watch pane. By default, all local variables are added. To add additional variables, expand the Variable Tree so that the variable you want to select is visible and do one of the following:
 - Single-click the variable and click the **Add** button directly above the Variable Tree.

- Right-click the variable and choose **Add Variable** from the pop-up menu.
- Single-click the variable and choose **Debug, Add Variable** from the menu bar.

When a variable is added, all variables appearing on its branch are also added. To add all the variables to the Watch pane, double-click All Variables (the top level of the Variable Tree).

- 5 Examine the displayed variable values. You can test different scenarios by changing the variable values in the Watch pane.
- 6 You can enter and execute expressions in the field above the Watch pane. These expressions can include global variables, and any of the PL/SQL DML operators. For example, SQL-Station Debugger can evaluate expressions such as `ip1+5*50`. To evaluate an expression, the debug session must already have run to the first breakpoint.
- 7 Continue execution by choosing one of the following:
 - To run to the next breakpoint, click the **Continue**  toolbar button. If the last breakpoint has already been passed, the code runs to completion.
 - If the position arrow is pointing at a procedure, package, or function, you can click **Step Into** to display its code in the Source pane and watch its execution. To execute it transparently, and move to the next step of the current procedure, click **Step Over**. Click **Step Out Of** to exit from the current procedure and return to the parent level.

Notice that the variable values in the Watch pane change as you step through the code.

- 8 Continue to execute the procedure in steps or between breakpoints until SQL-Station Debugger displays the Results dialog.
- 9 To restart execution from the beginning and run to the first breakpoint, choose **Restart**. To halt debugging and return to the

beginning, choose **Halt**. To end the debugging session and close the Debugger, choose **Exit**.

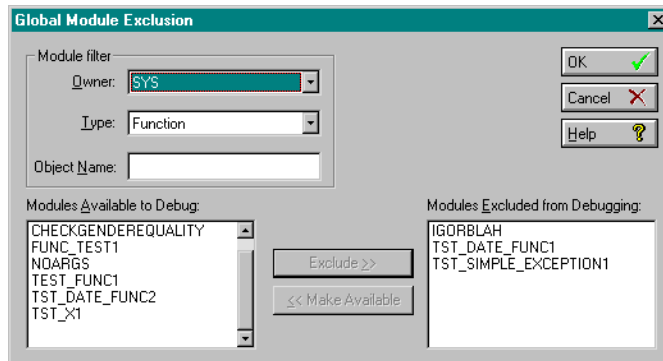
Excluding Modules

Many objects reference other objects. When you invoke the Debugger on a server object, the Debugger creates a debug version of the selected object and of each child module that it references. In some cases, however, SQL-Station Debugger may not be able to create debug versions of all child modules. The modules may be encrypted, for example, or the source code may not be available due to database security.

Note • For file objects, SQL-Station Debugger creates only the top level of the Call Tree.

Debugging fails if you attempt to debug an object whose child modules cannot be reverse-engineered and copied into debug versions. The Global Exclusion feature alleviates this constraint. When child modules are put on the Global Exclusion List, SQL-Station Debugger skips over them when the parent object is debugged.

The DBA maintains the Global Exclusion List of modules that cannot be debugged by users. When a child module is on the Global Exclusion List, SQL-Station Debugger does not generate a debug version. The child module does not appear on the Call Tree, and you cannot display its code, set breakpoints in it, or step into it during debugging.



To exclude an object so that it cannot be debugged, follow these steps:

- 1 In SQL-Station Debugger's Connection list box, choose a session for which the user has full administrative privileges.
- 2 Choose **Maintenance, Module Exclusion**.
- 3 *Optional.* To filter the listing of available modules, enter the desired restriction in the **Object Name** field, using the usual SQL wildcards.
- 4 Use the **Exclude** and **Make Available** buttons to transfer objects into the desired category.

Single-clicking an object toggles it between selected and unselected. To select multiple objects, single-click each object and transfer them to the desired category.

- 5 Click **OK**.

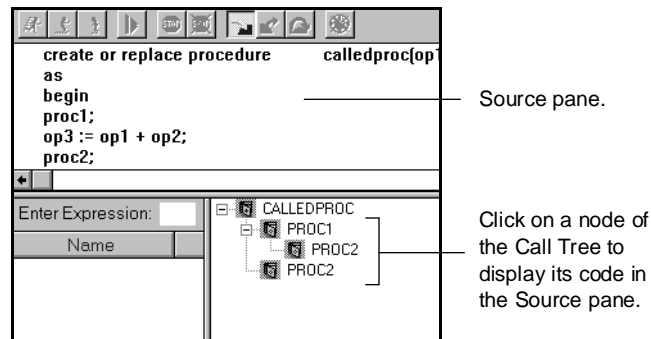
Working with Breakpoints


Set a breakpoint wherever you want execution to stop. A line of code in the Source pane can have one of three breakpoint states: no breakpoint, active breakpoint, or inactive (disabled) breakpoint.

Setting Breakpoints

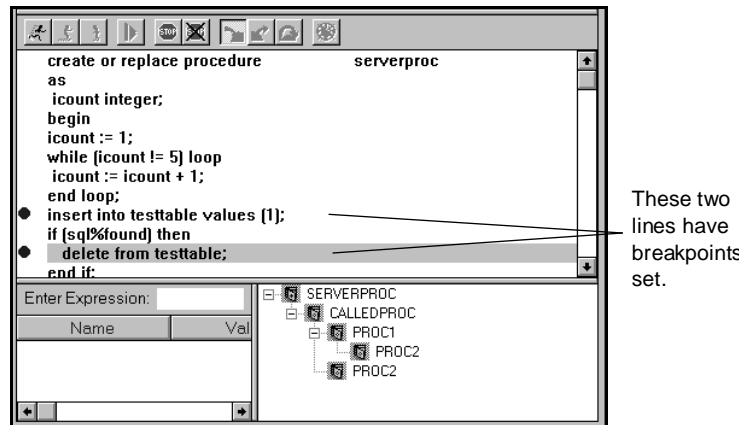
A solid, blue circle next to the line indicates an active breakpoint. To set a breakpoint, follow these steps:

- 1 Display the code in the Source pane of the Debugger. When you first invoke the Debugger, the Source pane displays the code for the top node—the top program module—of the Call Tree. To display the code for a component program module, click its node in the Call Tree. The code associated with that node then displays in the Source pane.




- 2 To set a breakpoint at a line, do one of the following:
 - Double-click on the line.
 - Click once on the line and click the Set Breakpoint  toolbar button.
 - Click once on the line and choose **Breakpoints, Set** from the main menu bar.

Note • If you attempt to place a breakpoint at a line that is not a logical stopping point, SQL-Station Debugger places the breakpoint at the next line that is appropriate.



To Disable a Breakpoint

You can disable a breakpoint, but retain a marker in its place. To disable a breakpoint, do one of the following.

- Double-click on a line that has a breakpoint set.
- Click the line once and click the Disable Breakpoint  toolbar button.
- Click the line once and choose **Breakpoints, Disable** from the main menu bar.


When a breakpoint is disabled, the blue circle to the left of the line becomes a hollow circle.

To Reactivate a Breakpoint

To reactivate a disabled breakpoint (indicated by a hollow circle to the left of it), do one of the following:


- Double-click *twice* on a line that has a disabled breakpoint.

Starting the Debug Session

- Click once to highlight a line with a disabled breakpoint and click the Enable Breakpoint  toolbar button.
- Click once to highlight a line with a disabled breakpoint and choose **Breakpoints, Enable**.

To Delete a Breakpoint


To delete a breakpoint, do one of the following:

- Double-click on a line that has a disabled breakpoint (indicated by a hollow circle to the left of it).
- Double-click *twice* on a line that has an active breakpoint set (indicated by a solid circle to the left of it).
- Click once on the line and click the Delete Breakpoint  toolbar button.
- Click once on the line and choose **Breakpoints, Delete** from the main menu.

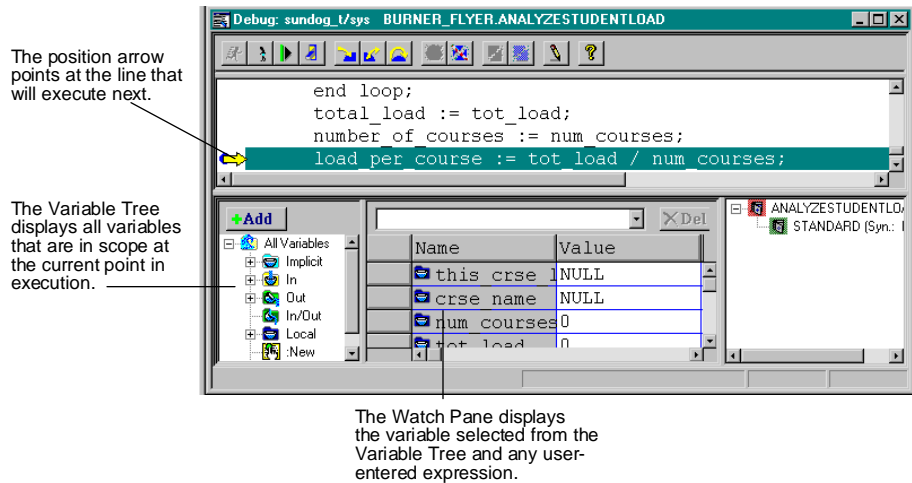
Cycling Breakpoint States

The preceding sections describe the setting of various breakpoint states as separate tasks. In fact, you can cycle a line through the three breakpoint states by continuing to double-click on it. The first double-click places an active breakpoint, the second double-click disables the breakpoint, and a third double-click removes the breakpoint.

Starting the Debug Session

After setting the initial breakpoints, start execution by clicking the Run  toolbar button, or by choosing **Debug, Go**. If no input parameters are required, SQL-Station Debugger executes the code to the first breakpoint and displays the value of each variable that is in scope in the Watch pane.

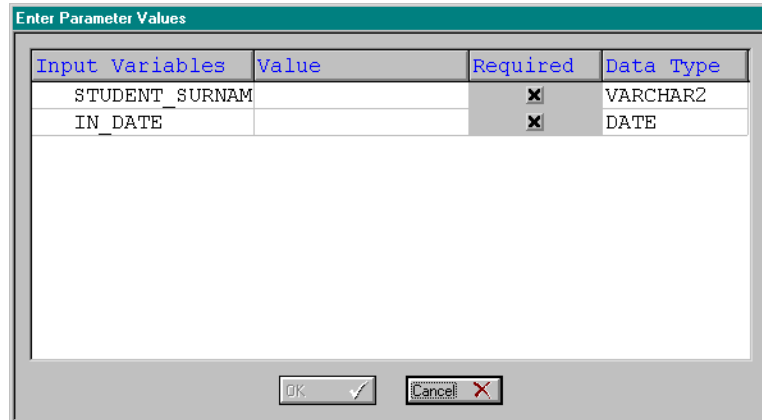
The *Position* arrow points at the line that will execute next.



If the object requires input parameters to execute, SQL-Station Debugger displays the Enter Parameter Values dialog.

Entering Parameters

When you start a debugging session on a procedure that requires input parameter values, SQL-Station Debugger displays the Enter Parameter Values dialog. You must provide values for each input parameter before SQL-Station Debugger can begin executing the code.



The dialog box titled "Enter Parameter Values" contains a table with four columns: "Input Variables", "Value", "Required", and "Data Type". The table has two rows of data. The first row shows "STUDENT_SURNAM" in the "Input Variables" column, an empty "Value" field, a required status of "X", and a "Data Type" of "VARCHAR2". The second row shows "IN_DATE" in the "Input Variables" column, an empty "Value" field, a required status of "X", and a "Data Type" of "DATE". Below the table is a large empty text area. At the bottom of the dialog are "OK" and "Cancel" buttons.

Input Variables	Value	Required	Data Type
STUDENT_SURNAM		X	VARCHAR2
IN_DATE		X	DATE

- 1 Click the **Value** field for the first input parameter.
- 2 Type a value and press the **Return** key to enter it. If you are entering more input values, you can press the **Tab** key or the down arrow key instead of pressing **Enter**.
- 3 Repeat for each input parameter. You must press **Enter**, **Tab**, or the down arrow key after entering the last value.
- 4 Click **OK** to close the dialog and return to the Debugger.

SQL-Station Debugger uses the specified input parameter values when it runs the procedure.

PL/SQL Table Inputs

SQL-Station Debugger supports PL/SQL tables as input parameters. See the section *PL/SQL Table Inputs* in this chapter for details on how to enter table parameters.

Using the Watch Pane

After the Debugger reaches the first breakpoint, you can select variables from the Variable Tree and add them to the Watch pane. This allows you to monitor specific variable values and to test different scenarios by changing these values. By default, all local variables are automatically added.

You can also add expressions to the Watch pane by using the Expression field.

Adding a Variable

To add a variable to the Watch pane, do the following:

- 1 After the Debugger executes the code to a breakpoint, expand the Variable Tree so that the variable you want to select is visible.
- 2 Add the variable to the Watch pane by doing one of the following:
 - Single-click the variable and click the **Add** button directly above the Variable Tree.
 - Right-click the variable and choose **Add Variable** from the pop-up menu.
 - Single-click the variable and choose **Debug, Add Variable** from the menu bar.

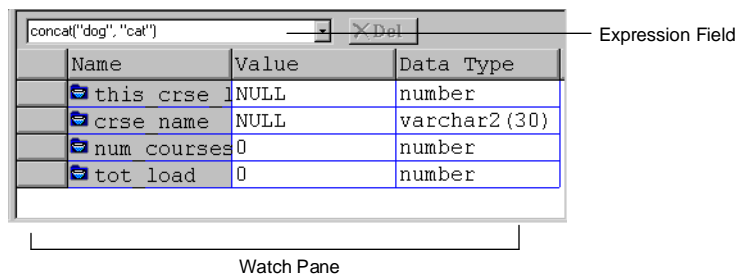
When a variable is added, all variables appearing underneath it on the same branch are also added.

To add *all* the variables to the Watch pane, double-click All Variables (the top level of the Variable Tree).

Evaluating Expressions

You can enter expressions for evaluation by typing them into the **Expressions** field in the Debugger. The current version of SQL-Station Debugger supports only expressions that include global variables, not local variables. To enter and evaluate an expression, follow these steps:

- 1 Run the debug session so that it is paused at a breakpoint. You cannot evaluate expressions either before beginning a session or after it has run to completion.
- 2 Click in the **Expression** field box.
- 3 Type in the expression.
- 4 Press **Enter**.



SQL-Station Debugger evaluates the expression and adds a line to the Watch pane showing the expression and its value.

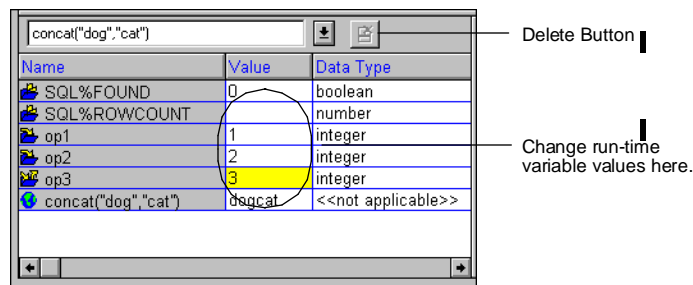
Changing Variable Values

At each point where execution stops during debugging, SQL-Station Debugger displays the values of variables that are in scope at the current position (indicated by the position arrow). The variables, their values, and the datatype are displayed in the Watch pane. To change a variable value, follow these steps:

- 1 In the Watch pane, click in the field that contains the variable value.

- 2 Enter the new value. SQL-Station Debugger allows you to enter a new value only if the variable is modifiable.
- 3 Press Enter.

When you resume execution, SQL-Station Debugger uses the new variable value. These changes are run-time only. To make permanent changes to an object, edit it in the Edit window. (See the chapter *Using the Edit Window*.)



Deleting Lines

You can remove unwanted variable or expression lines from the Watch pane by doing the following:

- 1 Click the button to the left of the Name in each line you want to remove.
- 2 Right-click the mouse **or** click the Delete button.

Note • The Delete button is located to the right of the Expression field, immediately above the Watch pane. You may need to resize the Watch pane to the right in order to access the Delete button.


Watching the Flow

The heart of SQL-Station Debugger's functionality is the control it offers you in the Debugger. Once you have set breakpoints, you can move through the execution in controlled stages, watching the value of variables as you go. At any point, you can enter expressions using global variables and can change the value of variables in order to test different execution scenarios.

In the following discussion, *program module* and *child module* are interchangeable terms that include procedures, functions, packages, and triggers.


After you have run the code to the first breakpoint, you have the following options: Continue, Step Into, Step Over, Step Out Of, or Stop. These options are available in the parent module and in any child module that you step into.

Continue to Next Breakpoint

To execute the code to the next breakpoint, click the Continue  toolbar button or choose **Debug, Continue**. SQL-Station Debugger executes all code up to but not including the line that has the breakpoint. If there is no other breakpoint, the code runs to completion.

All code between the two breakpoints executes, including any child modules. You see the values of variables only at points where execution stops. In order to see the code and variable values for child program modules, set a breakpoint inside the child module.

Step Into


When the position arrow is pointing at a line of code that calls a child program module—a function, procedure, or trigger—you can step into that module by clicking the Step Into  toolbar button or by choosing **Debug, Step Into**.

Note • To step into a trigger that is a child program module, you must first deploy its debug version on the server. See the section on *Trigger Maintenance* for information on how to deploy a debug version.

SQL-Station Debugger then displays the code for the child module in the Source pane, and places the position arrow at the first line of code in the child module. You can then move through the child module using the Continue, Step Into, and Step Over options. Choosing Step Out Of returns you to the parent module, in which case, the Position arrow will be pointing to the line of code following the one that called the child module


Note • When the position arrow is not pointing at a procedure or function call, Step Into executes the single current line of code and moves the position arrow to the next line.

Step Over

When the position arrow is pointing at a line of code that calls a child program module, you can execute that code transparently by clicking the Step Over  toolbar button or by choosing **Debug, Step Over**. SQL-Station Debugger executes the code for the module and places the position arrow at the next line of code in the current module. You do not see the values of any variables that are local to the child module.

When the position arrow is pointing at a line of code that does not call a child module, Step Over executes the single current line of code and moves the position arrow to the next line.


Step Out Of

Click the Step Out Of  toolbar button or choose **Debug, Step Out Of** to exit the current child module and return to the parent module. If you are already in the highest-level module in the Call Tree for the object being debugged, Step Out Of executes the remainder of the code for the object.

Ending the Debugging Session

You have three choices for terminating a current debugging session: Restart, Halt, and Stop

Restart

To return to the beginning of the session and execute to the first breakpoint, click the Restart  toolbar button or choose **Debug, Restart**. If the object requires input parameters, SQL-Station Debugger displays the Enter Parameter Values dialog with the previous values in place. You can accept the old values or enter new ones.

Halt

To terminate the current debugging session and return the position pointer to the first line of the top-level module, click the Halt  toolbar button.

Stop

To terminate the current debugging session and close the Debugger, choose **Debug, Stop Debugging**.

Using the Edit Window

This chapter describes how to use the Edit Window portion of SQL-Station Debugger's working environment.

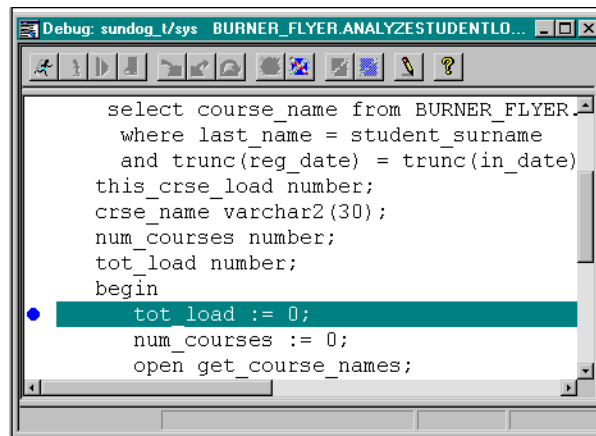
The Edit Window	6-3
Displaying an Edit Window	6-3
Using Multiple Windows	6-4
Edit Window Preferences	6-5
Setting the Result Display	6-6
Command Echoing	6-7
Executing SQL Code	6-7
Commit Options	6-8
Halting Execution	6-9
Display Options	6-9
Show Plan	6-9
Displaying DBMS Output	6-9
Editing Query Code	6-10
To Find and Replace Text	6-11
To Replace Text	6-11

Saving Text	6-12
Spooling Sessions to a Log File	6-13
To Stop Spooling	6-14

The Edit Window

The Edit Window is an editing environment in which you can create, edit, and execute SQL code. Changes that you make can be committed to the server.


The illustration below shows an Edit Window with reverse-engineered code from a server object. You can also display an empty Edit Window and type in code or insert the contents of a SQL text file.



Displaying an Edit Window

There are three ways to display an Edit Window in SQL-Station Debugger:

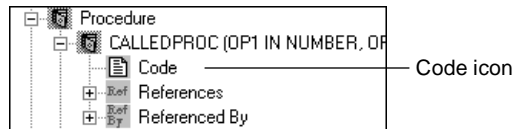
- *Edit the code in the Debugger source pane*

When you have been debugging an object and are ready to make changes to the code, click the Go to Edit Window  toolbar button or choose **Debug, Edit Window**. SQL-Station Debugger displays the SQL code that was in the Source pane of the Debugger in a new Edit Window, ready for editing.


- *Display the code for a server object*

Displaying an Edit Window

In the Catalog Browser, expand an object's icon so that the Code icon is visible. Click the Code icon. SQL-Station Debugger creates a new connection, reverse-engineers the selected server object, and displays the object's code in a new Edit Window.



■ *Display an empty Edit Window*

Click the SQL  toolbar button or choose **File, New** to display an empty Edit Window. You can then populate the Edit Window in any of the following ways:

- Type SQL code directly into the Editor.
- Choose **File, Insert** and select the name of the SQL text file to insert.
- Paste code copied from another SQL-Station Debugger Edit Window or from any SQL text file.

Using Multiple Windows

- 1 Open one or more additional Edit Windows using any of the techniques described in *Displaying an Edit Window*, above. Each one can contain different code.
- 2 Use the options on the **Window** menu to display the windows and to move between them

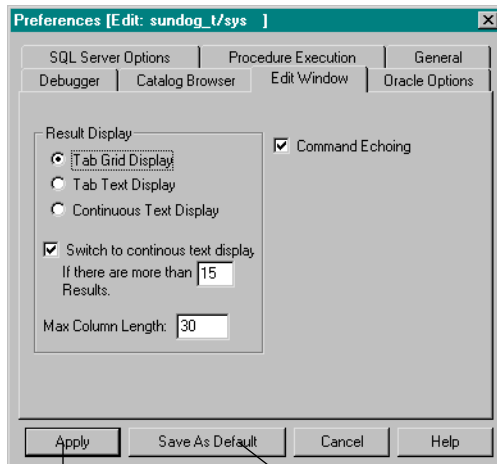
Note • If multiple Edit Windows are open, each one is numbered separately underneath **Window** in the SQL-Station Debugger menu bar.

You can cut, copy, and paste code between windows using the **Edit** menu options or the editing buttons provided in the toolbar.

Edit Window Preferences

You can choose whether the results of running a SQL query are displayed in grid form or in text form. You can also choose whether the command is echoed in a separate pane. You can set defaults for these items, and you can also change the settings temporarily without changing the defaults.


You can set the default behavior for the Edit Window by choosing **Edit, Preferences** and selecting the Edit Window tab. See *Edit Window Preferences* in the *Getting Started* chapter for a description of setting these preferences.



To override the defaults for the current session choose **Apply**.

To change the default settings, choose **Save as Default**.

To override the default behavior or change the defaults, follow these steps:

- 1 Choose **Edit, Preferences** or click the  toolbar button to display the Preferences dialog.

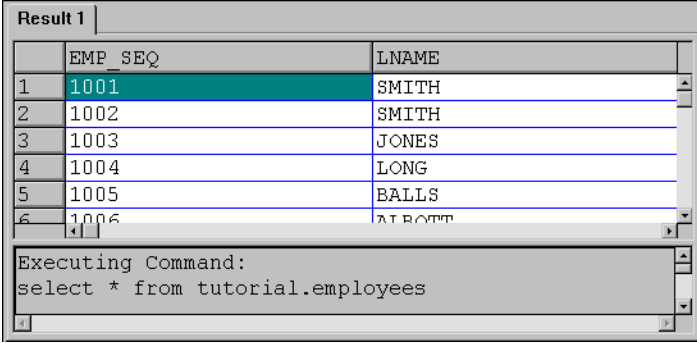
- 2 Make your desired changes. The **Result Display** and **Command Echoing** settings are discussed in more detail in the next section, *Setting the Result Display*.
- 3 To override the settings for the current session, click **Apply**. To change the defaults, choose **Save As Default**.

Setting the Result Display

You can choose to see the results of a query either in grid form or in text form.

Grid Display

To display the results of a query in grid form, choose **Tab Grid Display** in the **Result Display** section of the Preferences dialog. The result display then looks like this:

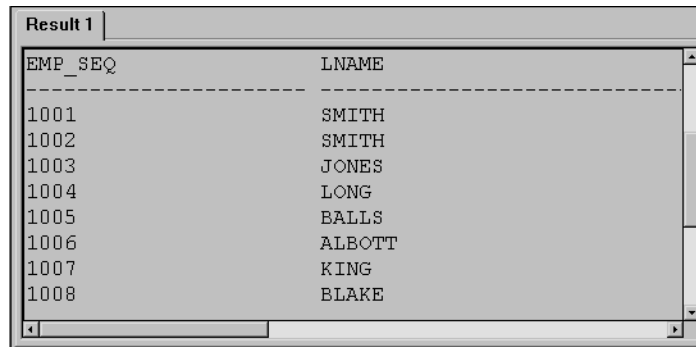


	EMP_SEQ	LNAME
1	1001	SMITH
2	1002	SMITH
3	1003	JONES
4	1004	LONG
5	1005	BALLS
6	1006	ALBERT

Executing Command:
select * from tutorial.employees

Text Display

To display the results of a query in text form, choose **Tab Text Display** in the **Result Display** section of the Preferences dialog. The illustration below is the same query result that is shown above in grid form, but this time it is displayed in text form:



The screenshot shows a window titled "Result 1" containing a table with two columns: "EMP_SEQ" and "LNAME". The table has eight rows of data. The first row is a header with dashed lines above and below it. The subsequent rows contain numerical values in the first column and names in the second column.

EMP_SEQ	LNAME
1001	SMITH
1002	SMITH
1003	JONES
1004	LONG
1005	BALLS
1006	ALBOTT
1007	KING
1008	BLAKE

Command Echoing

You can enable **Command Echoing** in the Preferences dialog—either as a default or as an override setting. SQL-Station Debugger then echoes the command along with the output. This is useful when the code includes several commands and therefore several results. The command appears in different places, depending on whether you choose text or grid for the result display.


Tab Grid Display The command appears in a separate pane below the result pane. Each result is in a separate pane. When you click a Result tab, the associated command appears in the Result pane below.

Tab Text Display The command appears in the result pane with the associated result.

Executing SQL Code

Once you have populated an Edit Window with SQL code, you can run that code to see the results.

- 1 Check to see that you have selected the Result Display options that you want. The default behavior is set in **Edit, Preferences**. If you want to see which command is associated with a particular result, be sure that **Command Echoing** is enabled.

- 2 If you want to run only a portion of the code in the query pane, highlight the desired portion of the code. To run all of the code, click anywhere in the query pane.
- 3 Click the Run  toolbar button or choose **SQL, Execute**. Note that this menu option may not be active if you have a current Debugging session.

SQL-Station Debugger runs the selected portion of the code and displays the results in the Result pane. Each command is displayed in a separate tab. If you enabled Command Echoing, there is an Echo pane below the Result pane. The command code associated with the selected result tab is displayed in this pane.

- 4 Result tab 1 displays the results of the first command. Click subsequent Result tabs to display the results of the other commands in the order they occur in the code.

Commit Options

You can choose to have changes committed automatically or to retain manual control.

Auto Commit

To have all changes committed automatically immediately upon execution, enable the **Auto Commit** option. When **Auto Commit** is enabled, you cannot roll back any transaction. To access this preference, check **Auto Commit** in the **Oracle Options** tab of the **Preferences** dialog.

Manual Commit

If you do not enable the Auto Commit option, you have the option of manually committing or rolling back the transaction. Any transaction not rolled back is committed when the session connection is terminated.

Halting Execution


While SQL-Station Debugger is executing a command, a large Cancel button appears above the query pane. You can click **Cancel** at any time to halt execution. SQL-Station Debugger then displays the results to the point where execution was stopped.

Display Options


Choosing **Edit, Preferences** from the main menu allows you to enable two options: **Show Plan** and **Show DBMS Output**.

Show Plan

Choosing Show Plan causes SQL-Station Debugger to display information about how the server is executing the code in the SQL window. You gain insight into what tables are being hit and what the server is doing. The information is displayed in a separate pane just below the Result pane.




To display this information choose **Show, Plan** or click the Show Plan  toolbar button before running the code in the Edit Window.

Displaying DBMS Output

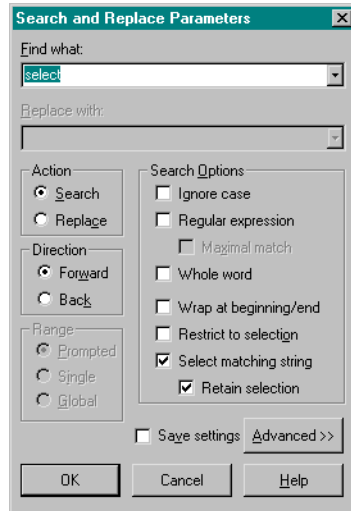
If your SQL code includes DBMS_OUTPUT commands, their content can be viewed after running code by clicking the DBMS  toolbar button or by choosing **Show DBMS Output** from the Preference dialog. The information is displayed in a separate pane just below the Result pane.

Editing Query Code

You can use the Edit menu or toolbar buttons to perform cut, copy, paste, find, replace, select all, and change case operations on code in each Edit Window. You can cut, copy, and paste text between SQL-Station Debugger Edit Windows, or between any SQL text file and a SQL-Station Debugger Edit Window.

Cut text	Highlight the text, click Cut  , or choose Edit, Cut
Copy text	Highlight the text, click Copy  , or choose Edit, Copy .
Paste text	Click an insertion point where you want the pasted text to occur, and click Paste  , or choose Edit, Paste .
Select all	Click in a Query pane or in an Echo pane to make it active. Choose Edit, Select All . SQL-Station Debugger highlights all the text in the active pane.

To Find and Replace Text



- 1 Click in a Query pane or in an Echo pane. The Find command operates only in the selected pane.
- 2 Choose **Edit, Find** and type the string to find in the **Find What** field.
- 3 Specify the action as search. Specify the other search criteria options. Click the Advanced button for more advanced search criteria.
- 4 Click **OK**.


To Replace Text

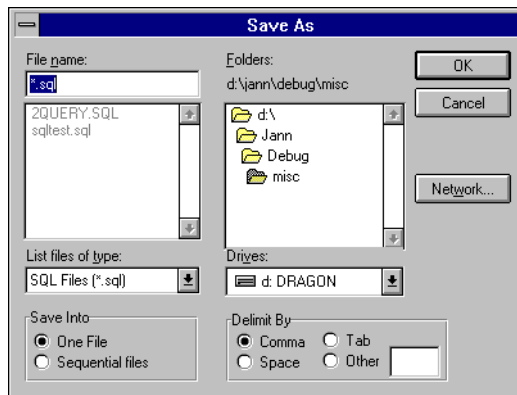
- 1 Click in a Query pane or in an Echo pane. The **Replace** command operates only in the selected pane. Choose **Edit, Find** and choose **Replace** as the action to perform.
- 2 Type the replacement string in the **Replace With** field.
- 3 Specify the other search criteria options.

- 4 Click **OK**. When SQL-Station Debugger highlights an occurrence of the search string, you will be prompted as to whether you want to replace this occurrence. Click **Yes** if you wish to replace it. To replace all occurrences in the active pane, click **Global**.

Saving Text

You can save the results of a SQL query.

- 1 Click in any pane of the Edit Window after running a SQL command.
- 2 Click the Save  toolbar button or choose **File, Save As**.
- 3 In the Save As dialog, specify a location and name for the new file.



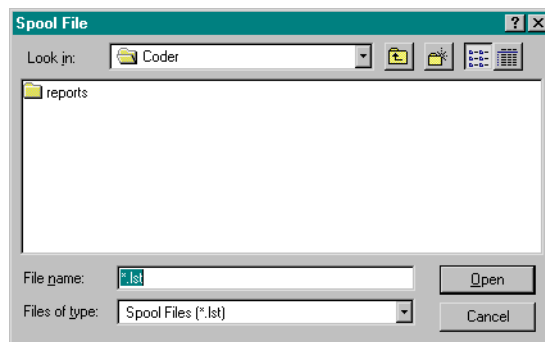
- 4 Choose whether to save the query and results in a single file or in multiple files.
 - To save the original query, all results, and all command echoing (if it's enabled) in a single file, choose **One File**.
 - To save the original query, each output, and each command echo (if enabled) in a separate file, choose the **Sequential Files** option. SQL-Station Debugger adds a 3-digit numbering sequence to the name you supply. For example, if you supply the name ABCDEFGH.TXT, there are two commands in the query, and Command

Echoing is enabled, SQL-Station Debugger creates ABCDEFGH000.TXT through ABCDEFGH004.TXT (one file for the original query, one result file for each of the two commands, and one command echo file for each of the two commands).


- 5 Select the delimiter to be used for separating the result columns: Click the appropriate radio button to use a comma, space, or tab. If you want a different character, choose **Other** and type the character in the adjacent field.
- 6 Click **OK**.

Spooling Sessions to a Log File

SQL-Station Debugger's spooling feature allows you to save all the contents of a SQL Query session. Once you enable spooling, SQL-Station Debugger saves the contents of the Query pane, Result panes, and Echo panes to a designated file. All query work is appended to this log file until you toggle spooling off.



To spool all input, results, and output of a query session to a log file, follow these steps:

- 1 Click the Spooling  toolbar button. SQL-Station Debugger displays the Spool File dialog.
- 2 Specify the directory and filename for the log file.

Spooling Sessions to a Log File

3 Click OK.

SQL-Station Debugger spools all input and output of Edit Window sessions to the specified file.

To Stop Spooling

Click the Spooling  toolbar button again to toggle spooling off.

Object Maintenance

This chapter describes how SQL-Station Debugger uses and names debug versions of server objects and how you can delete them.

About Debug Versions	7-2
Debug Version Preferences	7-2
Deleting Debug Versions	7-3
Trigger Maintenance	7-4
Debug Naming Conventions	7-5
Debug Object Status	7-6
Session Maintenance	7-7

About Debug Versions

When you debug a server object, SQL-Station Debugger makes a copy of the object—a *debug version*—and adds information that allows it to return information about its state to the SQL-Station Debugger user interface. Debugger gives these debug versions names beginning with **X#** and places them in the catalog. See the section *Debug Naming Conventions* later in this chapter for a complete description of naming conventions for debug versions.

When you debug an object, SQL-Station Debugger first checks to see if a debug version of that object already exists. If it finds one, it compares the datetime stamp of the debug version with that of the release version. If the debug version is current, SQL-Station Debugger uses it, saving the overhead of recreating it. If the debug version is out of date, SQL-Station Debugger deletes it and creates a new debug version from the current server object.

Debug Version Preferences

You can choose whether debug versions of objects remain on the server after a debug version ends. Set the default behavior by choosing **Edit, Preferences** and setting the **Clear debug objects after session** in the Debugger tab of the Preferences dialog. The **Debug** menu item is available only when a debugging window is active.

If you choose to leave debug versions on the server, you can easily get rid of out-of-date versions by choosing **Maintenance, Debug Objects**. See *Deleting Debug Versions* in this chapter for more about using the maintenance option.

Note • TRIGGERS—The instructions in the *Deleting Debug Versions* section apply only to procedures and functions. Triggers must be handled differently due to Oracle constraints. See the *Trigger Maintenance* section for information on managing trigger debug versions.

Deleting Debug Versions

To automatically remove out-of-date debug versions and optionally remove those that are still in sync, follow these steps:

- 1 Login as the owner whose debug objects you wish to delete. Be sure that a session for that owner is selected in the **Connection** list box. If you want to delete objects for more than one owner, you must log in as an account with full administrative privileges.
- 2 Choose **Maintenance, Debug Objects** from the main menu.

If you are logged in as a DBA, SQL-Station Debugger removes all out-of-date debug versions. If you are logged in as a user who is not a DBA, SQL-Station Debugger removes out-of-date debug versions belonging to the user name. It then displays the Debug Object Cleanup dialog, in which you can choose to remove debug objects that are not out-of-date.
- 3 Set **Min. Days on Server** to *n* to show only objects that have been on the server at least *n* days.
- 4 *To see information about an object*, click on the original object name in the **Object Names** field. The debug version name, modification date, and status display at the bottom of the dialog. If you have more than one object selected, this information is not displayed.
- 5 **To delete objects for a specific owner**, choose that owner from the **Object Owner** list box. If you are not logged in as a DBA, your current login is the only name available in the **Object Owner** list box. If you are logged in as a DBA, all owners are listed.
 - To delete *all* debug version objects for the selected owner, click **Delete All Types**. This deletes all debug versions owned by **Object Owner**, regardless of type or age.
 - To delete all debug version objects *of one type* for the selected owner, choose the object type from the **Type** list box and then choose **Delete All [type]** (where *type* is a selected type, such as functions or procedures).

- To delete all objects of a type for the selected owner that are n days or older, set **Minimum Days on Server** to n , then select all the objects that are listed in the **Object Names** field and choose **Delete Selected**.
 - To delete selected objects for the selected owner, choose the object type from the **Type of Object** list box and then click the first object that you want to delete in the **Object Names** list. Control-click subsequent objects to select them. When your selection list is complete, click **Delete Selected**.
- 6 When you have completed your cleanup, click **Close** to exit from the Debug Object Cleanup dialog.

Trigger Maintenance

Because Oracle does not allow two triggers of the same type on a table in versions prior to 7.2, SQL-Station Debugger cannot create a debug version of a trigger by copying it. Instead, Debugger stores the release version of the trigger in a table and deploys a debug version.

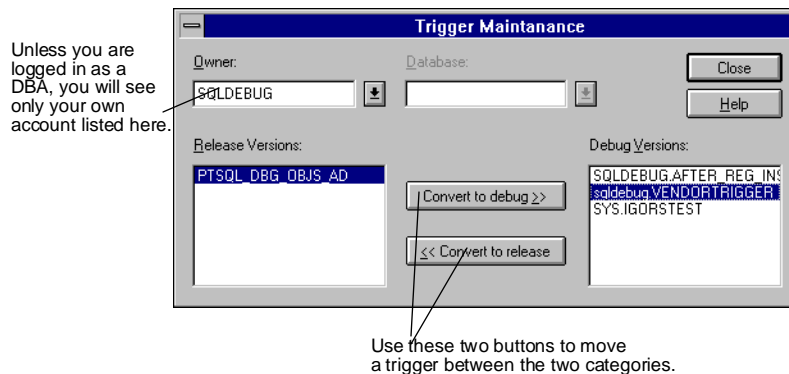
Debugger maintains the correct trigger version transparently, but if you experience a server or client-side crash, you may need to manually return the release version to the server.

To toggle a trigger between its debug version and its release version, follow these steps:

- 1 Click the Connect toolbar button or choose File, Connect and login as the owner whose triggers you wish to change. Be sure that session is selected in the **Connection** list box. If you want to maintain triggers for more than one owner, you must log in as an account with administrative privileges.
- 2 Choose **Maintenance, Trigger** from the main menu. This option will only be enabled if your Oracle version is lower than 7.2.

SQL-Station Debugger displays the Trigger Maintenance dialog. If you are connected to an Oracle server, the **Database** field is grayed out.

The **Owner** field displays multiple owners only if the selected connection is a DBA.



- 3 To change one or more triggers from debug versions to release versions, select them in the **Debug Versions** list and click **Convert to release**.
- 4 To change one or more triggers from release versions to debug versions, select them in the **Release Versions** list and click **Convert to debug**.

SQL-Station Debugger renames each trigger appropriately. In the case of release names that have been truncated to accommodate the debug version prefix characters, the full release names are properly restored.

Debug Naming Conventions

SQL-Station Debugger never operates on the release version of an object. When you debug an object, it performs the following steps:

- Makes a copy of the object.
- Inserts code which allows the object to keep SQL-Station Debugger informed of its state.

Debug Object Status

- Prepends several characters to the name of the original object to create a name for the debug version of the object.

The naming convention is slightly different for server objects and file objects.

Server object	26 characters or less	X#Z\$ plus original name
Server object	27 to 30 characters	X#Z\$n
File object	26 characters or less	X#X\$ plus the original name
File object	27 to 30 characters	X#X\$n

n is a sequence number ranging from 1 to 10²⁷–1.

Debug Object Status

SQL-Station Debugger provides a number of statistics about debug versions that are present in the catalog. To assess the validity of an object, SQL-Station Debugger first compares the datetime stamp of the debug version with that of the original object. If the debug version is older, the status is **INVALID**. If the debug version is current, SQL-Station Debugger performs a further check to see if all child modules are present and valid. Only if the object passes all of these tests is it listed as **VALID** in the Debug Object Information dialog.

- To display the Debut Object Information dialog, choose **Status, Debug Object**.

SQL-Station Debugger displays the Debug Object Information dialog. The names in the **Release Name** field are the original object names. The debug name is in the far right column. The statistics in the Debug Object Information dialog all apply to the debug versions, not the original objects. The **Source** column tells whether the original object was a server object or a file object.

Debug Object Information							
	Owner	Release Name	Type	Modified	Status	Source	Debug Name
1	AC	INS_INVOICE	PACKAGE	08-MAY-96	VALID	Catalog	X#Z\$INS_INVOICE
2	AC	SEL_INVOICE	PACKAGE	02-NOV-96	VALID	Catalog	X#Z\$SEL_INVOICE
3	AC	INS_INVOICE	PACKAGE	08-MAY-96	INVALID	Catalog	X#Z\$INS_INVOICE
4	AC	SEL_INVOICE	PACKAGE	02-NOV-96	INVALID	Catalog	X#Z\$SEL_INVOICE
5	AC	AC_JUNK	PROCEDU	01-NOV-96	INVALID	Catalog	X#Z\$AC_JUNK
6	BURNER	STUDENTSTAT	PACKAGE	04-DEC-96	VALID	Catalog	X#Z\$STUDENTSTAT
7	BURNER	STUDENTSTAT	PACKAGE	04-DEC-96	VALID	Catalog	X#Z\$STUDENTSTAT
8	BURNER	ANALYZESTUD	PROCEDU	03-DEC-96	VALID	Catalog	X#Z\$ANALYZESTUDE
9	BURNER	COURSEGENDE	PROCEDU	04-DEC-96	VALID	Catalog	X#Z\$COURSEGENDEF
10	BURNER	AFTER_REG_IN	TRIGGER	04-DEC-96	VALID	Catalog	X#Z\$AFTER_REG_INS
11	DANDBG	BANZHEF2	FUNCTION	22-NOV-96	VALID	Catalog	X#Z\$BANZHEF2
12	DANDBG	CHECKGENDEF	FUNCTION	22-NOV-96	VALID	Catalog	X#Z\$CHECKGENDERE
13	DANDBG	COMPARESALE	FUNCTION	22-NOV-96	VALID	Catalog	X#Z\$COMPARESALES
14	DANDBG	VERIFYUSEASN	FUNCTION	22-NOV-96	VALID	Catalog	X#Z\$VERIFYUSEASN
15	DANDBG	JUNKPACK	PACKAGE	03-DEC-96	VALID	Catalog	X#Z\$JUNKPACK
16	DANDBG	SALES_ANALYS	PACKAGE	22-NOV-96	VALID	Catalog	X#Z\$SALES_ANALYS

Session Maintenance

To see a listing of all sessions that are running on the current server, choose **Maintenance, Server Sessions**. The user for the session that is currently selected in the **Connections** list box of the Catalog Browser can then kill any session for which he has `ALTER SYSTEM` privileges.

Server Sessions						
	Owner	Type	User	Program	Lock Wait	Machine
1	SYS	USER	CONNT	OraPgm	NULL	TECHWRITE
2	JEFF	USER	burkh	coder.exe	NULL	RAT
3	HARVEST	USER	Administrator	CCC Harvest Ser	NULL	SUNDOG
4	SQLCODER	USER	BURKD	OraPgm	NULL	MONSTER
5	TEST	USER	weimi	SQL-Station Cod	NULL	HAWK
6	SYS	USER	CONNT	OraPgm	NULL	TECHWRITE
7	TEST	USER	SYSTEM	Purity'd E:\dev\h	NULL	TRIBBLE
8	JEFF	USER	burkh	coder.exe	NULL	RAT

Tips and Troubleshooting

This chapter contains hints and tips for using SQL-Station Debugger to solve real-world debugging problems.

Background Information	8-2
Tips and Techniques	8-2
Granting Privileges on Debug Versions	8-2
Putting an Object on the Exclusion List	8-3
Debug Version Naming and Maintenance	8-3
Advantages of Debugging File Objects	8-3
Disadvantages of Debugging File Objects	8-4
Hung Sessions	8-4
Locking Issues	8-5
Trigger Issues	8-5
Overloaded Objects	8-6

Background Information

When you debug a database object, SQL-Station Debugger generates a debugging version of the release object by inserting the additional code. This additional code enables the debug version of the code to communicate with the user interface when it is running. It can, for example, return information such as variable values. The original object—the release version—remains unaffected as you work with the debug version.

Tips and Techniques

Granting Privileges on Debug Versions

For database programmers to be able to work effectively with the debugger, they may need support from the DBA or from another user who has administrative privileges in the database. To debug objects in a given schema, the user must have `CREATE ANY PROCEDURE` and `DROP ANY PROCEDURE` privileges on that schema.

When a user invokes debugging on an object, SQL-Station Debugger tries to create debug versions of the procedure being debugged including all the dependent procedures. For each dependent procedure, the debugger creates the debug version only if all the following conditions are true: 1) There is no debug version in the catalog or the debug version has an earlier timestamp than the production version; 2) The object is not on the exclusion list; and 3) The user has `CREATE` privileges for the object.

The third condition is very restrictive in most environments, since programmers don't typically have `CREATE` privileges on other users' schemas. If a programmer tries to debug a procedure that references a procedure on which he has insufficient privileges, the debug version creation fails. For such cases, the DBA can create debug versions of the procedures and grant the necessary privileges on them to the programmers. Opening a procedure in the Debugger is sufficient to create the debug version.

Putting an Object on the Exclusion List

The DBA should use the Global Exclusion dialog to exclude procedures which he doesn't want programmers to debug. This Global Exclusion List allows the DBA control over object code.

Oracle security allows any user who has `EXECUTE ANY` privilege to see any object text except a package body through the `ALL_SOURCE` catalog view. However, the package body text is not accessible to non-DBA users by default. This limits debugging in the sense that programmers won't be able to step into package procedures, since the text is not available.

SQL-Station Debugger obtains package body text from an alternative view to enable debugging. If this is not desired, the DBA can specifically exclude any package by using the Global Exclusion List.

Debug Version Naming and Maintenance

The debug version of a stored object is named `X#Z$` followed by the original object name of up to 26 characters. File object names begin with `X#X$` followed by the original object name of up to 26 characters.

The prefix is obscure to minimize the chance of conflicting with names that may already be in use. If a shop uses names starting with this prefix, debugging objects in that schema is not recommended.

Objects Whose Length Is Greater than 26 Characters

For objects whose object name length is greater than 26 characters, the debugger replaces the original name with a unique sequence of characters when it names the debug version. This avoids overflowing the Oracle 30-character limit on object name length.

Advantages of Debugging File Objects

- Objects can be selected for debugging either from the Catalog Browser or from a file system. Selecting the file version of an object can sometimes be faster than selecting the catalog version. When you

debug an object from the file system, only its debug version is deployed.

- File objects have some restrictions and some advantages. Since SQL-Station Debugger views file objects as independent objects, it does no dependency analysis and does not create debug versions of dependent objects, although those objects must be present in the catalog. File object debugging can be very useful in cases where the problem object has been isolated and you do not need to step into dependent modules. Since there is no dependency analysis for file objects and no creation of debug versions for dependent procedures, this option may provide a huge performance advantage over debugging the corresponding catalog object, especially if the number of nodes in the dependency tree is large and there are many programmers modifying objects in the database.

Disadvantages of Debugging File Objects

- SQL-Station Debugger cannot generate debug versions of a file object if the release versions of referenced modules are not in the catalog.
- SQL-Station Debugger ignores any catalog object of the same name when generating the debug version of a file object. You must bear in mind that if there is a server object with the same name as the file object you are debugging, the server object is not affected. There is also no guarantee that the file object and the server object are the same in more than name.
- Triggers are not supported as file objects.

Hung Sessions

In client-server environments, there is always the question of how a product handles client side reboots when the product is active and there is a server-side problem. For SQL-Station Debugger, the issue of client-side reboots is even more a potential problem, since there can be two or more sessions associated with any debug window. SQL-Station Debugger handles this situation very cleanly: The debug procedure running on the

server has a timeout programmed into it, settable through the Preferences dialog. If the session hangs on the client side, the debug procedure runs to completion on the server after the specified timeout period. However, if your SQL*Net version is prior to 2.2, Oracle never removes hung sessions from its session queue. Your DBA may need to clean up these sessions manually. See the *Session Maintenance* section in the *Object Maintenance* chapter for more information about ending hung sessions. Server-side problems are handled through a client timeout, settable from the Preferences dialog.

Locking Issues

Oracle code object locking is very efficient in the sense that it locks code objects when execution references them and unlocks them when procedure execution exits out of them. What this means is that executing a particular code object does not restrict other users from modifying/dropping the dependent objects which are not currently being executed. Oracle locks them only when the dependent object is executed. However, the following scenario may cause a problem in the debugger: The object that is being debugged becomes invalidated if one of the dependent procedures is altered or modified while Oracle is running the object. However, Oracle detects the modification/drop only when the object actually references the dependent object. Oracle shuts down execution of the object with Oracle error 4605. SQL-Station Debugger handles this problem cleanly by detecting debug version modifications and warning the user.

Trigger Issues

In versions prior to 7.1, Oracle does not allow two triggers of the same type on the same table. Because of this Oracle issue, SQL-Station Debugger implements the following workaround: When debugging is invoked on a trigger, the release version of the trigger is temporarily taken out of the catalog and stored in a table. SQL-Station Debugger then deploys the debug version. After debugging, the trigger is switched back to the release version. Because of this switching, triggers should always be debugged in a development environment rather than in a production

environment. A potential consequence of this design is that debug versions of the trigger may remain in the catalog if there is a client or server side reboot. The trigger maintenance dialog is provided to handle this situation. The dialog allows you to switch between release and debug versions of a trigger. A trigger debug version behaves exactly like the release version if it is fired outside of the debugger, so data is not affected if debug triggers persist. See also the *Trigger Maintenance* section of the *Object Maintenance* chapter.

Overloaded Objects

Overloaded objects in Oracle packages are handled directly through the debugger catalog browser. The catalog browser is powerful enough to distinguish between overloaded versions of the same object.



Index

A

array inputs 4-11

B

breakpoints

- continuing to next 5-20

- cycling 5-14

- disabling 5-13

- reactivating 5-13

- removing 5-14

- setting 5-12

C

Call Tree 2-3

case, changing 6-10

Catalog Browser 4-3

- contents of 4-3–4-8

- displaying 4-2

- displaying tables in 4-15

- features 2-3

- filtering the display 4-9

- function and procedure names 4-7

Navigator 4-4

- object components 4-5

- Object Hierarchy 4-4

- object types in 4-5

- overview 2-10

- sorting the display 4-8

child module

- stepping into 5-20

- stepping out of 5-22

- stepping over 5-21

code

- displaying 4-16, 6-3

- editing 6-10

- running 6-7

Command Echoing 6-7

committing transactions 4-14, 6-8

connecting to a server 2-5, 2-6

copy text 6-10

cut text 6-10

D

Data icon 4-15



- DBMS Output, displaying 6-9
 - Debug Objects (Maintenance menu) 7-2
 - debug versions 4-8
 - about 7-2
 - deleting 7-3
 - naming conventions 7-5
 - preferences 7-2
 - status 7-6
 - Debugger
 - begin session 5-14
 - entering parameters 5-15
 - features 2-3, 5-3
 - illustration 5-4
 - preferences 2-16, 5-5
 - settings 5-5
 - Step Into 5-20
 - Step Out Of 5-22
 - Step Over 5-21
 - tracking execution 5-20, 5-21
 - using 5-3–5-18
 - viewing variables 5-17
 - Watch pane 5-17
 - debugging 5-4–5-18
 - breakpoints 5-11–5-14
 - continue to next breakpoint 5-20
 - debug versions 4-8, 7-2
 - description 2-4
 - ending a session 5-22
 - entering parameters 5-15
 - excluding modules from 5-8, 5-10
 - file objects 5-3, 5-7
 - overview 2-11
 - packages 5-6
 - server objects 5-3, 5-7
 - session maintenance 7-7
 - starting a session 5-14
 - Step Into 5-20
 - Step Out Of 5-22
 - Step Over 5-21
 - steps in 5-7
 - default parameters 4-10
 - deleting
 - debug versions 7-3
 - disabling breakpoints 5-13
 - displaying
 - DBMS output 6-9
 - Edit window 6-3
 - object code in an Edit window 4-16
 - tables 4-15
 - dropping objects 4-16
- E**
- Edit window 6-11
 - accessing from Browser 4-16
 - change case of selected text 6-10
 - command echoing 6-7
 - displaying 6-3
 - displaying code in 6-3
 - displaying DBMS output 6-9
 - editing code 6-10
 - features 2-4
 - halting execution 6-8
 - multiple windows 6-4
 - overview 2-13
 - preferences 2-18, 6-5
 - replacing text 6-11
 - result display options 6-6
 - running code 6-7
 - saving text 6-12

- settings 6-5
- Show Plan 6-9
- spooling sessions 6-13
- editing SQL code 6-4, 6-10
- evaluating expressions 5-18
- excluding modules from debugging 5-8, 5-10
- executing
 - committing after 4-14
 - overview 2-13
 - server objects 4-10–4-15
 - SQL code 2-13
- execution, server-side 2-4
- expressions, evaluating 5-18

F

- file
 - debugging 5-7
 - displaying in Edit window 6-4
- filtering in Catalog Browser 4-9
- filtering names in Catalog Browser 4-9
- finding text 6-11
- functions
 - debugging 2-11, 5-3–5-18
 - displaying code 4-16, 6-3
 - dropping 4-16
 - excluding from debugging 5-11
 - names 4-7

G

- Global Exclusion List 5-10
- group logins 2-8

H

- halt

- debugging 5-22
- execution of SQL code 6-8

I

- input parameters 4-10
- installing Watch_PL 1-2

L

- login
 - group login 2-8
 - shortcut 2-7
 - starting a session 2-6
- Login Name 2-6

M

- Maintenance
 - Debug Objects 7-2, 7-3
 - Server Sessions 7-7
 - triggers 7-4
- Min. Days on Server 7-3

N

- names
 - debug version names 7-5
 - object names 4-9
- Navigator 4-4

O

- object code
 - displaying 4-16
 - editing 6-10
- object components 4-5
- object information 7-3
- object types 4-5
- objects, dropping 4-16

P

- package header pane 5-6
- package headers, displaying 5-6
- packages
 - displaying in the Debugger 5-6
- parameters
 - boolean 4-10
 - entering 4-10
 - entering in Debugger 5-15
 - PL/SQL table 4-11, 5-16
 - required 4-10
 - supported types 4-12
 - Use Default 4-10
- Paste text 6-10
- PL/SQL table inputs 4-11
- preferences 2-15–??
 - Debugger 2-16, 5-5, 7-2
 - defaults 2-15
 - Edit window 2-18, 6-5
 - general 2-22
 - overrides 2-16
 - procedure execution 2-22
- procedures
 - debugging 2-11, 5-3–5-18
 - displaying code 4-16, 6-3
 - dropping 4-16
 - excluding from debugging 5-11
 - names 4-7

R

- reactivating breakpoints 5-13
- removing breakpoints 5-14
- replacing text 6-11
- restart debugging 5-22
- result display options 6-6

- running SQL code 6-7

S

- saving text 6-12
- server objects
 - committing after execution 4-14
 - debugging 5-3, 5-7
 - displaying code 6-3
 - dropping 4-16
 - excluding from debugging 5-10
 - executing 4-10–4-15
- Server Type 2-6
- server, connecting to 2-5
- session
 - ending 2-15
 - starting 2-6
- setting breakpoints 5-12
- Show Plan 6-9
- Sort By field 4-8
- Source column 7-6
- spooling Edit window sessions 6-13
- SQL code
 - editing 6-4
 - executing 2-13
- SQL text files
 - debugging 5-4
 - running 6-7
- status of debug objects 7-6
- stopping a debug session 5-22

T

- tables
 - as input parameters 4-11
 - displaying 4-15
- text

- copying 6-10
- cutting 6-10
- editing 6-10
- finding 6-11
- replacing 6-11
- saving 6-12
- triggers
 - debug versions 7-4
 - deleting 7-2
- tutorial 3-1

V

- variables
 - changing values 5-18
 - in debugging 2-12
 - viewing in the Watch pane 5-17

W

- Watch pane 2-4, 5-18
- Watch_PL overview 2-10–2-14

