

Using Rational Robot

Version 2000.02.10

Using Rational Robot

Copyright © 1998-2000 Rational Software Corporation. All rights reserved. The contents of this manual and the associated software are the property of Rational Software Corporation and are copyrighted. Any reproduction in whole or in part is strictly prohibited. For additional copies of this manual or software, please contact Rational Software Corporation.

Rational, the Rational logo, PerformanceStudio, SiteCheck, TestFactory, TestStudio, Object-Oriented Recording, and Object Testing are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other names are used for identification purposes only and are trademarks or registered trademarks of their respective companies.

U.S. GOVERNMENT RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational License Agreement and in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct 1988), FAR 12.212(a) 1995, FAR 52.227-19, or FAR 52.227-14, as applicable.

Revised 04/2000

This manual prepared by:
Rational Software Corporation
20 Maguire Road
Lexington, MA 02421
U.S.A.

Phone:
800-433-5444
408-863-4000

E-mail: support@rational.com
Web: <http://www.rational.com>

P/N 800-023366-000

▶ ▶ ▶ Contents

Preface

Resources	xxi
U sing Help	xxii
Contacting Rational Technical Publications	xxii
Contacting Rational Technical Support	xxiii

Part I Introducing Rational Robot

1 Introduction to Rational Robot

What Is Rational Robot?	1-1
Managing the Rational Repository with the Administrator . . .	1-2
Planning and Managing Tests in TestManager	1-3
Developing Tests in Robot	1-5
Creating Datapools	1-7
Analyzing Results in the LogViewer and Comparators	1-8
Managing Intranet and Web Sites with SiteCheck and Robot	1-10
U sing Robot with Other Rational Products	1-12
Testing Applications with Rational TestFactory	1-12
Managing Defects with Rational ClearQuest	1-12
Collecting Diagnostic Information During Playback	1-13
Performance Testing with Rational Suite PerformanceStudio	1-13
Managing Requirements with Rational RequisitePro	1-14
Sharing Data with Other Rational Products	1-14
Starting Robot and Its Components	1-15
Logging in	1-15
Changing to Another Project	1-16
Opening Other Rational Products and Components	1-16
Tasks You Can Perform with Robot and Its Components	1-17

Part II Planning Tests

2 Planning Your Tests

Working with Test Plans and Other Test Documents	2-1
Creating Test Document References	2-2
Editing Test Document References	2-2
Viewing Test Documents	2-3
Defining Test Requirements.	2-3
Working with Projects	2-4
Building the Requirements Hierarchy	2-5
Planning Scripts	2-10
About Scripts.	2-10
Assigning a Name and Other Properties.	2-11
Attaching Scripts to Test Requirements	2-13
Referencing Specification Files	2-15
Customizing Script Properties.	2-16
Displaying Script Statistics.	2-16
Referencing Include Files and External C Libraries.	2-16
Controlling Whether Robot Starts After Planning a Script	2-17
Deleting Scripts	2-18
Importing Scripts from Other Projects.	2-18
Planning LoadTest Schedules.	2-19
Attaching Schedules to Test Requirements	2-19
Referencing Specification Files	2-20
Customizing Schedule Properties.	2-21
Displaying Schedule Statistics	2-21
Deleting Schedules.	2-21
Managing Requirements.	2-21
Editing Requirement Properties	2-21
Deleting Requirements	2-22
Customizing Scripts and LoadTest Schedules.	2-22
Customizing the Environment List.	2-22
Customizing the Purpose List	2-23
Defining Custom Field Labels and Values	2-23

3	Managing Builds, Log Folders, and Logs	
	Overview	3-1
	Using Builds in Functional Testing	3-2
	Generating Log Files	3-2
	Organizing Log Folders	3-3
	Naming Log Folders for LoadTest Users	3-3
	Displaying Builds in the Asset Browser	3-4
	Creating a New Build	3-5
	Copying, Renaming, and Deleting Builds	3-6
	Renaming and Deleting Logs and Log Folders	3-8
	Displaying Log Properties	3-9
	Viewing Logs	3-9
	Working with Build States	3-9

Part III

Developing Tests

4	Recording GUI Scripts	
	The Recording Process	4-1
	The Recording Workflow	4-2
	Before You Begin Recording	4-3
	Establishing Predictable Start and End States for Scripts	4-3
	Setting Up Your Test Environment	4-3
	Creating Modular Scripts	4-3
	Planning Scripts in TestManager	4-4
	Enabling IDE Applications for Testing	4-5
	Setting GUI Recording Options	4-6
	Naming Scripts Automatically	4-7
	Controlling How Robot Responds to Unknown Objects	4-8
	Selecting an Object Order Preference	4-10
	Using Advanced Features Before Recording	4-11
	Customizing the Object Recognition Method Order	4-12
	Mapping Object Types and Classes Before Recording	4-14

- Recording a New GUI Script4-16
 - Restoring the Robot Main Window During Recording.4-19
 - Using the GUI Record and GUI Insert Toolbars4-20
 - Pausing and Resuming the Recording of a Script4-20
 - Defining Unknown Objects During Recording4-21
 - Switching to Low-Level Recording.4-22
 - Ending the Recording of a GUI Script4-23
- Defining Script Properties.4-24
- Coding a GUI Script Manually4-24
- Testing Your Recorded Script.4-25
 - Playing Back the Script.4-25
 - Editing and Compiling the Script4-25
 - Debugging the Script4-26
- Creating Shell Scripts to Play Back Scripts in Sequence.4-26
 - Creating a Shell Script4-26
 - Playing Back a Shell Script.4-27

5 Adding Features to GUI Scripts

- Starting an Application5-1
- Inserting a Call to Another Script5-3
- Inserting Verification Points5-4
- Inserting Timers5-6
 - Uses for Timers5-6
 - Inserting a Timer5-7
 - Playing Back a Script that Includes Timers5-8
- Inserting Comments5-8
- Inserting Log Messages5-9
- Inserting Delay Values.5-10
- Using the Insert Menu5-11
- Customizing SQABasic Scripts5-11
 - Library Source Files5-12
 - SQABasic Header Files5-14
 - Header and Library Source File Examples5-15
 - The Template File5-16

6	Creating Verification Points in GUI Scripts	
	About Verification Points	6-1
	Verification Points and Data Files	6-1
	Verification Points and Scripts	6-2
	Types of Verification Points	6-3
	Before You Create a Verification Point	6-6
	Tasks Associated with Creating a Verification Point	6-6
	Starting to Create a Verification Point	6-7
	Setting a Wait State for a Verification Point	6-8
	Setting the Expected Result for a Verification Point	6-9
	Selecting and Identifying the Object to Test	6-10
	Selecting a Verification Method	6-14
	Selecting an Identification Method	6-15
	Working with the Data in Data Grids	6-19
	Selecting the Data to Test in a Data Grid	6-19
	Testing Column Titles or Top Menus in a Data Grid	6-20
	Editing Captured Data in a Data Grid	6-21
	Changing a Column Width in a Data Grid	6-22
	Transposing Columns and Rows in a Data Grid	6-23
	Editing a Verification Point	6-23
	Viewing a Baseline File	6-24
	Renaming a Verification Point	6-25
	Copying a Verification Point	6-25
	Deleting a Verification Point	6-26
7	Editing, Compiling, and Debugging Scripts	
	Editing the Text of a Script	7-1
	Adding a User Action to an Existing GUI Script	7-2
	Adding a Feature to an Existing GUI Script	7-2
	Working with Low-Level Scripts	7-3
	Viewing Low-Level Scripts	7-4
	Renaming a Low-Level Script	7-4
	Copying a Low-Level Script	7-5
	Deleting a Low-Level Script	7-6
	Saving Scripts and SQABasic Files	7-7

Printing a Script or SQ ABasic File7-7

Compiling Scripts and SQ ABasic Library Source Files7-7

 Compiling One or All Scripts and Library Source Files7-8

 Batch Compiling Scripts and Library Source Files.7-8

 Locating Compilation Errors.7-9

Debugging GUI Scripts.7-9

 Setting and Clearing Breakpoints7-11

 Executing to a Selected Line7-13

 Executing in Animation Mode7-13

 Examining Variable Values.7-13

Deleting Scripts7-15

8 Working with Datapools

What Is a Datapool?8-2

 Datapool Tools8-2

 Datapool Cursor8-3

 Datapool Limits8-3

 What Kinds of Problems Does a Datapool Solve?8-4

Planning and Creating a Datapool8-4

Data Types8-6

 Standard and User-Defined Data Types.8-7

 Finding Out What Data Types You Need8-8

 Creating User-Defined Data Types8-9

 Generating Unique Values from User-Defined Data Types. .8-10

 Generating Multi-Byte Characters8-11

Using Datapools with GUI Scripts8-12

 Recording a GUI Script8-12

 Adding Datapool Commands to a GUI Script.8-13

 Example GUI Script.8-16

Managing Datapools8-17

 Creating a Datapool with TestManager8-18

 Editing Datapool Column Definitions with TestManager . .8-25

 Editing Datapool Values with TestManager8-26

 Renaming a Datapool.8-27

 Copying a Datapool8-27

Deleting a Datapool	8-28
Importing a Datapool	8-28
Exporting a Datapool	8-30
Managing Data Types	8-31
Editing User-Defined Data Type Values	8-31
Editing Standard Data Type Values	8-32
Editing User-Defined Data Type Definitions	8-32
Generating Values for a User-Defined Data Type	8-33
Importing a User-Defined Data Type	8-35
Renaming a User-Defined Data Type	8-35
Copying a User-Defined Data Type	8-36
Deleting a User-Defined Data Type	8-36
Generating and Retrieving Unique Datapool Rows	8-36
What You Can Do to Guarantee Unique Row Retrieval ...	8-37
Creating a Datapool Outside Rational Test	8-38
Datapool Structure	8-39
Example Using Microsoft Excel	8-40
Matching Datapool Columns with Script Variables	8-42
Maximum Number of Imported Columns	8-42
Creating a Column of Values Outside Rational Test	8-42
Step 1. Create the File	8-43
Step 2. Assign the File's Values to the Datapool Column ...	8-43
Generating Unique Values	8-44

Part IV

Playing Back Scripts and Analyzing Results

9 Playing Back GUI Scripts

Playback Phases	9-1
Test Development Phase	9-2
Regression Testing Phase	9-2
Restoring the Test Environment Before Playback	9-3
Setting GUI Playback Options	9-4
Acknowledging the Results of Verification Point Playback ...	9-5
Setting Log Options for Playback	9-5
Setting Wait State and Delay Options	9-7

- Setting Error Recovery Options9-9
- Setting Unexpected Active Window Options..... 9-10
- Setting Diagnostic Tools Options.....9-11
- Setting the Trap Options to Detect GPFs..... 9-16
- Playing Back a GUI Script9-18
- Viewing Results in the Rational LogViewer..... 9-20
- Analyzing Verification Point Results with the Comparators9-21

10 Reviewing Logs with the LogViewer

- Overview10-2
 - Usage Scenarios10-2
- Starting the LogViewer10-3
 - Starting the LogViewer Automatically from Robot10-3
 - Starting the LogViewer Automatically from LoadTest.....10-3
 - Starting the LogViewer from TestManager10-4
 - Starting the LogViewer from a Rational Test Product.....10-4
 - Starting the LogViewer from the Desktop10-4
- The LogViewer Main Window.....10-5
- Opening a Log File10-6
- Deleting a Log File.....10-6
- Viewing Log Event Properties.....10-7
- Modifying the Log Window10-7
 - Collapsing and Expanding Log Events10-7
 - Changing Column Widths.....10-8
 - Changing the Column Order10-8
- Locating Failed Log Events10-8
- Evaluating Verification Point Failures in a Comparator10-9
 - Viewing a Verification Point in the Comparators.....10-9
 - Viewing a Script10-10
 - Playback/Environmental Differences10-10
 - Intentional Changes to an Application Build10-10
- Filtering the Log Event Column.....10-11
 - Applying a Log Filter10-11
 - Creating or Editing a Log Filter.....10-11
 - Copying, Renaming, and Deleting a Log Filter10-12

Working with Reports	10-13
Setting a Default Report Layout.	10-13
Generating, Printing, and Saving a Quick Report	10-13
Entering and Modifying Defects.	10-14
About ClearQuest and Defect Tracking	10-15
Starting ClearQuest	10-18
Entering Defects	10-18
Finding Defects.	10-21
Modifying Defects	10-21
11 Using the Object Properties Comparator	
Overview	11-1
Starting the Object Properties Comparator	11-2
Starting the Comparator from Robot	11-2
Starting the Comparator from the LogViewer	11-2
The Main Window.	11-3
The Objects Hierarchy and the Properties List	11-4
Changing the Window Focus and Section Widths.	11-4
Working Within the Objects Hierarchy	11-5
Working Within the Properties List.	11-6
Locating and Comparing Differences.	11-6
Viewing Verification Point Properties	11-7
Adding and Removing Properties	11-7
Adding Properties to the Properties List	11-7
Removing Properties from the Properties List	11-8
Editing the Baseline File	11-9
Editing a Value in the Properties List	11-9
Cutting, Copying, and Pasting a Value	11-10
Copying Values from the Actual to the Baseline File	11-11
Changing a Verification Method	11-11
Changing an Identification Method	11-12
Replacing the Baseline File.	11-12
Saving the Baseline File	11-12

12 Using the Text Comparator

Overview	12-1
Starting the Text Comparator	12-2
Starting the Comparator from Robot	12-2
Starting the Comparator from the LogViewer	12-3
The Main Window	12-3
The Text Window	12-4
Scrolling the Text Window	12-4
Changing the Widths of the Text Panes	12-4
Using Word Wrap	12-4
Locating and Comparing Differences	12-5
Viewing Verification Point Properties	12-5
Editing the Baseline File	12-5
Editing Data in the Baseline File	12-6
Cutting, Copying, and Pasting Data	12-6
Copying Data from the Actual to the Baseline File	12-6
Replacing the Baseline File	12-7
Saving the Baseline File	12-7

13 Using the Grid Comparator

Overview	13-1
Starting the Grid Comparator	13-2
Starting the Comparator from Robot	13-2
Starting the Comparator from the LogViewer	13-3
The Main Window	13-3
The Grid Window	13-4
Differences List	13-4
Setting Display Options	13-5
Changing the Column Widths	13-5
Transposing the Grid Data	13-5
Synchronizing the Scroll Bars	13-5
Synchronizing the Cursors	13-6
Locating and Comparing Differences	13-6
Viewing Verification Point Properties	13-7
Using Keys to Compare Data Files	13-7

Editing the Baseline File	13-8
Editing Data in the Baseline Grid	13-8
Editing a Menu Item	13-9
Cutting, Copying, and Pasting Data	13-10
Copying Data from the Actual to the Baseline File	13-10
Replacing the Baseline File	13-11
Saving the Baseline File	13-11
14 Using the Image Comparator	
Overview	14-1
Starting the Image Comparator	14-2
Starting the Comparator from Robot	14-2
Starting the Comparator from the LogViewer	14-3
The Main Window	14-4
The Image Window	14-4
..... Differences List	14-5
Mask/OCR List	14-5
The Status Bar	14-6
Locating and Comparing Differences	14-6
Changing How Differences are Determined	14-7
Changing the Color of Masks, OCR Regions, or Differences ...	14-7
Moving and Zooming An Image	14-8
Viewing Image Properties	14-8
Working with Masks	14-9
Displaying Masks	14-9
Creating Masks	14-10
Moving and Resizing Masks	14-10
Cutting, Copying, and Pasting Masks	14-11
Duplicating Masks	14-11
Deleting Masks	14-12
Automatically Masking a Difference	14-12
Working with OCR Regions	14-13
Creating an OCR Region	14-13
Moving and Resizing OCR Regions	14-14
Cutting, Copying, and Pasting an OCR Region	14-15

Duplicating OCR Regions	14-16
Deleting OCR Regions	14-16
Replacing the Baseline File	14-17
Saving the Baseline File	14-18
Viewing Unexpected Active Window	14-18

Part V Running Queries and Reports

15 Querying the Rational Repository

Overview	15-1
Running Queries	15-2
The Query Window	15-2
Deleting Scripts, Schedules, and Sessions	15-3
Creating New Queries	15-4
Opening the Query Properties Dialog Box	15-4
Choosing Fields to Display	15-5
Specifying the Sort Order	15-5
Adding a Filter Statement	15-6
Editing Existing Queries	15-8
Viewing Query Properties	15-8
Setting Query Options	15-9
Configuring the Query Window	15-10
Replacing or Inserting a Column	15-10
Deleting a Column	15-10

16 Running TestManager Reports

Types of Reports	16-1
Listing Reports	16-2
Coverage Reports	16-2
Progress Reports	16-3
Selecting Which Reports to Use	16-3
Working with Listing Reports	16-4
Creating Listing Reports	16-4
Running Listing Reports	16-5
Opening Listing Reports	16-7

Working with Coverage Reports	16-7
Creating Coverage Reports	16-7
Running Planning and Development Coverage Reports	16-11
Running Execution Coverage Reports	16-13
Opening Coverage Reports	16-14
Working with the Test Results Progress Report	16-15
Creating a Test Results Progress Report	16-15
Running a Test Results Progress Report	16-16
Opening a Test Results Progress Report	16-17
Copying, Renaming, and Deleting Reports	16-17

Part VI Testing IDE Applications

17 Testing Visual Basic Applications

About Robot Support for Visual Basic Applications	17-1
Try it! with Visual Basic	17-3
Verifying that the Visual Basic Extension Is Loaded	17-3

18 Testing Oracle Forms Applications

About Robot Support for Oracle Forms Applications	18-1
Try it! with Oracle Forms	18-2
Making Oracle Forms Applications Testable	18-2
Installing the Rational Test Oracle Forms Enabler	18-3
Running the Enabler on Your Application	18-3
Verifying that the Oracle Forms Extension Is Loaded	18-8
Recording Actions and Testing Objects	18-9
Recording Actions	18-9
Testing Objects	18-9
Testing an Object's Properties	18-12
Object Properties Verification Point	18-12
Object Scripting Commands	18-15
Testing an Object's Data	18-16
Testing Base-Table Blocks and Base-Table Items	18-16
Testing LOVs and Record Groups	18-17

19 Testing HTML Applications

About Robot Support for HTML Applications	19-1
Configuring Internet Explorer for Testing	19-2
Disabling the Cookie Prompt	19-2
Try It! with HTML	19-3
Making HTML Applications Testable	19-3
Verifying that the HTML Extension Is Loaded	19-3
Enabling HTML Testing in Robot	19-3
Testing Data in HTML Elements	19-4
Additional Examples.	19-7
How Robot Maps HTML Elements	19-8
Supported Data Tests for HTML Testing	19-10
Testing Properties of HTML Elements	19-11
Playing Back Scripts in Netscape Navigator	19-12
Configuring Robot for Netscape Playback	19-12
Differences Between Internet Explorer and Navigator	19-13
Recording Tips.	19-14
Capturing the Properties of Java Applets in HTML Pages	19-14
Synchronizing Pages.	19-15
Recording Mouse Movements	19-16
Ensuring Browser Compatibility.	19-17
Enhancing Object Recognition of HTML Elements	19-17

20 Testing Java Applets and Applications

About Robot Support for Java.	20-2
Robot Support for Testing Java Applets and Applications	20-3
Supported Foundation Class Libraries	20-3
Making Java Applets and Applications Testable.	20-4
Running the Java Enabler.	20-5
Verifying that the Java Extension Is Loaded	20-7
Setting Up the Sample Java Applet	20-7
Installing the Sample Java Applet	20-8
Installing the Swing Foundation Classes	20-8
Starting the Sample Java Applet	20-10

Testing Data in Java Components	20-10
Testing the Contents of a Java Panel	20-12
Support for Custom Java Components	20-13
For More Information About Java Support	20-14
Supported Data Tests for Java Testing	20-14
Testing Properties of Java Components	20-15
Enhancing Object Recognition of Java Components	20-17
21 Testing PowerBuilder Applications	
About Robot Support for PowerBuilder Applications	21-1
Verifying that the PowerBuilder Extension Is Loaded	21-2
Try it! with PowerBuilder	21-2
Recording Actions on DataWindows	21-3
Parameters for a Mouse-Click Action	21-4
Value-Based Recording	21-4
Testing an Expression Value of a DataWindow Property	21-5
Testing DataStore Controls and Hidden DataWindows	21-6
Capturing Data in a DropDownDataWindow/ListBox	21-7
Testing the Value of a DataWindow Computed Field	21-8
22 Testing PeopleTools Applications	
About Robot Support for PeopleTools Applications	22-1
Verifying that the PeopleTools Extension Is Loaded	22-2
Testing a Component's Properties	22-2
Testing a Component's Data	22-3
PeopleTools Commands	22-3

Part VII **Appendixes**

A Working With Toolbars

Viewing Information About Toolbar Buttons	A-1
Displaying Toolbars.	A-2
Anchoring and Floating Toolbars	A-2
Setting Toolbar Options	A-3
Adding, Deleting, and Moving Toolbar Buttons.	A-3
Creating Your Own Toolbar.	A-3
Resetting and Deleting Toolbars	A-4

B Working with Data Tests

About Data Tests	B-1
An Example of a Data Test	B-2
What the All Data Test Does	B-2
The Definition of the All Data Test	B-3
Changing a Data Test Definition	B-4
Creating or Editing a Custom Data Test	B-5
Copying, Renaming, or Deleting a Data Test	B-8

C Standard Datapool Data Types

Standard Data Type Table.	C-1
Data Type Ranges	C-9

D Rational Robot Command-line Options

E Working with Manual and External Scripts

About Manual Scripting	E-1
Working with Manual Scripts in TestManager	E-2
Setting the Default Editor for Manual Scripts	E-3
Planning and Creating a Manual Script	E-3
Running a Manual Script in TestManager	E-7
Viewing the Results in the LogViewer	E-10

Working with Manual Scripts on the Web	E-11
Overview of Tasks	E-12
Software Requirements	E-13
About Shared Repositories	E-14
Installing a Web Server	E-14
Configuring a Microsoft Internet Information Server	E-16
Configuring a Microsoft Personal Web Server	E-19
Setting Up a Web Browser	E-21
Troubleshooting for Manual Scripting on the Web	E-22
Running a Manual Script on the Web	E-23
Working with External Scripts	E-26
Planning an External Script	E-26
Logging Results of an External Script	E-26
Logging Results of Several External Scripts	E-28
Running an External Script	E-29
Viewing the Results of Running an External Script	E-29

Glossary

Index

Contents

▶ ▶ ▶ Preface

Rational Robot is a complete set of tools for automating the testing of Microsoft Windows client/server and Internet applications running under Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

This manual describes how to use Rational Robot to test the quality of your applications. The manual explains how to plan tests, develop automated scripts, play back the scripts, and analyze the results. This manual is intended for application developers, quality assurance managers, and quality assurance engineers.

Other Resources

- ▶ This product contains complete online Help. From the main toolbar, choose an option from the **Help** menu.

For information about context-sensitive Help, see the following section.

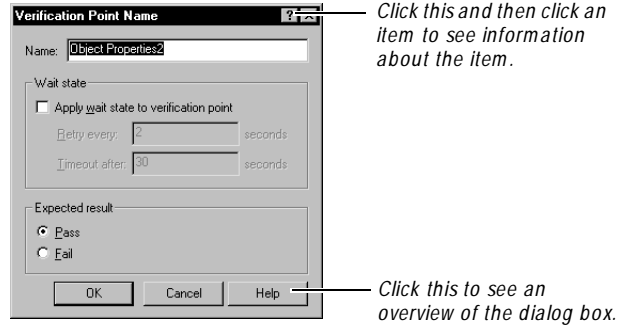
- ▶ All manuals for this product are available online in PDF format. These manuals are on the *Rational Solutions for Windows* Online Documentation CD.
- ▶ For information about training opportunities, see the Rational University Web site: <http://www.rational.com/university>.

Using Help

This product contains context-sensitive Help for dialog boxes, menus, and toolbars.

Dialog Box Help

Most dialog box Help includes overviews and detailed item information.



Menu Command Help



For menu command Help, highlight the command and press F1, or click the Help button on the toolbar and select the command. A brief description of the command also appears in the status bar.

Toolbar Button Help



For toolbar button Help, pause the pointer over the button. A yellow ToolTip appears below the button, and a brief description appears in the status bar. For more detailed information, click the Help button on the toolbar, and then select the button for which you want more information.

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our technical publications department at techpubs@rational.com.

Contacting Rational Technical Support

If you have questions about installing, using, or maintaining this product, contact Rational Technical Support as follows:

Rational Technical Support

Location	Contact Information	Notes
North America	Telephone: 800-433-5444 408-863-4000 E-mail: support@rational.com	Please be prepared to supply the following information: <ul style="list-style-type: none">– Your name, telephone number, and company name– Computer make and model
Europe	Telephone: +31 (0) 20 4546 200 E-mail: support@europe.rational.com	<ul style="list-style-type: none">– Operating system and version number– Product release number and serial number– Your Case ID number (if you are calling about a previously reported problem)
Asia Pacific	Telephone: +61-2-9419-0111 E-mail: support@apac.rational.com	
World Wide Web	http://www.rational.com	Click the Technical Support link.

▶▶▶ Part I

Introducing Rational Robot

▶▶▶ C H A P T E R 1

Introduction to Rational Robot

This chapter introduces you to Rational Robot and its components. It includes the following topics:

- ▶ What is Rational Robot?
- ▶ Using Robot with other Rational products
- ▶ Starting Robot and its components
- ▶ Tasks you can perform with Robot and its components

What Is Rational Robot?

Rational Robot is a complete set of components for automating the testing of Microsoft Windows client/server and Internet applications running under Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

The main component of Robot lets you start recording tests in as few as two mouse clicks. After recording, Robot plays back the tests in a fraction of the time it would take to repeat the actions manually.

Other components of Robot are:

- ▶ **Rational Administrator** – Use to create and manage Rational repositories, which store your testing information.
- ▶ **Rational TestManager** – Use to plan your tests, manage test assets, and create and run manual and external scripts.
- ▶ **Rational LogViewer** – Use to review and analyze test results.
- ▶ **Object Properties, Text, Grid, and Image Comparators** – Use to view and analyze the results of verification point playback.
- ▶ **Rational SiteCheck** – Use to manage Internet and intranet Web sites.

Managing the Rational Repository with the Administrator

You use the Rational Administrator to create and manage Rational repositories.

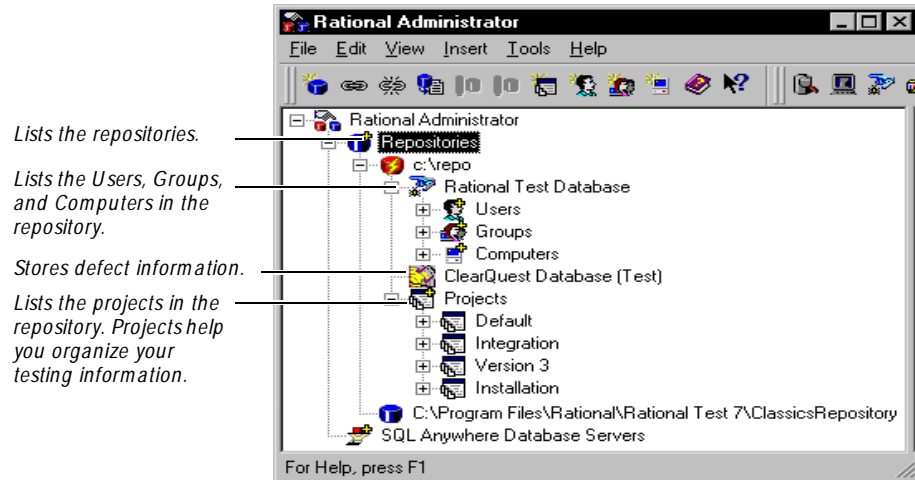
Rational repositories store application testing information, such as scripts, verification points, queries, and defects. Each repository consists of a database and several directories of files. All Rational Test components on your computer update and retrieve data from the same active repository.

Within the repository, information is categorized by projects. Projects help you organize your testing information and resources for easy tracking. Repositories and projects are created in the Rational Administrator, usually by someone with administrator privileges.

Use the Administrator to:

- ▶ Create and delete a repository.
- ▶ Connect to a repository.
- ▶ Configure a SQL Anywhere database server.
- ▶ Create and manage users, groups, and computers for a Rational Test database.
- ▶ Create and manage projects containing Rational RequisitePro databases and Rational Rose models.
- ▶ Manage security privileges for the entire Rational repository.
- ▶ Change Rational Test and ClearQuest database types.
- ▶ Use the centralized Rational License Key Administrator.
- ▶ Synchronize data among Rational Test, RequisitePro, and Rational Rose datastores using the Rational Synchronizer.

The following figure shows the main Rational Administrator window after you have created some repositories and projects:



For information about the Administrator and repositories, see the *Using the Rational Administrator* manual.

Planning and Managing Tests in TestManager

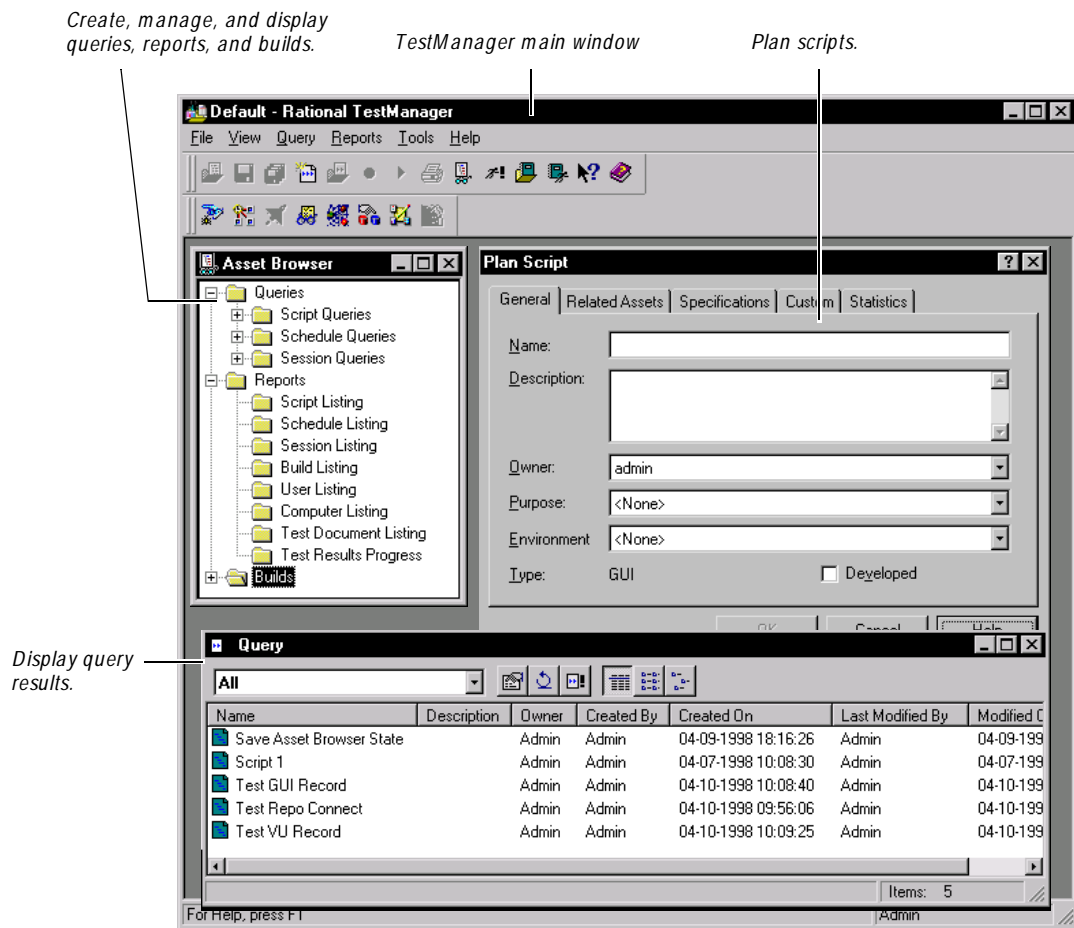
You use Rational TestManager to plan your tests, manage your test assets, and run queries and reports.

Use TestManager to:

- ▶ **Define test requirements.** These are the features and functionality that you plan to test. You can define test requirements in TestManager and Rational RequisitePro.
- ▶ **Plan scripts and schedules.** A script is a set of instructions used to navigate through and test an application. A schedule can contain scripts, information about how and where to run the script, and how to coordinate script playback. (Schedules are used in LoadTest, which is available only in Rational Suite PerformanceStudio).
- ▶ **Create, manage, and run queries.** The query tools help you manage your scripts, schedules, and sessions. You can use the default queries provided with TestManager or create queries of your own.
- ▶ **Create, manage, and run reports.** The reporting tools help you track assets such as scripts, builds, and test documents. They also help you track test coverage and progress.

- ▶ **Create and manage builds, log folders, and logs.** Logs are created when you run a script or schedule. Log folders are used to organize logs.
- ▶ **Create and manage datapools and data types.** A datapool supplies data values to variables in a script during script playback. A data type is a source of data for one datapool column.
- ▶ **Create and run manual and external scripts.** A manual script contains a set of instructions to be run by a human tester. An external script runs a program created with any tool.

The following figure shows the main TestManager window and some of the other windows and dialog boxes that you can use to manage your tests.



Developing Tests in Robot

You use Robot to develop two kinds of scripts: GUI scripts for functional testing and virtual user scripts for performance testing.

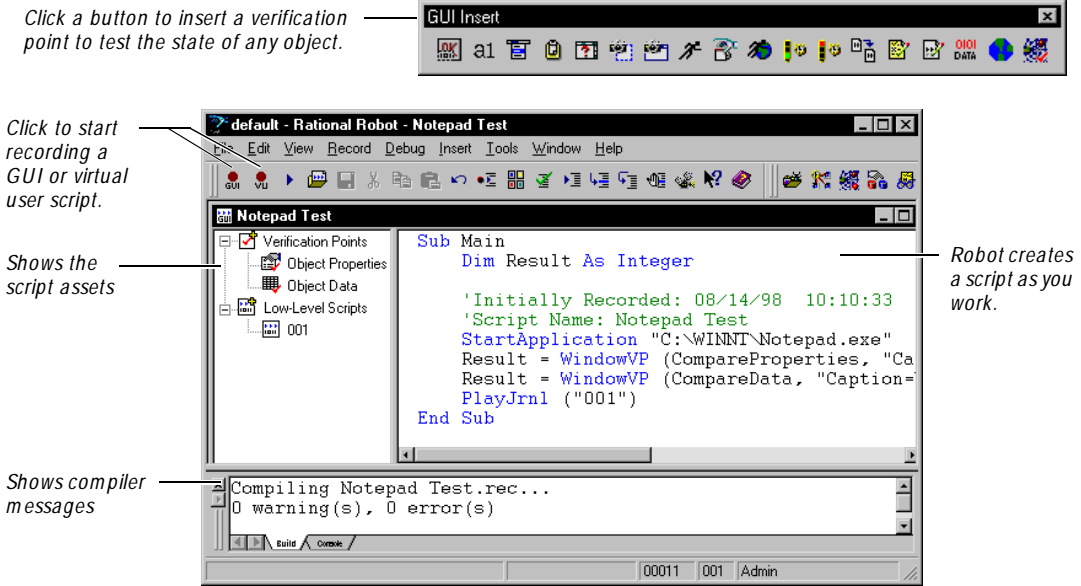
NOTE: Virtual user scripts are available only in Rational Suite PerformanceStudio. For information about virtual user scripts, see the *Using Rational LoadTest* manual.

Use Robot to:

- ▶ **Perform full functional testing.** Record and play back scripts that navigate through your application and test the state of objects through verification points.
- ▶ **Perform full performance testing.** Use Robot and LoadTest together to record and play back scripts that help you determine whether a multi-client system is performing within user-defined standards under varying loads.
- ▶ **Create and edit scripts using the SQABasic and VU scripting environments.** The Robot editor provides color-coded commands with keyword Help for powerful integrated programming during script development. (VU scripting is available only in Rational Suite PerformanceStudio.)
- ▶ **Test applications developed with IDEs** such as Java, HTML, Visual Basic, Oracle Forms, and PowerBuilder. You can test objects even if they are not visible in the application's interface.
- ▶ **Collect diagnostic information about an application during script playback.** Robot is integrated with Rational Purify[®], Rational Quantify[™], and Rational PureCoverage[™]. You can play back scripts under a diagnostic tool and see the results in the log.

The Object-Oriented Recording[®] technology in Robot lets you generate scripts by simply running and using the application-under-test. Robot uses Object-Oriented Recording to identify objects by their internal object names, not by screen coordinates. If objects change locations or their text changes, Robot still finds them on playback.

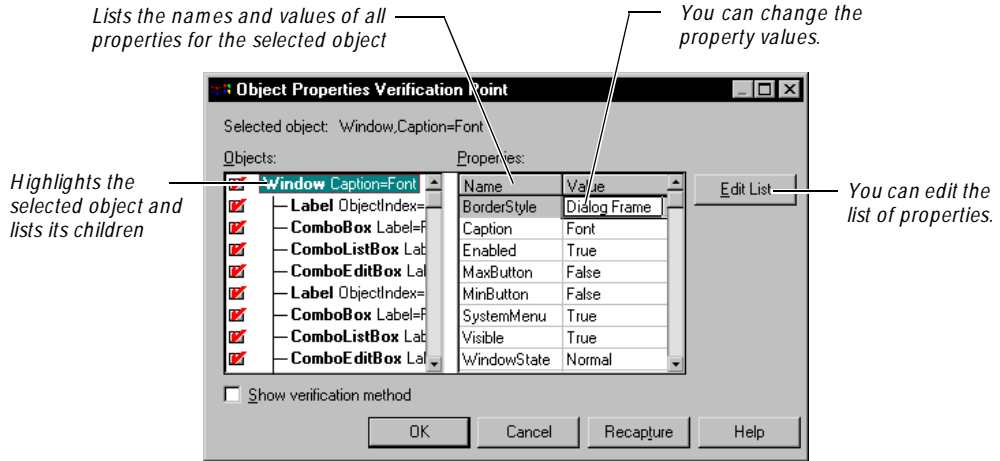
The following figure shows the main Robot window after you have recorded a script.



The Object Testing[®] technology in Robot lets you test any object in the application-under-test, including the object's properties and data. You can test standard Windows objects and IDE-specific objects, whether they are visible in the interface or hidden.

In functional testing, Robot provides many types of verification points for testing the state of the objects in your application. For example, you use the Object Properties verification point to capture the properties of an object during recording, and to compare these properties during playback.

The following figure shows the Object Properties Verification Point dialog box.



Creating Datapools

A datapool is a source of test data that scripts can draw from during playback.

If a script sends data to a server during playback, consider using a datapool as the source of the data. By accessing a datapool, a script transaction that is executed multiple times during playback can send realistic data and even unique data to the server each time. If you do not use a datapool, the same data (the data you recorded) is sent each time the transaction is executed.

When creating a datapool, you specify the kinds of data (called data types) that the script will send — for example, customer names, addresses, and unique order numbers. When you finish defining the datapool, TestManager automatically generates the number of rows of data that you specify.

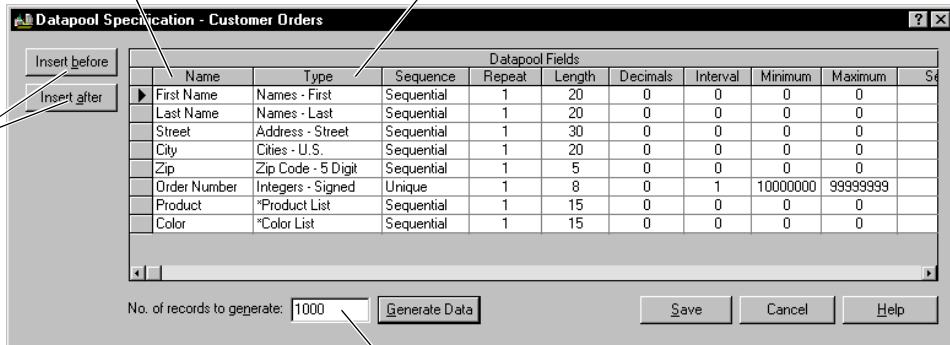
TestManager is shipped with many commonly used data types. In addition, TestManager lets you create your own data types.

The following figure shows a datapool being defined. Note that most of the data types in the **Type** column are standard data types shipped with TestManager. Two data types, Product List and Color List, are user-defined data types.

Columns to generate in the datapool file

Data types that supply data to datapool columns

Inserts new datapool columns



Number of rows to generate in the datapool file

Analyzing Results in the LogViewer and Comparators

You use the Rational LogViewer to view the logs that are created when you run scripts and schedules.

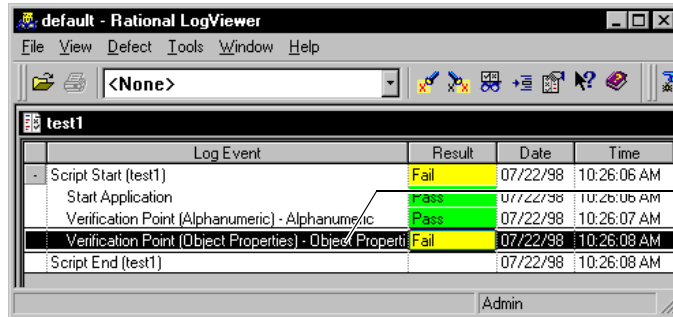
Use the LogViewer to:

- ▶ **View the results of running a script**, including verification point failures, procedural failures, aborts, and any additional playback information. Reviewing the results in the LogViewer reveals whether each script and verification point passed or failed.

Use the Comparators to:

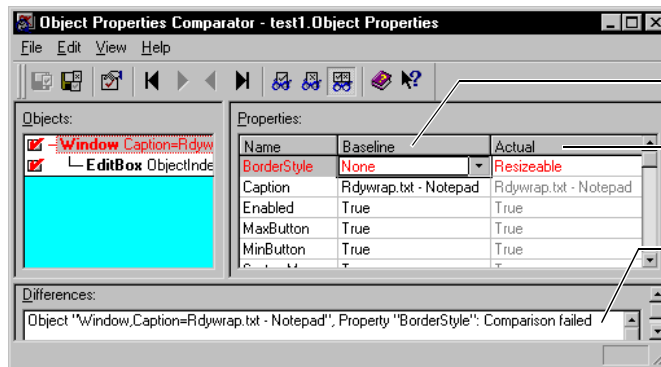
- ▶ **Analyze the results of verification points** to determine why a script may have failed. Robot includes four Comparators:
 - Object Properties Comparator
 - Text Comparator
 - Grid Comparator
 - Image Comparator

The following figure shows a log file in the LogViewer that contains a failed Object Properties verification point.



Failed Object Properties verification point. Double-click the line to open the Comparator and view the failure.

When you double-click the line that contains the failed Object Properties verification point, the Object Properties Comparator opens, as shown in the following figure. In the Comparator, the Baseline column shows the original recording, and the Actual column shows the playback that failed. Compare the two files to determine whether the difference is an intentional change in the application or a defect.



Properties in the baseline data file

Properties in the actual data file

Shows the differences between the baseline and actual files. Click a difference to highlight it in the Properties list above.

Managing Intranet and Web Sites with SiteCheck and Robot

You use Rational SiteCheck to test the structural integrity of your intranet or World Wide Web site. SiteCheck is designed to help you view, track, and maintain your rapidly changing site.

Use SiteCheck to:

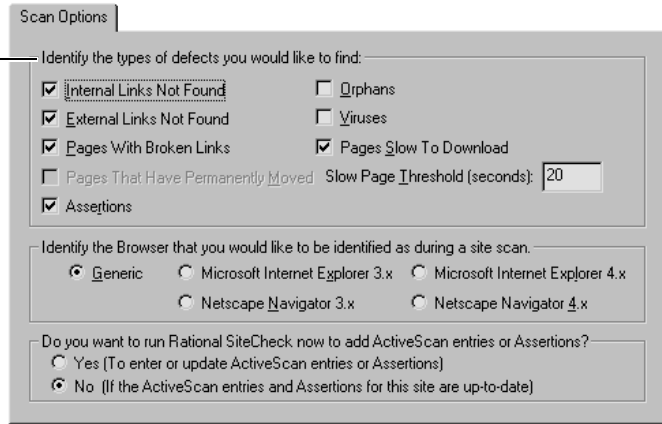
- ▶ **Visualize the structure of your Web site** and display the relationship between each page and the rest of the site.
- ▶ **Identify and analyze Web pages with active content**, such as forms, Java, JavaScript, ActiveX, and Visual Basic Script (VBScript).
- ▶ **Filter information** so that you can inspect specific file types and defects, including broken links.
- ▶ **Examine and edit the source code** for any Web page, with color-coded text.
- ▶ **Update and repair files** using the integrated editor, or configure your favorite HTML editor to perform modifications to HTML files.
- ▶ **Perform comprehensive testing of secure Web sites.** SiteCheck provides Secure Socket Layer (SSL) support, proxy server configuration, and support for multiple password realms.

Robot has two verification points for use with Web sites:

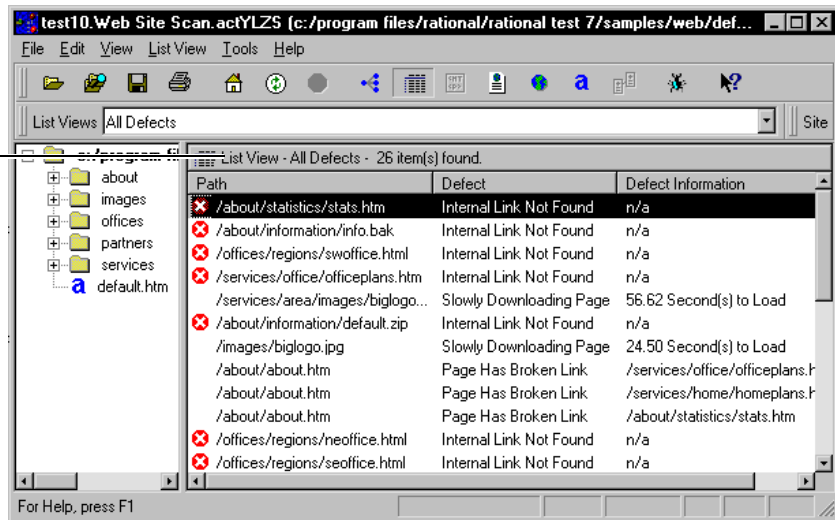
- ▶ Use the Web Site Scan verification point to check the content of your Web site with every revision and ensure that changes have not resulted in defects.
- ▶ Use the Web Site Compare verification point to capture a baseline of your Web site and compare it to the Web site at another point in time.

The following figures show the types of defects you can scan for using a Web Site verification point, and the list of defects displayed in SiteCheck.

During recording, insert a Web Site Scan verification point that checks for defects on your Web site.



During playback, SiteCheck lists all the defects on your Web site.



For more information about SiteCheck, see the *Try it! with Rational SiteCheck* card and the SiteCheck Help. For more information about the Web Site verification points, see the Robot Help.

Using Robot with Other Rational Products

Rational Robot is integrated with many other Rational products and components, including TestFactory, ClearQuest, Purify, Quantify, PureCoverage, LoadTest, and RequisitePro. The products and components are available based on what you have installed.

Testing Applications with Rational TestFactory

Rational TestFactory is a component-based testing tool that automatically generates TestFactory scripts according to the application's navigational structure.

TestFactory is integrated with Robot and its components to provide a full array of tools for team testing under Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

With TestFactory, you can:

- ▶ Automatically create and maintain a detailed map of the application-under-test.
- ▶ Automatically generate both scripts that provide extensive product coverage and scripts that encounter defects, without recording.
- ▶ Track executed and unexecuted source code, and report its detailed findings.
- ▶ Shorten the product testing cycle by minimizing the time invested in writing navigation code.
- ▶ Automate distributed testing with TestAccelerator, an application that drives and manages the execution of scripts on remote machines.
- ▶ Play back Robot scripts in TestFactory to see extended code coverage information and to create regression suites; play back TestFactory scripts in Robot to debug them.

For more information about TestFactory, see its manuals and Help.

Managing Defects with Rational ClearQuest

Rational ClearQuest is a change-request management tool that tracks and manages defects and change requests throughout the development process. With ClearQuest, you can manage every type of change activity associated with software development, including enhancement requests, defect reports, and documentation modifications.

With Robot and ClearQuest, you can:

- ▶ Submit defects directly from the LogViewer or SiteCheck.
- ▶ Modify and track defects and change requests.
- ▶ Analyze project progress by running queries, charts, and reports.

For information about using the LogViewer to enter defects into ClearQuest, see *Entering and Modifying Defects* on page 10-14. For information about using ClearQuest, see its manuals and Help.

Collecting Diagnostic Information During Playback

Use the Rational diagnostic tools to perform runtime error checking, profile application performance, and analyze code coverage during playback of a Robot script.

- ▶ **Rational Purify** is a comprehensive C/C++ run-time error checking tool that automatically pinpoints run-time errors and memory leaks in all components of an application, including third-party libraries, ensuring that code is reliable.
- ▶ **Rational Quantify** is an advanced performance profiler that provides application performance analysis, enabling developers to quickly find, prioritize and eliminate performance bottlenecks within an application.
- ▶ **Rational PureCoverage** is a customizable code coverage analysis tool that provides detailed application analysis and ensures that all code has been exercised, preventing untested code from reaching the end-user.

For information about playing back scripts under these products, see *Setting Diagnostic Tools Options* on page 9-11. For information about using the diagnostic tools, see their manuals and Help.

Performance Testing with Rational Suite PerformanceStudio

Rational Suite PerformanceStudio is a sophisticated tool for automating performance tests on client/server systems. A client/server system includes client applications accessing a database or application server, and browsers accessing a Web server.

PerformanceStudio includes Rational Robot and Rational LoadTest. Use Robot to record client/server conversations and store them in scripts. Use LoadTest to schedule and play back the scripts. During playback, LoadTest can emulate hundreds, even thousands, of users placing heavy loads and stress on your database and Web servers.

With PerformanceStudio, you can:

- ▶ Find out if your system-under-test performs adequately.
- ▶ Monitor and analyze the response times that users actually experience under different usage scenarios.
- ▶ Test the capacity, performance, and stability of your server under real-world user loads.
- ▶ Discover your server's break point and how to move beyond it.

For information about PerformanceStudio, see its manuals and Help.

Managing Requirements with Rational RequisitePro

Rational RequisitePro is a requirements management tool that helps project teams control the development process. RequisitePro organizes your requirements by linking Microsoft Word to a requirements repository and providing traceability and change management throughout the project lifecycle.

A baseline version of RequisitePro is included with Rational TestManager. When you define a test requirement in RequisitePro, you can access it in TestManager.

With the full version of RequisitePro, you can:

- ▶ Customize the requirements database and manage multiple requirement types.
- ▶ Prioritize, sort, and assign requirements.
- ▶ Control feature creep and ensure software quality.
- ▶ Track what changes have been made, by whom, when, and why.
- ▶ Integrate with other tools, including Rose, ClearCase, Rational Unified Process, and SoDA.

Sharing Data with Other Rational Products

You can share or link data between applications that store data in Rational Test, RequisitePro, or Rose datastores. However, when you update data in one of these products, you often want to update this data in other products. You use the Rational Synchronizer to update data.

Use the Synchronizer to:

- ▶ Ensure consistency of data across several products. A single change in one type of data can be updated in other types of data simultaneously.
- ▶ Jump-start work in one product with the data from another product. For example, you can create requirements using TestManager or RequisitePro, and then import one or more of these requirements to a Rose model.

- ▶ Analyze the impact of changes. You can synchronize data in several different products to gain a deeper understanding of how changes in requirements or design changes impact your overall software development process.

You can start the Synchronizer from the Administrator or from TestManager. For more information about using the Synchronizer, see the *Using the Rational Administrator* manual.

Starting Robot and Its Components

Before you start using Robot, you need to have:

- ▶ Rational Robot installed. For information, see one of the following manuals: *Installing Rational Suite* or *Installing Rational TeamTest and Rational Robot*.
- ▶ A Rational repository and a project within the repository. For information, see the *Using the Rational Administrator* manual.

Logging in

When you log into Robot or one of its components, you provide your user ID and password, which are assigned by your administrator. You also specify the repository and project to log into.

To log in:

- ▶ From **Start** → **Programs** → *Rational product name*, start **Rational Robot** or one of its components to open the Rational Repository Login dialog box.

Type your user ID and password.
If you do not know these, see your administrator.

Select a repository. To change repositories later, exit all Robot components and log in again. (Repositories are created in the Rational Administrator.)

Select a project. You can change to another project within the same repository after you log in.

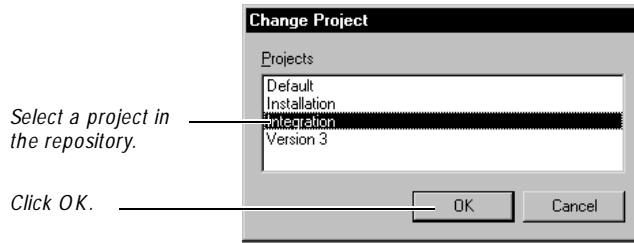
Click OK to log in.

Changing to Another Project

Once you are logged into a Robot component, you can easily change to another project in the same repository.

To change to another project:

- ▶ Click **File** → **Change Project**.

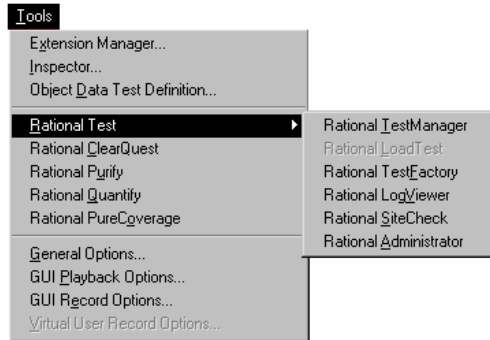


To change to a project in a different repository, exit all Robot components and then log in again. Select the repository and project in the Rational Repository Login dialog box. (New repositories and projects are created in the Rational Administrator.)

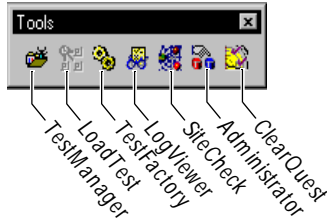
Opening Other Rational Products and Components

Once you are logged into a Robot component, you can start other products and components from either:

The Tools menu



The Tools toolbar



Some components also start automatically when you perform certain functions in another component.

Tasks You Can Perform with Robot and Its Components

The following table lists the tasks that you can perform, the component to use, and where to find more information:

To	Use this component	See
Plan tests and manage test assets	TestManager	Chapter 2, <i>Planning Your Tests</i> Chapter 3, <i>Managing Builds, Log Folders, and Logs</i>
Record GUI scripts	Robot	Chapter 4, <i>Recording GUI Scripts</i> Chapter 5, <i>Adding Features to GUI Scripts</i>
Create verification points to test the state of objects	Robot	Chapter 6, <i>Creating Verification Points in GUI Scripts</i>
Edit, compile, and debug scripts	Robot	Chapter 7, <i>Editing, Compiling, and Debugging Scripts</i>
Supply data values to the variables in a script during playback	Robot TestManager	Chapter 8, <i>Working with Datapools</i>
Play back GUI scripts	Robot	Chapter 9, <i>Playing Back GUI Scripts</i>
Review and analyze test results, and enter defects	LogViewer	Chapter 10, <i>Reviewing Logs with the LogViewer</i>
View and analyze the results of verification points	Object Properties Comparator Text Comparator Grid Comparator Image Comparator	Chapter 11, <i>Using the Object Properties Comparator</i> Chapter 12, <i>Using the Text Comparator</i> Chapter 13, <i>Using the Grid Comparator</i> Chapter 14, <i>Using the Image Comparator</i>

To	Use this component	See
Create and run queries to help you manage information in your projects	TestManager	Chapter 15, <i>Querying the Rational Repository</i>
Create and run reports to help you manage your testing efforts	TestManager	Chapter 16, <i>Running TestManager Reports</i>
Test Visual Basic applications	Robot	Chapter 17, <i>Testing Visual Basic Applications</i>
Test Oracle Forms applications	Robot	Chapter 18, <i>Testing Oracle Forms Applications</i>
Test HTML applications	Robot	Chapter 19, <i>Testing HTML Applications</i>
Test Java applications	Robot	Chapter 20, <i>Testing Java Applets and Applications</i>
Test PowerBuilder applications	Robot	Chapter 21, <i>Testing PowerBuilder Applications</i>
Test PeopleTools applications	Robot	Chapter 22, <i>Testing PeopleTools Applications</i>
Create and run manual and external scripts	TestManager	Appendix E, <i>Working with Manual and External Scripts</i>
Manage Internet and intranet Web sites	SiteCheck	<i>Try it!</i> with Rational SiteCheck card Rational SiteCheck Help

NOTE: All Robot components include complete Help. For information about how to use the Help, see *Using Help* in the Preface.

▶ ▶ ▶ Part II

Planning Tests

▶▶▶ CHAPTER 2

Planning Your Tests

The first phase of a software testing effort involves test planning. In this phase, you write test plans, define test requirements, and plan test scripts that will validate the test requirements. Rational TestManager is the tool you use to plan your tests.

This chapter describes the planning phase of your testing effort. It includes the following topics:

- ▶ Working with test plans and other test documents
- ▶ Defining test requirements
- ▶ Planning scripts
- ▶ Planning LoadTest schedules
- ▶ Managing requirements
- ▶ Customizing scripts and LoadTest schedules

Working with Test Plans and Other Test Documents

Often, the first part of the test planning phase involves the creation of test plans and other documents. A **test plan** defines a testing project so it can be properly measured and controlled. The test plan usually describes the features and functions you are going to test and how you are going to test them. Often, the test plan discusses resource requirements and defines a project schedule. Large software projects may require several test plans, each one written by a separate quality engineer.

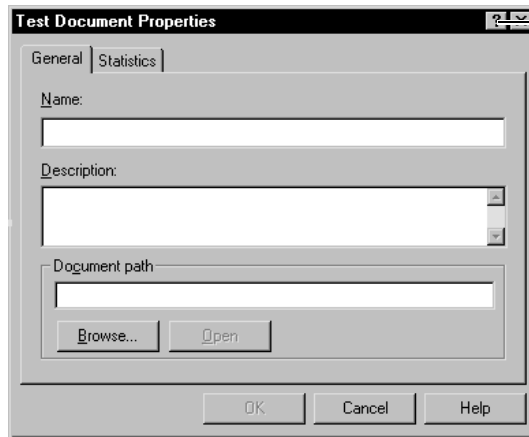
In TestManager, test plans, project schedules, and other related documents are called **test documents**. You develop your test documents using any word processing or scheduling program, and you create references to these documents in TestManager. The references are stored in the Rational repository. If the test documents themselves are modified during the project, you can use TestManager to rename, copy, or delete the references. You can also open the test documents from TestManager.

Creating Test Document References

After you create test plans and other documents, you can use TestManager to create references to these documents. By doing this, you can use TestManager as your “control center” from which you can access any document that is important to the project.

To create a reference to a test document:

1. Click **Tools** → **Manage Test Documents**:
2. Click **New**.



For detailed information about an item, click the question mark, and then click the item.

3. Type a name (40 characters maximum) that refers to an existing test document. For clarity, use a name that is the same or very similar to the actual document name.
4. Type a short description of the test document.
5. Type the path to the test document, or click **Browse** to search for the path.
6. Click **OK** to close the Test Document Properties dialog box.
7. Click **Close** to close the Manage Test Documents dialog box.

Editing Test Document References

If test documents are modified or deleted during a project, you can use TestManager to rename, copy, or delete the reference. You can also edit the reference’s name, description, or path.

To edit a reference to a test document:

1. Click **Tools** → **Manage Test Documents**.
2. Select a test document from the list.
3. Click one of the following buttons:

Edit – To change the description of the reference or to map the reference to a different source document.

Rename – To change the name of the reference.

Delete – To remove the reference.

Copy – To copy a reference to a test document (not the document itself).

4. Complete the edit, rename, deletion, or copy process.
5. Click **Close**.

Viewing Test Documents

TestManager provides easy access to your test documents.

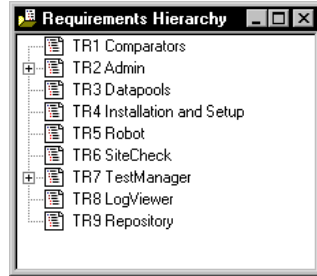
To open a test document:

1. Click **Tools** → **Manage Test Documents**.
2. Select a test document from the list.
3. Click **Edit**.
4. Click **Open** to open the document using its associated editor.

Defining Test Requirements

After you write your test plans and other test documents, the next step in test planning is to define your **test requirements** — the features and functionality you plan to test. You can define test requirements using the Requirements Hierarchy in TestManager. The Requirements Hierarchy is a graphical outline of requirements and nested child requirements. You can define a hierarchy of your project's requirements in any way that is appropriate for your project.

The following figure shows a sample Requirements Hierarchy:



Requirements in TestManager are stored in a Rational RequisitePro database. RequisitePro is a requirements management tool that helps project teams control the development process by organizing, managing, and tracking the changing requirements of your system. You can define and access test requirements in either RequisitePro or TestManager.

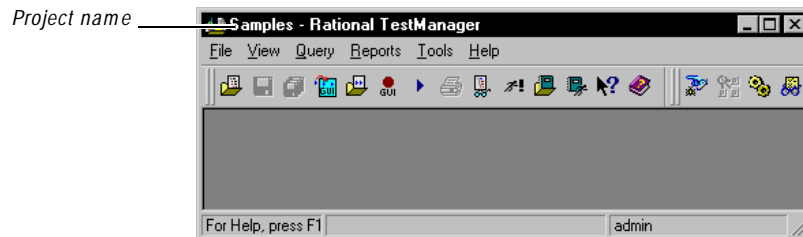
TestManager includes a baseline version of RequisitePro that you can use to build a test requirements hierarchy and then associate scripts and schedules with the requirements in the hierarchy.

With the full version of RequisitePro, you can customize the requirements database and incorporate other advanced features such as traceability, change notification, and requirements attribute management. The full version of RequisitePro is included as part of Rational Suites, or you can purchase it separately.

Working with Projects

Rational Test categorizes test information within a repository by project. You can use the Rational Administrator to create and manage projects.

The **project** represents the top-level name of a software testing effort. Its name appears in the TestManager title bar and in the title bar of the other components of Rational Test.



Projects help you categorize your software testing information and resources for easy tracking. Each project can include information about requirements, test documents, scripts, verification points, and schedules.

The number of projects in a repository depends on the complexity of the application-under-test or on the number of ongoing unrelated testing efforts. For example, you could divide a large application into several smaller projects, or you could define a separate project for each unrelated testing effort.

The Requirements Hierarchy in TestManager is project-specific — that is, it is associated with a particular project. When you log into TestManager, you select the repository and project you want to work with. The default is the last project that was used. For information about accessing other projects, see *Changing to Another Project* on page 1-16. For information about creating and deleting projects, see the *Using the Rational Administrator* manual.

TestManager includes an Import Test Assets wizard that you can use to copy or import test scripts and other test assets, such as datapools and data types, from one project to another. The destination project may reside within the same repository or in another repository. For more information, see *Importing Scripts from Other Projects* on page 2-18.

Building the Requirements Hierarchy

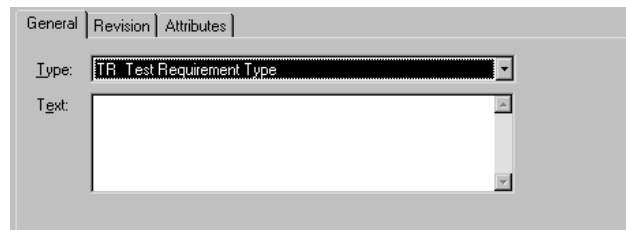
The Requirements Hierarchy is displayed in its own window within TestManager. It displays requirements as well as the scripts and schedules that you can associate with those requirements. You build the Requirements Hierarchy as you would build an outline. Each requirement can have multiple levels of nested sub-requirements, called child requirements.

Inserting a Requirement

To insert a requirement in the hierarchy:



1. Click **File** → **Open Requirements** to open the Requirements Hierarchy.
2. Click **Edit** → **Insert Requirement** to open the Requirement Properties dialog box.



3. Select a requirement type from the list.

Requirement types are defined in RequisitePro.

NOTE: You cannot change the type of an existing requirement. In addition, a child requirement must be the same type as its parent.

4. Type the requirement name in the **Text** box.
5. Click the **Revision** tab.
6. In the **Description** box, type a description for this version of the requirement.

The screenshot shows a dialog box with three tabs: General, Revision, and Attributes. The Revision tab is active. It contains a 'Last Saved' section with the following fields: Revision (1.0000), Label (empty), Date (01/26/2000), Time (2:13:39 PM), and Author (admin). There is a Description text area and a History... button.

For example, you might type *Update priority attribute value from medium to high.*

7. Type a label for the revision — for example, *beta 2*, *alpha 1*, or the build number.
8. Click the **Attributes** tab.

Attribute	Value
AutoTest	False
Tested By	<no entry>
Test Status	<no entry>
Date Tested	
Build #	
Output	
Test Notes	
Priority	Medium
Priority Desc.	
Risk	Medium

Attributes describe the requirements in the Requirements Hierarchy. In TestManager, you can assign values to attributes. To add or modify attributes and their values, you must use RequisitePro.

9. Click in the Value row next to the attribute you want to edit.
 - For a text field, type a value.
 - For a single-value field, select a value from the list.
 - For a multi-value field, double-click the **Value** column to open the Select Attribute Values dialog box. Move the values from the **Available** list to the **Selected** list and click **OK**.
10. Click **OK**.

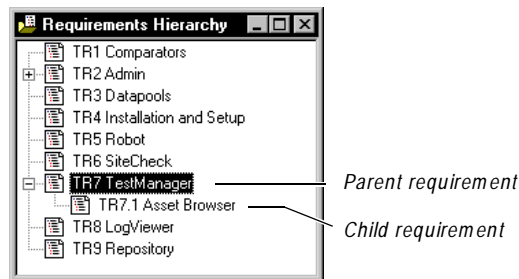
The new requirement appears in the hierarchy.

Inserting a Child Requirement

To insert a child requirement:

1. Select a requirement in the hierarchy and click **Edit** → **Insert Child Requirement**, or right-click a requirement in the hierarchy and click **Insert Child Requirement**.
2. Fill in the Requirement Properties dialog box as described in *Inserting a Requirement* on page 2-5. Remember that a child requirement must be the same type as its parent.

When you finish, the child requirement appears beneath its parent in the hierarchy.



Importing Requirements from PowerBuilder Libraries

In TestManager, you can import requirements directly from a PowerBuilder library (version 5.0 and later).

To import requirements from a PowerBuilder library:

1. Make sure PowerBuilder is set to the application from which you want to import hierarchy information.

2. From the Requirements Hierarchy, select the requirement to which you want to append the new requirements.
3. Click **File** → **Import PowerBuilder PBL**.
4. Select the PowerBuilder library to import and click **Open**.

NOTE: If TestManager cannot find the PB.INI file, it opens a dialog box for specifying the path of the file. Type or browse the full path to this file and click **OK**. The PB.INI file is usually located in the PowerSoft installation directory. If your PowerBuilder application uses only a single library, you can leave the edit box blank and click **OK**.

5. In the Object Types section of the Library Entries dialog box, select the type of PowerBuilder objects to import.
6. In the Object Structures section of the Library Entries dialog box, select the type of object structure to use for the hierarchy.

Hierarchy builds a Requirements Hierarchy based on the tree structure of the PowerBuilder application.

List by Type builds a Requirements Hierarchy sorted by the selected object types.

7. If you want TestManager to automatically create script names that correspond to the requirements, select the **Generate Test Scripts** check box.
8. Click **OK**.

TestManager reads the structure information from the PowerBuilder library file and appends the data to the Requirements Hierarchy.

Editing Requirement Properties

To change the name of a requirement or to change other properties:

1. Right-click the requirement and click **Properties** or double-click the requirement to open the Requirement Properties dialog box.
2. Make edits to the properties as needed.
3. Click **OK**.

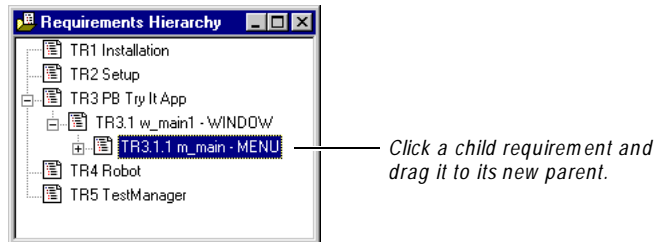
Expanding and Collapsing the Requirements Hierarchy

For large applications, the Requirements Hierarchy can become quite extensive. While creating or editing a hierarchy, you can simplify navigation through the tree structure by showing or hiding specific levels of the hierarchy. All parent requirements display a plus (+) or minus (-) sign on their left side.

- ▶ To view the children, click the plus (+) sign.
- ▶ To hide the children, click the minus (-) sign.

Reparenting Child Requirements

In TestManager, you can drag a child requirement in the Requirements Hierarchy from one parent requirement to another. When you reparent a requirement, the entire sub-tree of children is also moved to the new parent and is renumbered.



Reparenting rules are as follows:

- ▶ The child requirement must be the same requirement type as the new parent.
- ▶ Document-based requirements from RequisitePro documents cannot be reparented.
- ▶ Traceability relationships in RequisitePro are not allowed between the new parent and its new children.
- ▶ The parent-child hierarchy that is created when reparenting cannot exceed 25 levels.

Planning Scripts

After you create your test documents and define your test requirements, you are ready to use TestManager to plan the scripts and attach them to test requirements. After you complete this process, you can:

- ▶ Record a GUI or virtual user script. To do this, start recording in Robot and select a planned script.
- ▶ Create a manual script. To do this, open the planned script in TestManager.

By planning your scripts, you will know the size of the testing effort, and you will be able to assign the development of each script to the appropriate individual. This model can also help you develop logical and consistent naming conventions for all of your scripts.

To plan scripts, open the Plan Script dialog box and assign properties, such as a name, description, and owner. For details, see *Assigning a Name and Other Properties* on page 2-11.

Other tasks you can perform with scripts include:

- ▶ Attaching a script to a test requirement
- ▶ Referencing specification files, such as functional specifications or design documents
- ▶ Customizing script properties
- ▶ Displaying script statistics
- ▶ Referencing include files and external C libraries (VU scripts only)
- ▶ Controlling whether Robot starts after planning a script
- ▶ Deleting and importing scripts

These tasks are described in more detail in the following sections.

About Scripts

Much of your testing effort involves planning and recording GUI and virtual user scripts. These types of scripts have two parts:

- ▶ A file (extension .rec) that can be run by Robot or by a LoadTest schedule. (LoadTest requires Rational Suite PerformanceStudio).
- ▶ A set of script properties, such as the type and purpose of the script.

You generate a script file when you record your GUI actions or client/server requests with Robot. Robot translates your actions into scripting language commands (SQABasic for GUI activities, VU for client/server requests), and writes them to the script.

Typically, you define the script's properties when you plan the script with TestManager. You can also define script properties in Robot after you record the script.

By playing back scripts in Robot, you can perform functional tests of GUI objects. A functional test helps you determine whether a system functions as intended by verifying objects over successive builds of the application-under-test.

If you have Rational Suite PerformanceStudio, you can also play back GUI scripts and virtual user scripts in LoadTest schedules. For information about virtual user scripts and the kinds of tests you can perform in LoadTest, see the *Using Rational LoadTest* manual.

For information about manual and external scripts, see Appendix E, *Working with Manual and External Scripts*. For information about scripts in TestFactory, see the *Using Rational TestFactory* manual.

Assigning a Name and Other Properties

All scripts must have a name and can have several other optional properties. Script properties include:

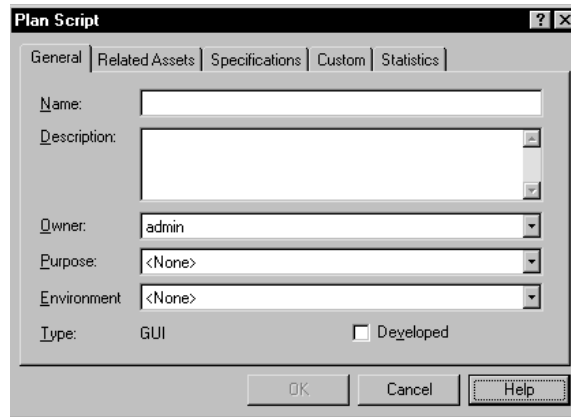
- ▶ Name, description, owner, purpose, and test environment
- ▶ Related assets such as test requirements
- ▶ Notes and specification files
- ▶ Custom keywords

To assign a name and other properties to a script:

1. Do one of the following to open the Plan Script dialog box:
 - To plan a script and attach it to a requirement, open the Requirements Hierarchy (**File** → **Open Requirements**). Right-click the requirement and click **Plan** and the type of script.
 - To plan a script without attaching it to a requirement, make sure that no requirement is selected. Click **File** → **Plan** and the type of script.

The appearance of the Plan Script dialog box differs slightly depending on the type of script you are planning.

The Plan Script dialog box for GUI scripts



The screenshot shows the 'Plan Script' dialog box with the 'General' tab selected. The fields are as follows:

- Name: [Empty text box]
- Description: [Empty text box with scrollbars]
- Owner: [Dropdown menu showing 'admin']
- Purpose: [Dropdown menu showing '<None>']
- Environment: [Dropdown menu showing '<None>']
- Type: GUI [Checked], Developed [Unchecked]

Buttons at the bottom: OK, Cancel, Help.

2. Type a name (40 characters maximum) for the script, using a logical naming convention.
3. Type an explanation of the script in the **Description** box.
4. Select the script's developer from the **Owner** list. Owners are defined in Rational Administrator.
5. Select the purpose of the script from the **Purpose** list. For information about adding purposes to the list, see *Customizing the Purpose List* on page 2-23.
6. Select the environment (operating system) in which the script will be used from the **Environment** list. For information about adding environments to the list, see *Customizing the Environment List* on page 2-22.
7. Click **OK** and proceed to the following section, *Attaching Scripts to Test Requirements*.

NOTE: The **Developed** check box indicates whether or not a script has actually been developed. TestManager considers a script to be developed when it has been recorded or edited. When planning a new GUI or virtual user script, the check box should be cleared. For information about manual and external scripts, see Appendix E, *Working with Manual and External Scripts*.

Attaching Scripts to Test Requirements

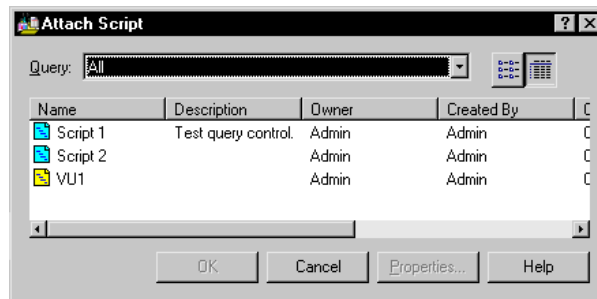
By attaching scripts to test requirements, you can more easily track which areas of the application-under-test have test coverage. When you plan a script from the Requirements Hierarchy, the script is automatically attached to a test requirement. However, if you use **File** → **Plan** to plan the script, and no requirement is selected, you will need to explicitly attach the script to a test requirement.

NOTE: A script can be attached to only one requirement at a time.

Attaching Scripts from the Requirements Hierarchy

To attach a script to a requirement from the Requirements Hierarchy:

1. Right-click a requirement in the hierarchy and click **Attach Script**, or select a requirement in the hierarchy and click **Edit** → **Attach Script**:

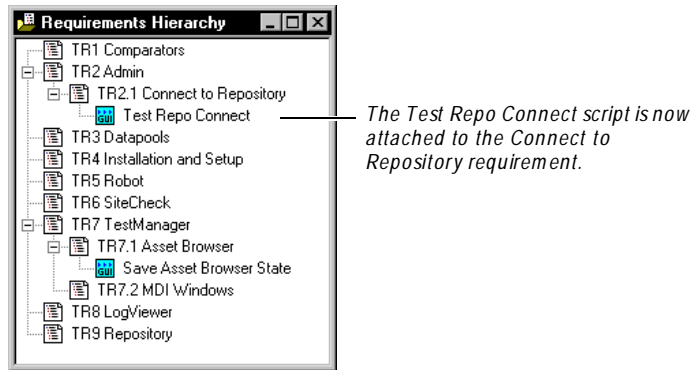


2. Optionally, select a query from the Query list.

The query lets you narrow down the list of scripts that appear. This is extremely useful in projects with hundreds of scripts. For example, you can select a query that lists all scripts, or lists only GUI or virtual user scripts. For information about how to create a query, see *Creating New Queries* on page 15-4.

3. Select a script from the list and click **OK** to attach it to the requirement.

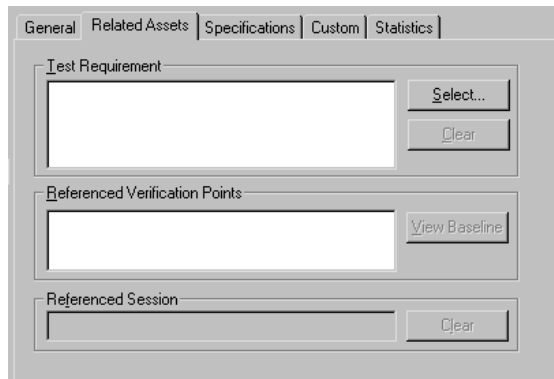
The script appears in the hierarchy beneath the requirement.



Attaching Scripts from the Plan Script Dialog Box

To attach a script to a requirement from the Plan Script dialog box:

1. Click **File** → **Plan** and the type of script.
2. Supply a name, description, owner, purpose, environment, and type for the script. For more information, see *Planning Scripts* on page 2-10.
3. Click the **Related Assets** tab.



NOTE: If a requirement is selected in the Requirements Hierarchy, the requirement will appear automatically in the Test Requirement box.

4. Click **Select**.
5. Select the test requirement to attach the script to.
6. Click **OK**.
7. Click **OK** to close the Plan Script dialog box.

NOTE: You can also use this procedure to attach a script to a test requirement from within Robot. To do so, open the Script Properties dialog box and follow the previous procedure.

Detaching a Script from a Requirement

When you detach a script from a requirement, you simply eliminate the mapping between them. Afterwards, you can attach the script to another requirement.

To detach a script from a requirement:

- ▶ Right-click the script in the Requirements Hierarchy and click **Detach**, or click **Edit** → **Detach**.

Moving a Script from One Requirement to Another

To move a script to a different requirement, select the script in the Requirements Hierarchy and drag it to a different requirement, or use the **Attach Script** command to attach the script to the new requirement.

Referencing Specification Files

You can set up your scripts so that each script references a specification file, such as a functional or design specification, or you can simply add a detailed note describing the script.

To reference a specification file:

1. Right-click in the Requirements Hierarchy and click **Plan** and the type of script, or click **File** → **Plan** and the type of script.
2. Supply a name, description, owner, purpose, environment, and type for the script. For more information, see *Planning Scripts* on page 2-10.
3. Click the **Specifications** tab.
4. Type any additional information about the script in the **Notes** box.
5. Type the path to the specification file, or click **Browse** to locate the file.

6. To open the specification file, click **Open**. Close the file when you have finished viewing it.
7. Click **OK** to close the Plan Script dialog box.

Customizing Script Properties

When you use the Plan Script dialog box to define a script, you assign several properties, such as a name, description, owner, environment, purpose, and type. With TestManager, you can add your own properties and also customize the values that are used with many of the standard properties. You can define both the property itself (the label) and the values that can be used with that property. For example, you could add a new property called Test Type, and you could add Test Type values, such as UI, functional, and performance. For more information about customizing script properties, see *Customizing Scripts and LoadTest Schedules* on page 2-22.

Displaying Script Statistics

To display statistics about a script, such as the creation date, modification date, or the name of the developer:

1. Right-click a script in the Requirements Hierarchy and click **Properties**.
2. Click the **Statistics** tab.

Referencing Include Files and External C Libraries

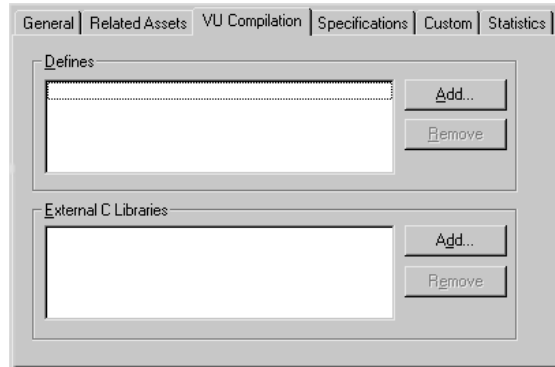
NOTE: This feature requires Rational Suite PerformanceStudio.

Use the **VU Compilation** tab of the Script Properties dialog box to add # defines and external C libraries that may be needed to compile virtual user scripts used in Rational LoadTest.

To reference include files and external C libraries:

1. Right-click in the Requirements Hierarchy and click **Plan** and the type of script, or click **File** → **Plan** and the type of script.
2. Supply a name, description, owner, purpose, environment, and type for the script. For more information, see *Planning Scripts* on page 2-10.

3. Click the **VU Compilation** tab.



4. Click the **Add** button in the **Defines** section.

In this section you can add C-preprocessor directives, such as # define, # include, # ifdef, and # if to virtual user scripts.

5. Type the name of the directive.
6. Click the **Add** button in the **External C Libraries** section.

In this section you can add references to user-written external C libraries.

7. Type or browse to the name of the library, and then click **OK**.

NOTE: LoadTest requires that all external C libraries (.DLLs on Windows NT 4.0) be placed in the Script\externC directory of the current project. When specifying the library name, you do not need to include the .DLL extension.

8. Click **OK**.

Controlling Whether Robot Starts After Planning a Script

You can control whether or not Robot starts automatically after you plan a new script in TestManager. You can plan all of your scripts before you start recording in Robot, or you can plan and record one script at a time.

To control whether Robot starts after you plan a new script:

1. Click **Tools** → **Options**.
2. Set the **Launch Robot** option:
 - Select the **Launch Robot after creating scripts** check box to start Robot automatically after you plan a new script.
 - Clear the **Launch Robot after creating scripts** check box to prevent Robot from starting automatically after you plan a new script.
3. Click **OK**.

Deleting Scripts

There may be times when you want to delete scripts that are no longer useful. To delete a script:

- ▶ Right-click a script in the Requirements Hierarchy and click **Delete**.

Deleting a GUI script from the repository also deletes its corresponding script file (.rec), executable file (.sbx), verification points, and low-level scripts.

Deleting a virtual user script deletes the .s file and its properties, but not the associated watch file (.wch).

For information about deleting scripts from Robot, see *Deleting Scripts* on page 7-15. For information about deleting scripts from the Query window, see *Deleting Scripts, Schedules, and Sessions* on page 15-3.

Importing Scripts from Other Projects

You can use the Import Test Assets wizard to import scripts from another project in the same repository or from a different repository.

To import a script:

1. Click **File** → **Import Test Assets**.
2. Select **Script** from the list, click one of the **Overwrite** options, and click **Next** to move to the next page.
3. Select the repository and project that contain the scripts you want to import. Then, type a user ID and password for the project and repository and click **Next**.
4. Select the scripts you plan to import and click **Next**. If necessary, select a query from the list to display just the types of scripts you are interested in. For details about how to work with queries, see Chapter 15, *Querying the Rational Repository*.

- Review the summary page that is displayed. If you want to change any settings, click **Back**. If you are satisfied with the settings, click **Finish**.

TestManager displays a status window while the scripts are being imported.

Planning LoadTest Schedules

NOTE: This feature requires Rational Suite PerformanceStudio.

If you are a Rational LoadTest user, you can plan and organize your LoadTest schedules before you actually build them. A **schedule** is a file that contains scripts, information about how to run the scripts, and information about how to coordinate script playback. You can plan your LoadTest schedules in much the same way that you plan your scripts.

To plan your schedules:

- Right-click the requirement you want to attach the schedule to and click **Plan** → **Schedule**, or click **File** → **Plan** → **Schedule**.

- Type a name for the schedule. The name can contain up to 40 characters and should be similar to the requirement name.
- Type a brief explanation of the schedule in the **Description** box.
- Select the schedule's developer in the **Owner** list. Owners are defined in Rational Administrator.
- Click **OK**.

Attaching Schedules to Test Requirements

You can attach a LoadTest schedule to a requirement just as you would attach a script. You can attach a schedule to a requirement from the Requirements Hierarchy or from the Plan Schedule dialog box. A schedule can be attached to only one requirement at a time.

Attaching Schedules from the Requirements Hierarchy

To attach a schedule to a requirement from the Requirements Hierarchy:

1. Right-click a requirement in the hierarchy and click **Attach Schedule**, or select a requirement in the hierarchy and click **Edit** → **Attach Schedule**.
2. Select a query from the list. The query lets you narrow down the list of schedules that are displayed.
3. Select the schedule from the list and click **OK** to attach it to the requirement.

The schedule appears in the hierarchy beneath the requirement.

Attaching Schedules from the Plan Schedule Dialog Box

To attach a schedule to a requirement from the Plan Schedule dialog box:

1. Click **File** → **Plan** → **Schedule**.
2. Type the name of the schedule.
3. Click the **Test Requirement** tab.
4. Click **Select** to open the Requirements Hierarchy. Select the test requirement you want to attach the schedule to.
5. Click **OK** to close the Requirements Hierarchy.
6. Click **OK** to close the Plan Schedule dialog box.

Detaching a Schedule from a Requirement

To detach a schedule from a requirement, right-click the schedule in the Requirements Hierarchy and click **Detach**, or click **Edit** → **Detach**.

Moving a Schedule from One Requirement to Another

To move a schedule to a different requirement, select the schedule in the Requirements Hierarchy and drag it to a different requirement, or use the **Attach Schedule** command to attach the schedule to the new requirement.

Referencing Specification Files

A schedule can reference a specification file just as a script can. For more information, see *Referencing Specification Files* on page 2-15.

Customizing Schedule Properties

With TestManager, you can add your own customized schedule properties. You can define both the property (the label) and the values that can be used with that property. For information about customizing schedule properties, see *Customizing Scripts and LoadTest Schedules* on page 2-22.

Displaying Schedule Statistics

To display statistics about a schedule, such as the creation date, the modification date, or the name of the developer:

1. Right-click a schedule in the Requirements Hierarchy and click **Properties**.
2. Click the **Statistics** tab.

Deleting Schedules

There may be times when you want to delete schedules that are no longer useful. To delete a schedule, right-click a schedule in the Requirements Hierarchy and click **Delete**.

You can also delete schedules from the Query window. For information, see *Deleting Scripts, Schedules, and Sessions* on page 15-3.

Managing Requirements

As a software project evolves, some of your test requirements may need to be changed or deleted. TestManager helps you maintain requirements throughout the project's life cycle.

Editing Requirement Properties

To edit requirement properties:

1. Click **File** → **Open Requirements** to open the Requirements Hierarchy.
2. Right-click a requirement in the hierarchy and click **Properties**.
3. If necessary, edit the requirement text.
4. Click the **Revision** tab. Update the description and label as needed. Click the **History** button to view the requirement's history.
5. Click the **Attributes** tab. If necessary, update the attribute values.
6. Click **OK**.

Deleting Requirements

When you delete a requirement, all of its child requirements are also deleted. In addition, any references to the deleted test requirements in scripts are removed.

To delete a test requirement:

1. From the Requirements Hierarchy, right-click the requirement to delete.
2. Click **Delete**.
3. Click **Yes** to confirm the deletion.

Customizing Scripts and LoadTest Schedules

You can tailor much of the terminology associated with scripts and schedules to the standards and practices used within your organization. For example, you can:

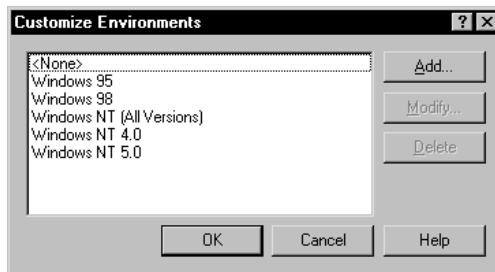
- ▶ Customize the values that are displayed in the Script Properties and Plan Script dialog boxes, such as the list of operating environments and test types.
- ▶ Define custom fields to be used with scripts and schedules.
- ▶ Add custom labels to be used with custom fields.

Customizing the Environment List

You can add new operating environments to the list and modify or delete existing ones.

To add, modify, or delete environments:

1. Click **Tools** → **Customize** → **Environments**.



2. Click the appropriate button.

Customizing the Purpose List

Another way to customize scripts is to edit the purpose list. You assign a purpose to indicate why you would use a script. You can add new purposes to the list and modify or delete existing ones.

To add, modify, or delete purposes:

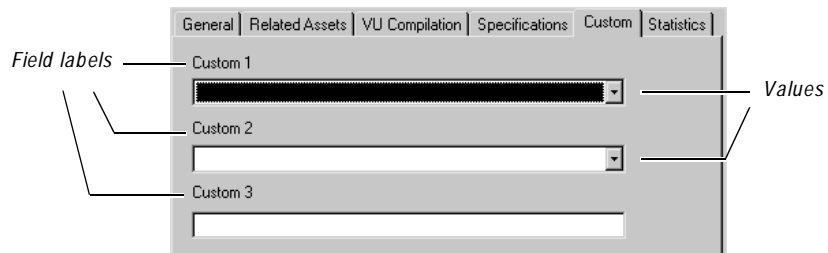
1. Click **Tools** → **Customize** → **Purpose**.
2. Click the appropriate button.

Defining Custom Field Labels and Values

When you use the Plan Script and Plan Schedule dialog boxes to plan new scripts and schedules, you assign a name, description, owner, purpose, environment, and type.

You can add your own values to the purpose, environment, and type lists. You can also customize the field labels and list items that appear on the **Custom** tab of the Plan Script, Plan Schedule, Script Properties, and Schedule Properties dialog boxes.

For example, you could change **Custom 1** to *Test Type* and add Test Type values, such as *UI*, *functional*, *performance*, and so on. You could change **Custom 2** to something that might indicate the complexity level or whether the script or schedule has any dependencies.

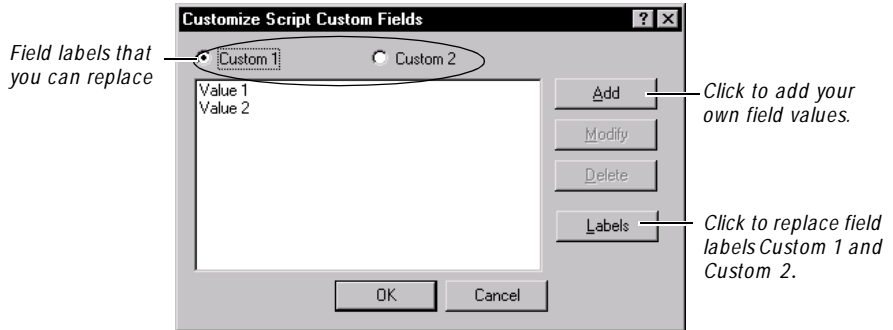


NOTE: The examples used in this section show how to customize field labels and values for your scripts. Use the same procedures to customize field labels and values for your schedules.

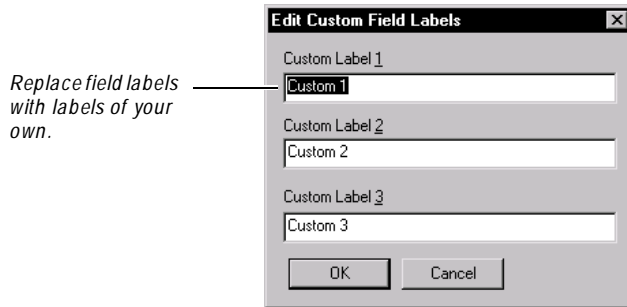
Defining Custom Field Labels

To define custom field labels for your scripts:

1. Click **Tools** → **Customize** → **Script Custom Fields**.



2. Click **Labels**.



3. Replace **Custom Label 1**, **Custom Label 2**, or **Custom Label 3** with your own field labels and click **OK** to return to the **Customize Script Custom Fields** dialog box.
4. Click **OK**.

Modifying Custom Field Values

To modify values for custom fields:

1. Click **Tools** → **Customize** → **Script Custom Fields**.
2. Click either **Custom 1** or **Custom 2**.
3. Click one of the default values, **Value 1** or **Value 2**.
4. Click **Modify**, type the new name for the value, and click **OK**.
5. Repeat steps 2 – 4 to modify other custom values.

Adding Custom Field Values

To add a value to a custom field:

1. Click **Tools** → **Customize** → **Script Custom Fields**.
2. Click either **Custom 1** or **Custom 2**.
3. Click **Add**, type the name for the new value, and click **OK**.
4. Repeat steps 2 and 3.
5. Click **OK**.

Planning Your Tests

▶▶▶ C H A P T E R 3

Managing Builds, Log Folders, and Logs

This chapter explains how to manage builds, log folders, and logs. It includes the following topics:

- ▶ Overview
- ▶ Displaying builds in the Asset Browser
- ▶ Creating a new build
- ▶ Copying, renaming, and deleting builds
- ▶ Renaming and deleting logs and log folders
- ▶ Displaying log properties
- ▶ Viewing logs
- ▶ Working with build states

Overview

After you complete the test planning phase of your project, you can begin the test development and analysis phases. In the test planning phase (described in Chapter 2, *Planning Your Tests*) you create test documents, define test requirements, and plan test scripts and schedules.

In the test development phase, you record the scripts and create the schedules that you planned in the planning phase. In the analysis phase, you play back the scripts, run the schedules, and analyze the results.

Before you start the test development and analysis phases, you should know how to use TestManager to organize both the software builds and the test logs that are generated during the test analysis phase.

Using Builds in Functional Testing

Functional tests compare the way an application-under-test *actually* behaves in the current build against its *expected* behavior as recorded and validated in a previous build. Typically, as software developers add new features or fix defects, a new build is generated. Because this process happens on a continual basis during application development, hundreds of builds may be generated.

You use TestManager to define and manage builds. You can assign properties such as a name, description, owner, and build state. Afterwards, you can copy, rename, or delete a particular build as necessary.

Generating Log Files

Your test team may develop hundreds of scripts to validate your test requirements. When you play back a script, Robot runs the script and generates a log file that reports the results. A **log** file contains the record of events that occurred during playback of a script. Each log file clearly specifies the pass/fail results of each verification point in the script, as shown in the following figure:

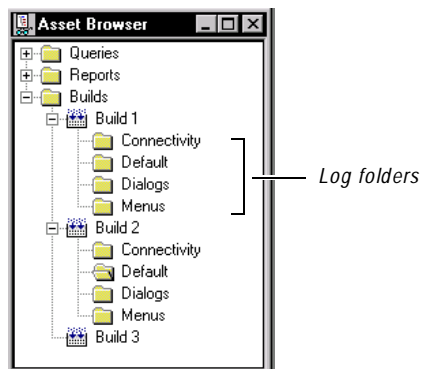
Log Event	Result	Date	Time	Defect
Script Start (QPtest4)	Fail	9/13/99	10:57:07 AM	
Start Application	Pass	9/13/99	10:57:07 AM	
Verification Point (Object Properties) - Object Properties	Pass	9/13/99	10:57:08 AM	
Verification Point (Object Properties2) - Object Properties	Fail	9/13/99	10:57:09 AM	
Script End (QPtest4)	Fail	9/13/99	10:57:09 AM	

Because you normally play back scripts against each build of the application-under-test, you can easily generate thousands of log files. You can use these log files to generate reports that will help you determine the quality of each build and the progress of your testing effort. For example, the Execution Coverage report tells you the percentage and number of tests that have passed or failed for a specific build. For information about reports, see Chapter 16, *Running TestManager Reports*.

Organizing Log Folders

To more easily manage your log files, you can create log folders. Each build folder in the Asset Browser contains one default log folder, named **Default**. You can store all of your log files in the Default folder, or you can create your own log folders. For information about adding log folders, see *Creating a New Build* on page 3-5.

You can name and organize your log folders in any way that makes sense for your project. One logical way to organize your log folders is to mirror the top-level organization of your Requirements Hierarchy. If your Requirements Hierarchy is organized according to functional area, for example, you could create log folders for each of those functional areas.



When you play back a script in Robot, Robot may prompt you to specify a particular build and log folder in which to store the log. For information about setting up Robot to prompt you for the build and log folder, see Chapter 9, *Playing Back GUI Scripts*.

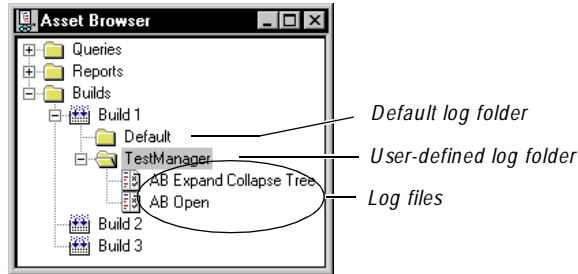
Naming Log Folders for LoadTest Users

NOTE: This feature requires Rational Suite PerformanceStudio.

When you run a LoadTest schedule, LoadTest prompts you to specify the name of the build and log folder in which to store the log file that is created. By default, LoadTest uses the name of the schedule for the log folder name. Typically, you store all of the logs for that schedule in the same folder.

Displaying Builds in the Asset Browser

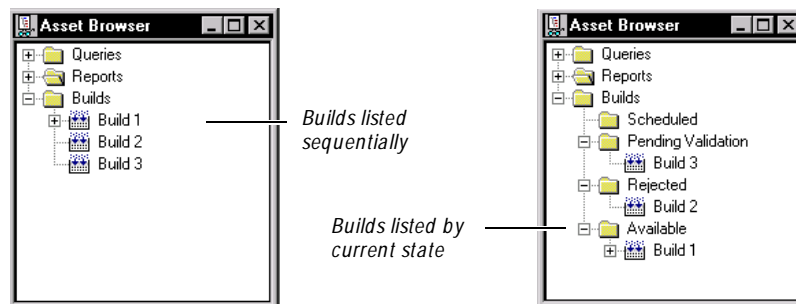
Builds are displayed as one of the top-level folders in the Asset Browser. The **Asset Browser** is the TestManager window that you can use to create, manage, and display builds, queries, and reports, as show in the following figure:



You can organize the way builds are displayed in the Asset Browser in one of two ways:

- ▶ **Sequentially** – Organized in alphanumeric order.
- ▶ **By State** – Organized within the build state to which they are assigned. For information about build states, see *Working with Build States* on page 3-9.

The following figure displays the build portion of the Asset Browser and compares the two ways to organize the folder:



To toggle between the two views:

1. Right-click a build folder or a specific build in the Asset Browser.
2. Click **List Builds Sequentially** or **List Builds by State**.

To define the default view:

1. Click **Tools** → **Options**.
2. Click the **Miscellaneous** tab.
3. Click **List builds sequentially** or **List builds by state**.

Creating a New Build

You can create new builds and copy, rename, or delete existing ones. When you create a new build, you can add important information about the build in the form of notes. For example, you can include release note information or warnings to testers.

To create a new build:

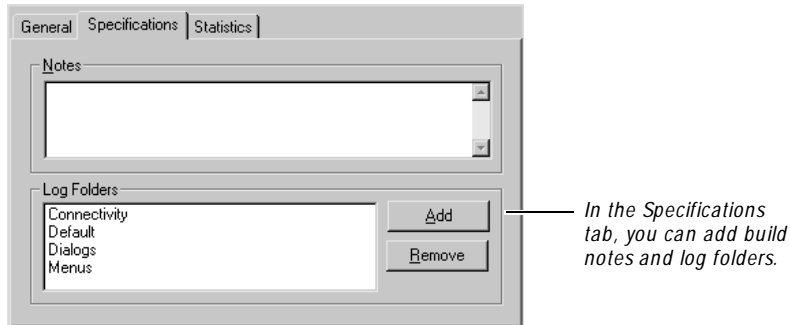
1. Do one of the following:
 - Right-click the Builds folder or any build states folder in the Asset Browser and click **New Build**. (For information about build states, see *Working with Build States* on page 3-9.)
 - Click **Tools** → **Manage Builds**, and then click **New**.

The image shows a 'Build Properties' dialog box with the following fields and options:

- Name:** A text input field.
- Description:** A text area with a scroll bar.
- Owner:** A dropdown menu with 'admin' selected.
- State:** A dropdown menu with '<None>' selected.
- Buttons:** OK, Cancel, and Help.

2. Type a name (40 characters maximum) for the build.
3. Type a description (255 characters maximum) of the build.
4. Select the owner responsible for the build. Owners are defined in the Rational Administrator. For details, see the *Using the Rational Administrator* manual.
5. Select the build state.

6. Click the **Specifications** tab.



7. Type any notes that are relevant to this particular build.
8. To add a log folder, click **Add**. Type the name of the new log folder and click **OK**.
9. Click **OK**.

NOTE: There are two other ways to create log folders. You can right-click a build in the Asset Browser and click **New Log Folder**, or you can create a new log folder from the dialog box that appears when you play back a script in Robot.

Copying, Renaming, and Deleting Builds

After you create a build, organize your log folders, and play back your scripts, you can copy the entire build folder in preparation for testing the next build.

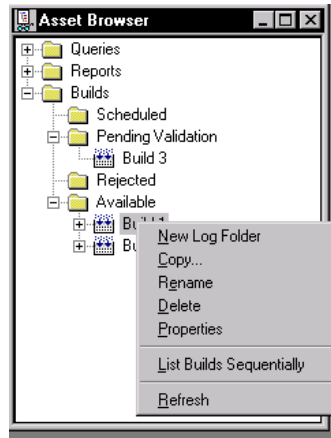
When you copy a build, you also copy the log folders that reside within the build folder. The logs themselves are not copied because they represent the test results for only one particular build.

In addition, you can rename or delete builds. For example, if a particular build has been approved as the beta release, you might want to rename it **Beta**. After the project is finished, you might want to archive your alpha, beta, and final builds and delete all of the other ones. When you delete a build, you delete everything contained within the build folder, including log folders and logs.

You can copy, rename, and delete builds from the Asset Browser or from the Manage Builds dialog box.

To copy, rename, or delete builds from the Asset Browser:

1. Right-click a build in the Asset Browser.

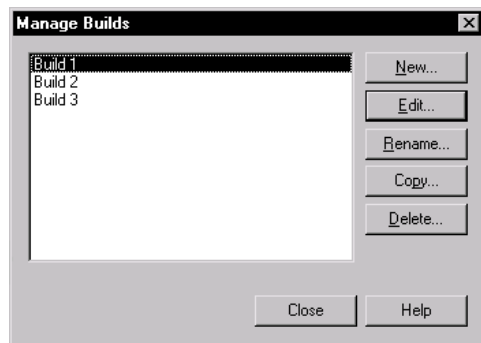


2. Do one of the following:

- Click **Copy**, type a name for the new build, and click **OK**.
- Click **Rename**, type the new name for the build, and press **ENTER**.
- Click **Delete**, and then click **Yes** to delete the build.

To copy, rename, or delete builds from the Manage Builds dialog box:

1. Click **Tools** → **Manage Builds**.



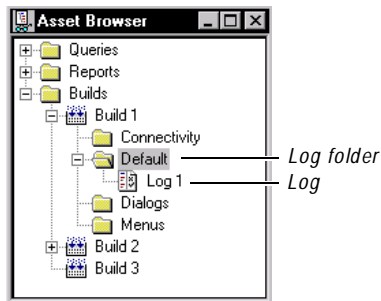
2. Select a build from the list and do one of the following:
 - Click **Rename**, type the new name for the build, and click **OK**.
 - Click **Copy**, type a name for the new build, and click **OK**.
 - Click **Delete**, and then click **Yes** to delete the build.
3. Click **Close**.

Renaming and Deleting Logs and Log Folders

When you play back a script in Robot or run a schedule in LoadTest, you assign a name to the log file that is created. By default, a log has the same name as the script or schedule. Later, you can rename the log or log folder from the Asset Browser in TestManager. You can also delete logs and log folders from the Asset Browser.

To rename or delete logs and log folders:

1. Right-click a log or log folder in the Asset Browser.



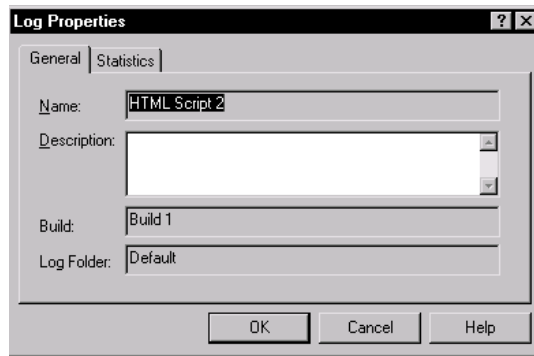
2. Do one of the following:
 - Click **Rename**, type the new name for the log or log folder, and press **ENTER**.
 - Click **Delete**, and then click **Yes** to delete the log or log folder.

Displaying Log Properties

Log properties include a name and description, as well as the build and log folder to which the log belongs.

To display log properties:

1. Right-click a log in the Asset Browser.
2. Click **Properties**.



3. Optionally, you can edit the log's description.

Viewing Logs

To view a log in the LogViewer:

- ▶ Double-click a log in the Asset Browser, or right-click a log in the Asset Browser and click **Open**.

For information about logs, see Chapter 10, *Reviewing Logs with the LogViewer*.

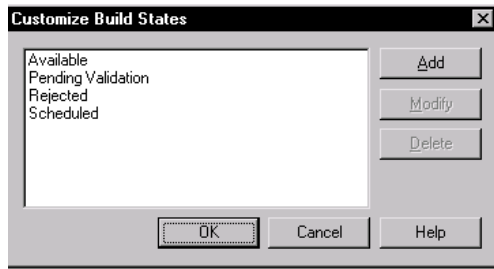
Working with Build States

You can organize the builds in the Asset Browser sequentially or by build state. **Build states** provide a way to organize builds within a build hierarchy. When you organize by build state, you can use the default build states that are provided (Available, Pending Validation, Rejected, and Scheduled), or you can create your own. You can also modify or delete the existing build states.

Managing Builds, Log Folders, and Logs

To add a new build state:

1. Click **Tools** → **Customize** → **Build States**.



2. Click **Add**. Type a build state, and then click **OK** to add it to the list.
3. Click **OK**.

To modify or delete an existing build state:

1. Click **Tools** → **Customize** → **Build States**.
2. Select a build state from the list.
3. Do one of the following:
 - Click **Modify**, type a new name for the build state, and then click **OK**.
 - Click **Delete**.
4. Click **OK**.

▶ ▶ ▶ Part III

Developing Tests

▶▶▶ CHAPTER 4

Recording GUI Scripts

This chapter describes the recording process and tells you how to record GUI scripts in Rational Robot. It includes the following topics:

- ▶ The recording process
- ▶ The recording workflow
- ▶ Before you begin recording
- ▶ Enabling IDE applications for testing
- ▶ Setting GUI recording options
- ▶ Using advanced features before recording
- ▶ Recording a new GUI script
- ▶ Defining script properties
- ▶ Coding a GUI script manually
- ▶ Testing your recorded script
- ▶ Creating shell scripts to play back scripts in sequence

NOTE: For information about recording virtual user scripts, see the *Using Rational LoadTest* manual.

The Recording Process

When you record a GUI script, Robot records:

- ▶ Your actions as you use the application-under-test. These **user actions** include keystrokes and mouse clicks that help you navigate through the application.

- ▶ Verification points that you insert to capture and save information about specific objects. A **verification point** is a point in a script that you create to confirm the state of an object across builds. During recording, the verification point captures object information and stores it as the baseline. During playback, the verification point recaptures the object information and compares it to the baseline.

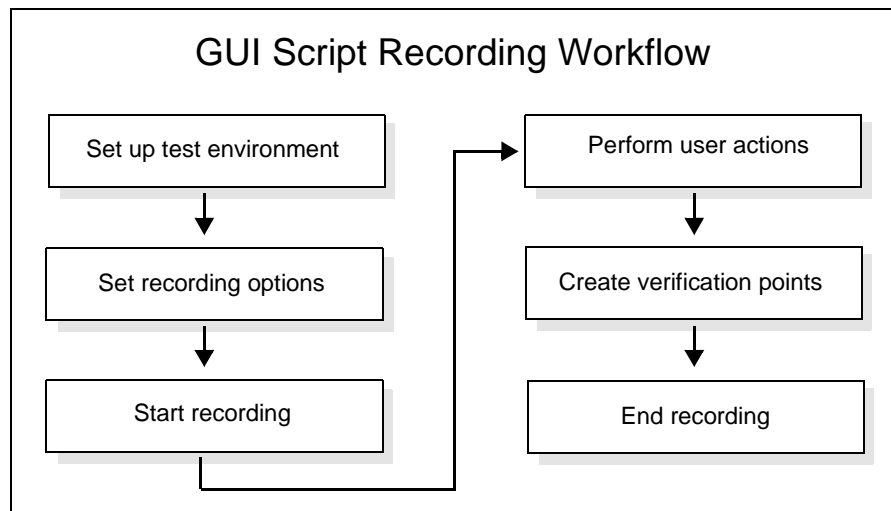
The recorded GUI script establishes the baseline of expected behavior for the application-under-test. When new builds of the application become available, you can play back the script to test the builds against the established baseline in a fraction of the time that it would take to perform the testing manually.

The Recording Workflow

Typically, when you record a GUI script, your goal is to:

- ▶ Record actions that an actual user might perform (for example, clicking a menu command or selecting a check box).
- ▶ Create verification points to confirm the state of objects across builds of the application-under-test (for example, the properties of an object or the text in an entry field).

The following figure outlines the general process for recording a GUI script.



Before You Begin Recording

You should plan to use Robot at the earliest stages of the application development and testing process. If any Windows GUI objects such as menus and dialog boxes exist within the initial builds of your application, you can use Robot to record the corresponding verification points.

Consider the following guidelines before you begin recording:

- ▶ Establish predictable start and end states for your scripts.
- ▶ Set up your test environment.
- ▶ Create modular scripts.
- ▶ Plan your scripts in Rational TestManager.

These guidelines are described in more detail in the following sections.

Establishing Predictable Start and End States for Scripts

By starting and ending the recording at a common point, scripts can be played back in any order, with no script being dependent on where another script ends. For example, you can start and end each script at the Windows desktop or at the main window of the application-under-test.

Setting Up Your Test Environment

Any windows that are open, active, or displayed when you begin recording should be open, active, or displayed when you stop recording. This applies to all applications, including Windows Explorer, e-mail, and so on.

Robot can record the sizes and positions of all open windows when you start recording, based on the recording options settings. (For information about setting the recording options, see *Setting GUI Recording Options* on page 4-6.) During playback, Robot attempts to restore windows to their recorded states, and inserts a warning in the log if it cannot find a recorded window.

In general, close any unnecessary applications before you start to record. For stress testing, however, you may want to deliberately increase the load on the test environment by having many applications open.

Creating Modular Scripts

Rather than defining a long sequence of actions in one GUI script, you should define scripts that are short and modular. Keep your scripts focused on a specific area of testing — for example, on one dialog box or on a related set of recurring actions.

When you need more comprehensive testing, modular scripts can easily be called from or copied into other scripts. They can also be grouped into **shell scripts**, which are top-level, ordered groups of scripts.

The benefits of modular scripts are:

- ▶ They can be called, copied, or combined into shell scripts.
- ▶ They can be easily modified or re-recorded if the developers make intentional changes to the application-under-test.
- ▶ They are easier to debug.

Planning Scripts in TestManager

Planning scripts and defining script properties are important parts of the test planning process. You typically define a script's properties in TestManager *before* you record the script in Robot. You can then start Robot, select a planned script, and start recording.

Script properties include:

- ▶ The script name, description, owner, purpose, and test environment.
- ▶ Related assets such as test requirements.
- ▶ Notes and specification files.
- ▶ Custom keywords.

To plan scripts and define properties before you record the script, open TestManager and click **File** → **Plan** → **GUI Script**. For more information, see *Planning Scripts* on page 2-10.

Use this dialog box to define script properties in TestManager before you record scripts.

The screenshot shows the 'Plan Script' dialog box with the following fields and options:

- Name:** [Empty text box]
- Description:** [Empty text box with scrollbars]
- Owner:** [Dropdown menu showing 'admin']
- Purpose:** [Dropdown menu showing '<None>']
- Environment:** [Dropdown menu showing '<None>']
- Type:** GUI Developed

Buttons at the bottom: OK, Cancel, Help.

You can also define or change these properties in Robot when you begin recording the script or after you record the script. For information, see *Defining Script Properties* on page 4-24.

Enabling IDE Applications for Testing

Robot provides specialized support for testing the objects in applications that are created in many integrated development environments (IDEs).

To successfully test the objects in Oracle Forms, HTML, Java, C++ , and Visual Basic 4.0 applications, you need to enable the applications as follows before you start recording your scripts:

- ▶ **Oracle Forms** – Install the Rational Test Enabler for Oracle Forms. Run the Enabler to have it add the Rational Test Object Testing Library and three triggers to the .fmb files of the application. For information, see Chapter 18, *Testing Oracle Forms Applications*.
- ▶ **HTML** – While recording or editing a script, use the **Start Browser** toolbar button to start Internet Explorer from Robot. This loads the Rational ActiveX Test Control, which lets Robot recognize Web-based objects. For information, see Chapter 19, *Testing HTML Applications*.
- ▶ **Java** – Run the Java Enabler to have it scan your hard drive for Java environments such as Web browsers and Sun JDK installations that Robot supports. The Java Enabler only enables those environments that are currently installed. For information, see Chapter 20, *Testing Java Applets and Applications*.
- ▶ **C/C++** – To test the properties and data of ActiveX controls in your applications, install the Rational ActiveX Test Control. This is a small, non-intrusive custom control that acts as a gateway between Robot and your application. It has no impact on the behavior or performance of your application and is not visible at runtime. Manually add the ActiveX Test Control to each OLE container (Window) in your application. For instructions, see the documentation that comes with your C/C++ development environment.
- ▶ **Visual Basic 4.0** – Install the Rational Test Enabler for Visual Basic. Attach the Enabler to Visual Basic as an add-in. Have the Enabler add the Rational ActiveX Test Control to every form in the application. This is a small, non-intrusive custom control that acts as a gateway between Robot and your application. For information, see *Visual Basic support, making Visual Basic applications testable* in the Robot Help Index.

You can install the Enablers and the ActiveX Test Control from the Rational Software Setup wizard. For instructions, see one of the following manuals: *Installing Rational Suite* or *Installing Rational TeamTest and Rational Robot*.

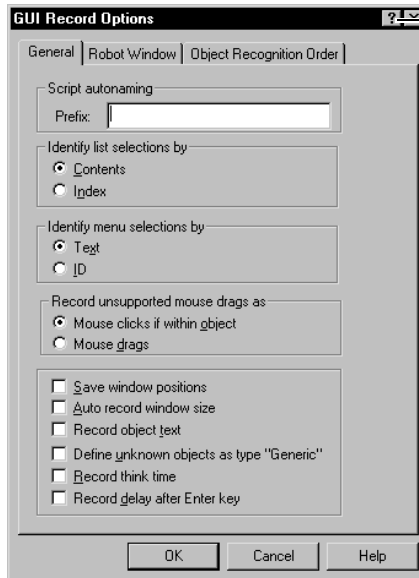
NOTE: You do not need to enable applications created in Visual Basic 5.0 and later, PowerBuilder, or PeopleTools.

Setting GUI Recording Options

GUI recording options provide instructions to Robot about how to record and generate GUI scripts. You can set these options either before you begin recording or early in the recording process.

To set the GUI recording options:

1. Open the GUI Record Options dialog box by doing one of the following:
 - Before you start recording, click **Tools** → **GUI Record Options**.
 - Start recording by clicking the **Record GUI Script** button on the toolbar. In the Record GUI dialog box, click **Options**.



For detailed information about an item, click the question mark, and then click the item.

2. Set the options on each tab.
3. Click **OK**.

Naming Scripts Automatically

Robot can assist you in assigning names to scripts with its script autonaming feature. Autonaming inserts your specified characters into the **Name** box of a new script and appends a consecutive number to the prefix.

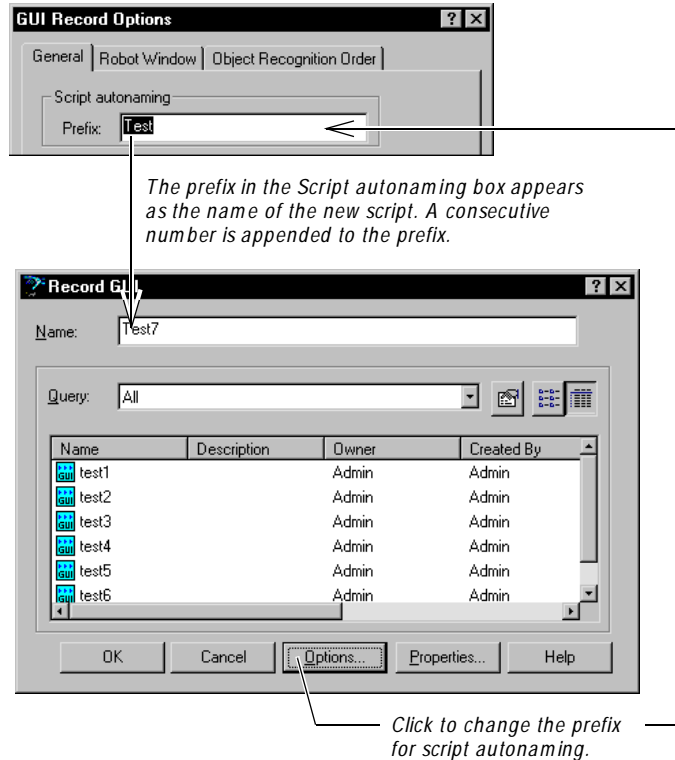
This is a useful feature if you are recording a series of related scripts and want to identify their relationship through the prefix in their names. For example, if you are testing the menus in a Visual Basic application, you might want to have every script name start with VBMenu.

To turn on script autonaming:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 4-6.)
2. In the **General** tab, type a prefix in the **Prefix** box.
Clear the box if you do not want a prefix. If the box is cleared, you will need to type a name each time you record a new script.
3. Click **OK** or change other options.

The next time you record a new script, the prefix and a number appear in the **Name** box of the Record GUI dialog box.

In the following figure, the autonaming prefix is *Test*. When you record a new script, *Test7* appears in the **Name** box because there are six other scripts that begin with *Test*.



If you change the script autonaming prefix by clicking **Options** in the Record GUI dialog box, changing the prefix, and then clicking **OK**, the name in the **Name** box changes immediately.

Controlling How Robot Responds to Unknown Objects

During recording, Robot recognizes all standard Windows GUI objects that you click, such as check boxes and list boxes. Each of these objects is associated with one of a fixed list of object types. The association of an object with an object type is generally based on the class name of the window associated with the object.

Robot also recognizes many custom objects defined by IDEs that Robot supports, such as Visual Basic, Oracle Forms, Java, and HTML. For example, if you click a Visual Basic check box, Robot recognizes it as a standard Windows check box. This mapping is based on the object's Visual Basic assigned class name of `ThunderCheckBox`.

These **built-in object mappings** are delivered with Robot and are available to all users no matter which repository they are using.

During recording, you might click an object that Robot does *not* recognize. In this case, Robot's behavior is controlled by a recording option that you set. You can have Robot either:

- ▶ Open the Define Object dialog box, so that you can map the object to a known object type.

Mapping an object to an object type permanently associates the class name of the object's window with that object type, so that other objects of that type will be recognized. For more information, see *Defining Unknown Objects During Recording* on page 4-21.

- ▶ Automatically map unknown objects encountered while recording with the **Generic** object type. This permanently associates the class name of the unknown object's window with the Generic object type.

This is a useful setting if you are testing an application that was written in an IDE for which Robot does not have special support and which therefore might contain many unknown objects. When an object is mapped to the Generic object type, Robot can test a basic set of its properties, but it cannot test the special properties associated with a specific object type. Robot also records the object's *x,y* coordinates instead of using the more reliable object recognition methods to identify the object. (For information about the recognition methods, see the following section, *Selecting an Object Order Preference*.)

These **custom object mappings** are stored in the repository that was active when the mappings were created.

To control how Robot behaves when it encounters an unknown object during recording:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 4-6.)
2. In the **General** tab, do one of the following:
 - Select **Define unknown objects as type "Generic"** to have Robot automatically associate unknown objects encountered while recording with the Generic object type.
 - Clear **Define unknown objects as type "Generic"** to have Robot suspend recording and open the Define Object dialog box if it encounters an unknown object during recording. Use this dialog box to associate the object with an object type.
3. Click **OK** or change other options.

You can also map object types and classes *before* you start recording. For information, see *Mapping Object Types and Classes Before Recording* on page 4-14.

NOTE: The custom mapping from class name to object type is stored in the repository and is shared among all users of the repository.

Selecting an Object Order Preference

Robot uses a variety of **object recognition methods** to uniquely identify objects in the application-under-test that are acted on during recording sessions. For example, Robot can identify a check box in the application-under-test by its object name, associated label or text string, index value, or ID value.

These recognition methods are saved as arguments in script commands so that Robot can correctly identify the same objects during playback.

Robot has two predefined preferences for the recognition method order for each standard object type. While recording an action on an object, Robot tries each method within the selected preference in sequence until it finds a method that uniquely identifies the object.

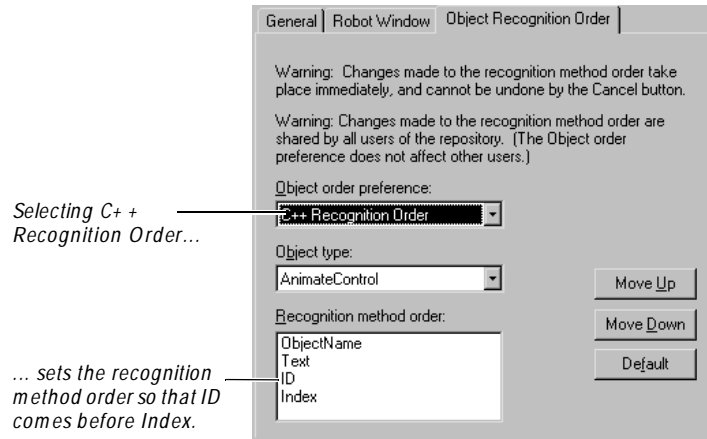
The following table describes the two predefined preferences.

Object order preference	Recognition method order	Comments
< Default >	Object Name Label and/or Text Index ID	Index comes before ID. In some environments, such as PowerBuilder and Visual Basic, the ID changes each time the developer creates an executable file and is therefore not a good recognition method.
C++ Recognition Order	Object Name Label and/or Text ID Index	ID comes before index. In some environments, such as C++, the ID does not usually change and is therefore a good recognition method.

The < Default > object order preference is the initial setting. If you plan to test C++ applications, change the preference to C++ Recognition Order.

To change the object order preference:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 4-6.)
2. Click the **Object Recognition Order** tab.
3. Select a preference in the **Object order preference** list.



4. Click **OK** or change other options.

NOTE: The object order preference is specific to each user. For example, you can record with C++ preferences while another user is recording with <Default> preferences at the same time.

For information about changing the order of the recognition methods within an object order preference, see *Customizing the Object Recognition Method Order* on page 4-12.

Using Advanced Features Before Recording

In addition to setting the standard GUI recording options, you can take some additional steps to refine your testing. You can:

- ▶ Customize the order of the object recognition methods to make the script more readable and stable.
- ▶ Map object types and classes to identify custom objects during record and playback.

Customizing the Object Recognition Method Order

As explained in the previous section, Robot has two predefined preferences for the recognition method order for each standard object type: < Default> and C++ Recognition Order. When you record an action on an object, Robot tries each method within the selected preference in sequence until it finds one that uniquely identifies the object.

You can redefine the order in which Robot tries recognition methods for each object type. This order has an effect on both the readability and stability of script commands. For example, when you read script files, it is easier to locate a command on a specific object if that command uses the object name or label for identification. However, if the object name or label is likely to change between builds, another recognition method may provide more stability.

You should evaluate your own development and testing environment before you change the default order of object recognition methods.

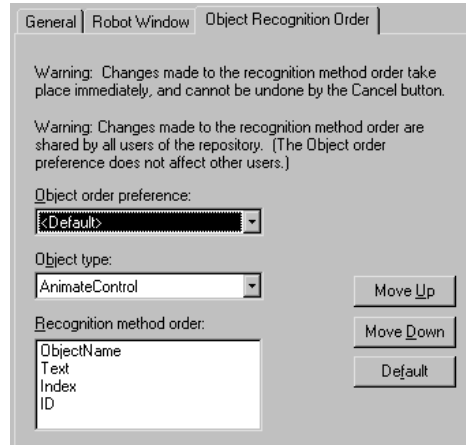
Important Notes

- ▶ Changes to the recognition method order affect scripts that are recorded after the change. They do not affect the playback of scripts that have already been recorded.
- ▶ Changes to the recognition method order are stored in the repository. For example, if you change the order for the CheckBox object, the new order is stored in the repository and affects all users of that repository.
- ▶ Changes to the order for an object affect only the currently-selected preference. For example, if you change the order for the CheckBox object in the < Default> preference, the order is not changed in the C++ preference.

Changing the Order of Object Recognition Methods

To change the order of the object recognition methods for an object type:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 4-6.)
2. Click the **Object Recognition Order** tab.



3. Select a preference in the **Object order preference** list.

If you will be testing C++ applications, change the object order preference to **C++ Recognition Order**.

4. From the **Object type** list, select the object type to modify.

The fixed set of recognition methods for the selected object type appears in the **Recognition method order** list in its last saved order.

5. Select an object recognition method in the list, and then click **Move Up** or **Move Down**.

Changes made to the recognition method order take place immediately, and cannot be undone by the **Cancel** button. To restore the original default order, click **Default**.

6. Click **OK**.

NOTE: Changes to the recognition method order are stored in the repository. For example, if you change the order for the **CheckBox** object, the new order is stored in the repository and affects all users of that repository.

Creating a New Object Order Preference

Robot has two predefined object order preferences: < Default> and C++ Recognition Order. You can create additional preferences to handle special situations.

To create a new object order preference:

1. In an ASCII editor, create an empty text file with the extension **.ord**.
2. Save the file in the Dat folder of the repository.
3. Click **Tools** → **GUI Record Options**.
4. Click the **Object Recognition Order** tab.
5. From the **Object order preferences** list, select the name of the file you created.
6. Change the method order to customize your preferences.

Mapping Object Types and Classes Before Recording

As explained in *Controlling How Robot Responds to Unknown Objects* on page 4-8, Robot recognizes all standard Windows GUI objects and many custom objects. You can also set a recording option so that Robot either automatically maps unrecognized objects to the Generic object type, or stops during recording so that you can map the object to a standard object type.

If you know in advance that the application-under-test contains a custom object or any object that Robot does not recognize, you can create a custom object mapping *before* you start recording. You do this by adding the object's class to the list of classes that Robot recognizes, and then associating the class to a standard object type. Robot saves this custom class/object-type mapping in the repository and uses it to identify the custom object during playback.

NOTE: The custom mapping from class name to object type is stored in the repository and is shared among all users of the repository. Be careful about changing existing mappings because this may cause already-recorded scripts to play back incorrectly.

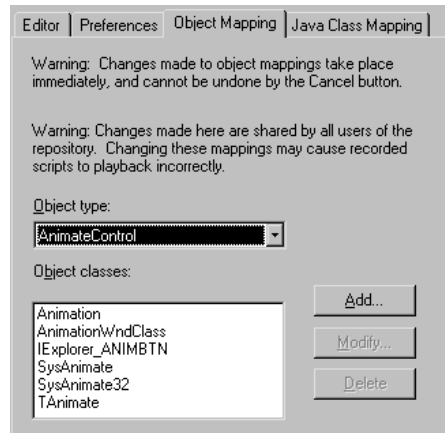
Defining an Object Class Mapping

To define an object class and map an object type to it:

1. Identify the class name of the window that corresponds to the object.

You can use the Spy++ utility in Visual C++ to identify the class name. You can also use the Robot Inspector tool by clicking **Tools** → **Inspector**.

- In Robot, click **Tools** → **General Options**, and then click the **Object Mapping** tab.



- From the **Object type** list, select the standard object type to be associated with the new object class name.

Robot displays the class names already available for that object type in the **Object classes** list box.

- Click **Add**.
- Type the class name you identified in step 1 and click **OK**.
- Click **OK**.

NOTE: An object class can be mapped to only one object type. If you try to map an object class to more than one object type, a message asks you to confirm that you want to remap the class.

Modifying or Deleting a Custom Class Name

To modify or delete a custom class name:

- Click **Tools** → **General Options**, and then click the **Object Mapping** tab.
- From the **Object type** list, select the standard object type that is associated with the object class name.

Robot displays the class names already available for that object type in the **Object classes** list.

- From the **Object classes** list, select the name to modify or delete.

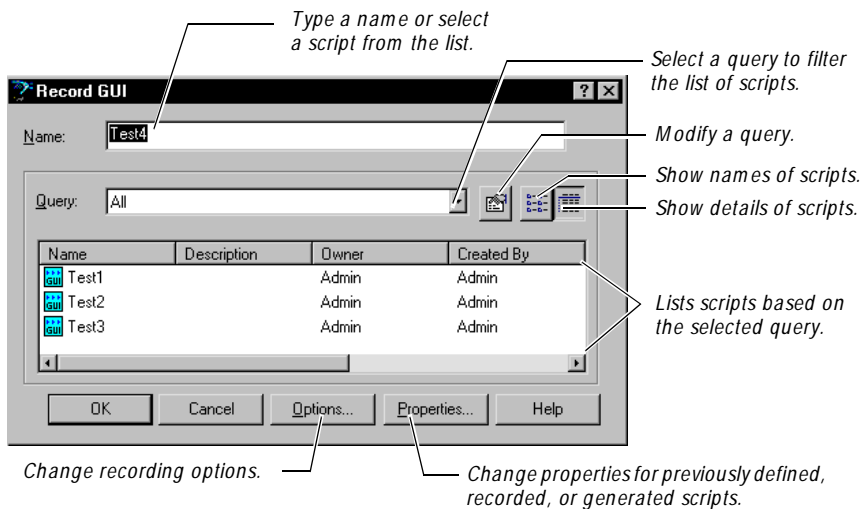
4. Do one of the following:
 - To modify the class name, click **Modify**. Change the name and click **OK**.
 - To delete the object class mapping, click **Delete**. Click **OK** at the confirmation prompt.
5. Click **OK**.

NOTE: You cannot modify or delete a built-in class name.

Recording a New GUI Script

To record a GUI script:

1. Prepare to record the script. (See *Before You Begin Recording* on page 4-3.)
2. If necessary, enable your application for testing. (See *Enabling IDE Applications for Testing* on page 4-5.)
3. Make sure your recording options are set appropriately for the recording session. (See *Setting GUI Recording Options* on page 4-6.)
4. Click the **Record GUI Script** button on the toolbar to open the Record GUI dialog box.



5. Type a name (40 characters maximum) or select a script from the list.

The listed scripts have already been defined in TestManager, recorded in Robot, or generated in TestFactory. To change the list, select a query from the **Query** list. The query lets you narrow down the displayed list, which is useful in projects with hundreds of scripts. You create queries in TestManager, and you modify queries in TestManager or Robot. (For information about queries, see Chapter 15, *Querying the Rational Repository*.)

If a prefix has been defined for script autonaming, Robot displays the prefix in the **Name** box. To edit this name, either type in the **Name** box, or click **Options**, change the prefix in the **Prefix** box, and click **OK**. (For more information, see *Naming Scripts Automatically* on page 4-7.)

6. To change the recording options, click **Options**. When finished, click **OK**.
7. If you selected a previously defined or recorded script, you can change the properties by clicking **Properties**. When finished, click **OK**.

To change the properties of a *new* script, record the script first. After recording, click **File** → **Properties**. (For more information, see *Defining Script Properties* on page 4-24.)

8. Click **OK** to start recording. The following events occur:
 - If you selected a script that has already been recorded, Robot asks if you want to overwrite it. Click **Yes**. (If you record over a previously-recorded script, you overwrite the script file but any existing properties are applied to the new script.)
 - Robot is minimized by default. (For information, see *Restoring the Robot Main Window During Recording* on page 4-19.)
 - The floating GUI Record toolbar appears. You can use this toolbar to pause or stop recording, display Robot, and insert features into a script. (For more information, see *Using the GUI Record and GUI Insert Toolbars* on page 4-20.)
9. Start the application-under-test as follows:
 - a. Click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - b. Click the appropriate **Start** button on the GUI Insert toolbar.
 - c. Fill in the dialog box and click **OK**.



NOTE: It is essential that you start the application correctly, depending on the type of application and how you plan to play it back. For information, see *Starting an Application* on page 5-1.

10. Perform actions as needed to navigate through the application.
11. Insert features as needed. You can insert features such as verification points, comments, and timers. (For information, see Chapter 5, *Adding Features to GUI Scripts*.)
12. If necessary, switch from Object-Oriented Recording to low-level recording. (For information, see *Switching to Low-Level Recording* on page 4-22.)

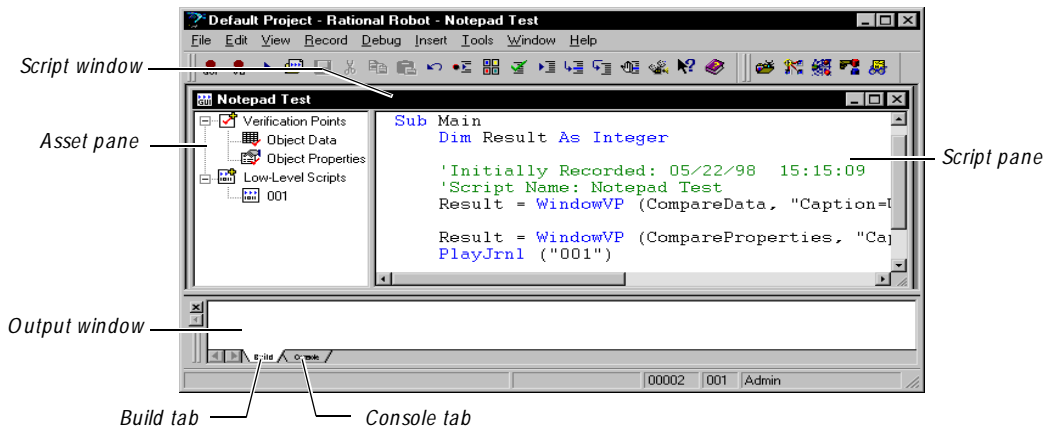
Object-Oriented Recording examines Windows GUI objects and other objects in the application-under-test without depending on precise timing or screen coordinates. **Low-level recording** tracks detailed mouse movements and keyboard actions by screen coordinates and exact timing.



13. When finished, click the **Stop Recording** button on the GUI Record toolbar.

The Robot main window appears as follows:

- The script that you recorded appears in a Script window within the Robot main window.
- The verification points and low-level scripts in the script (if any) appear in the Asset pane on the left.
- The text of the script appears in the Script pane on the right.



NOTE: The **Build** tab of the Output window shows compilation results when you compile or play back a script. (For information, see *Compiling Scripts and SQABasic Library Source Files* on page 7-7.) The **Console** tab of the Output window is reserved for your messages. (For information, see the *SQABasic Language Reference*.)

14. Optionally, change the script properties by clicking **File** → **Properties**. (For information, see *Defining Script Properties* on page 4-24.)

Restoring the Robot Main Window During Recording

When you begin recording, the Robot main window becomes minimized by default, allowing you unobstructed access to the application-under-test.

At any time during recording, you can restore the Robot window without affecting the script you are recording. For example, you might want to restore the Robot window to reset your recording options.

When Robot is minimized or is hidden behind other windows during recording, you can bring it to the foreground in any of the following ways:

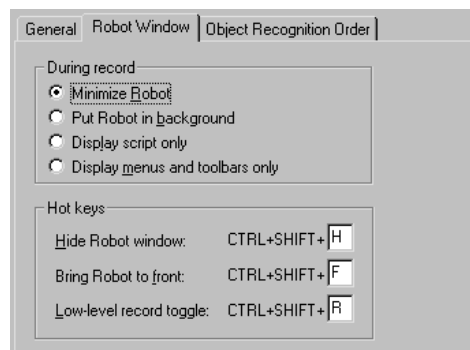


- ▶ Click the **Open Robot Window** button on the GUI Record toolbar.
- ▶ Click the Robot button on the Windows taskbar.
- ▶ Use the hot key combination CTRL+ SHIFT+ F to display the window and CTRL+ SHIFT+ H to hide the window.

You can also use the standard Windows ALT+ KEY combination.

To change the default behavior of the Robot main window and the default hot keys:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 4-6.)
2. Click the **Robot Window** tab.

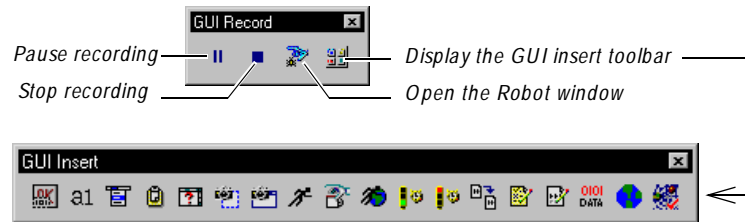


3. Select an option under **During record**.
4. Change the letter of a hot key under **Hot keys**.
5. Click **OK**.

Using the GUI Record and GUI Insert Toolbars

When you begin to record a GUI script, Robot displays the floating GUI Record toolbar. This toolbar gives you quick access to activities you might want to perform during recording.

If you click the rightmost button on the GUI Record toolbar, the GUI Insert toolbar appears. Use this toolbar to insert features (such as verification points, timers, and comments) into the script.




For information about customizing the toolbars, see Appendix A, *Working With Toolbars*.

Pausing and Resuming the Recording of a Script


During recording, if you click an enabled Robot toolbar button or menu command (for example, **Tools** → **GUI Record Options**), Robot pauses the recording. After Robot completes your action (for example, after you click **OK** in the dialog box), recording resumes and you can continue working with the application-under-test.

You can also pause recording manually. For example, if you need to check your e-mail, you can pause recording so that the mouse clicks and keystrokes are not recorded as part of the script.

To pause recording:

- ▶  Click the **Pause** button on the GUI Record toolbar. Robot indicates a paused state by:
 - Depressing the **Pause** button.
 - Displaying *Recording Suspended* in the status bar.
 - Displaying a check mark next to the **Record** → **Pause** command.

To resume recording:

- ▶  Click **Pause** again.

Always resume recording with the application-under-test in the same state that it was in when you paused.

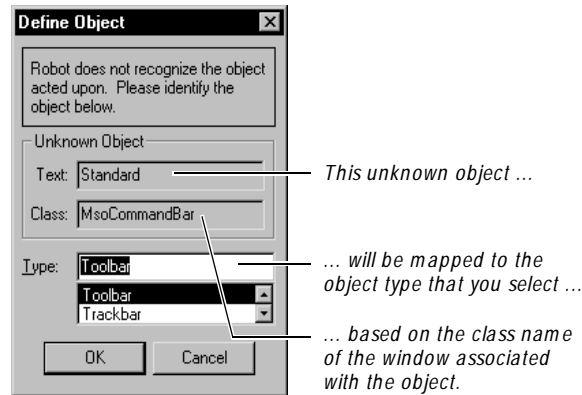
Defining Unknown Objects During Recording

As explained in *Controlling How Robot Responds to Unknown Objects* on page 4-8, Robot recognizes all standard Windows GUI objects and many custom objects. You can also set a recording option so Robot automatically associates unrecognized objects with the Generic object type.

If you have not set this option, Robot displays the Define Object dialog box if you click an object that Robot does not recognize. Use this dialog box to map the object to a known object type.

To define an unknown object while recording:

1. From the **Type** list in the Define Object dialog box, select an object type to associate with the unknown object.



If possible, select an object type that is appropriate for the object you are defining. For example, if the unknown object is a custom toolbar that has the same behavior as a standard Windows toolbar and supports the same programmatic interface, select **Toolbar** from the **Type** list. By mapping the object to a known object type, you will make your script more readable and Robot will be able to test the special properties associated with that object type. Also, Robot will be able to identify the object more accurately by using the object recognition methods.

However, using an incorrect object mapping can cause problems during playback. For example, an object might look and act like a standard toolbar but might actually not respond correctly to the messages that are sent to a standard toolbar. If you are not sure which type to use, select **Generic**. Robot will be able to test the basic set of the object's properties, and will use the object's *x,y* coordinates to locate the object.

2. Click **OK** to continue recording.

Robot stores the mapping between the window class name and the object type in the repository in case the same object type is captured again.

Important Notes

- ▶ If you want Robot to automatically define unknown objects as Generic during recording, click **Tools** → **GUI Record Options**, click the **General** tab, and select **Define unknown objects as type "Generic"**. (For more information, see *Controlling How Robot Responds to Unknown Objects* on page 4-8.)
- ▶ If you know in advance that the application-under-test contains an object that Robot will not recognize, you can map the class name of the object's window to a standard object type *before* recording. Robot saves this custom class/object-type mapping in the repository and uses it to identify the custom object during playback. (For more information, see *Mapping Object Types and Classes Before Recording* on page 4-14.)

Switching to Low-Level Recording

Robot has two recording modes:

- ▶ **Object-Oriented Recording mode** – Examines objects in the application-under-test at the Windows layer during recording and playback. Robot uses internal object names to identify objects, instead of using mouse movements or absolute screen coordinates. If objects in your application's graphical user interface (GUI) change locations, your tests still pass because the scripts are not location dependent. As a result, Object-Oriented Recording insulates the GUI script from minor user interface changes and simplifies GUI script maintenance.
- ▶ **Low-level recording mode** – Tracks detailed mouse movements and keyboard actions by screen coordinates and exact timing. Use low-level recording when you are testing functionality that requires the tracking of detailed mouse actions, such as in painting, drawing, or CAD applications.

To switch between the two modes during recording, do one of the following:

- ▶ Press **CTRL+ SHIFT+ R**.
- ▶ Click the **Open Robot Window** button on the GUI Record toolbar (or press **CTRL+ SHIFT+ F**) to bring Robot to the foreground. Click **Record** → **Turn Low-Level Recording On/Off**.

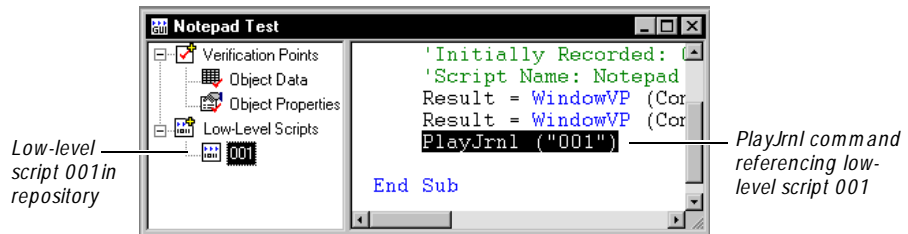


NOTE: To redefine hot keys, click **Tools** → **GUI Record Options**, click the **Robot Window** tab, and type the letter for the hot key.

When you switch to low-level recording mode, Robot does the following:

- ▶ Records low-level actions in a binary script file that cannot be edited, and stores this file in the repository.
- ▶ Adds a `PlayJrnl` command to your script that references the low-level script file.

Robot gives each low-level script a consecutive number. These numbers appear in the Asset pane in the Script window, under **Low-Level Scripts**.



To view the contents of the low-level binary file, double-click the file in the Asset pane. This displays an ASCII version of the binary file in Notepad. The file lists the actions that occurred during low-level recording. (For more information, see *Working with Low-Level Scripts* on page 7-3.)

Ending the Recording of a GUI Script

You should finish recording by returning the application-under-test to the same state it was in when recording began. This lets you play back the script without manually resetting the environment.

If you started recording from the Windows desktop, stop recording at the desktop. If you started recording from the main window of the application, stop recording at the main window, making sure that the window is in the same state as it was in when you started recording. For example, if the application is an editor and it had no documents open when you started recording, make sure that no documents are open when you stop recording.

To end the recording of a script:

- ▶ Click the **Stop Recording** button on the GUI Record toolbar.



Defining Script Properties

Defining script properties is an important part of the test planning process. For that reason, you typically define a script's properties in TestManager before you record the script. For more information, see *Planning Scripts* on page 2-10.

Script properties include:

- ▶ The script's name, description, owner, purpose, and test environment.
- ▶ Related assets such as test requirements.
- ▶ Notes and specification files.
- ▶ Custom keywords.

You can also define or edit these properties *after* you record the script in Robot.

To define script properties in Robot:

1. Do one of the following:
 - If the script is open, click **File** → **Properties**.
 - If the script is not open, click **File** → **Open** → **Script**. Select the script and click the **Properties** button.
2. In the Script Properties dialog box, define the properties. (For more information, see *Planning Scripts* on page 2-10.)

For detailed information about an item, click the question mark near the upper-right corner of the dialog box, and then click the item.

3. Click **OK**.

If you record over an existing GUI script, you overwrite the script file, but any existing properties are applied to the new script.

Coding a GUI Script Manually

By far, the fastest and easiest way to generate a GUI script is to let Robot record your actions and generate the script automatically. However, you can also hand-code a GUI script using the SQABasic scripting language.

To code a script manually:

1. In Robot, click **File** → **New** → **Script**.
2. Type a script name (40 characters maximum) and, optionally, a description of the script.

3. Click **GUI**.
4. Click **OK**. Robot creates an empty script with the following lines:

```
Sub Main
  Dim Result As Integer
  'Initially Recorded: 01/17/00 14:55:53
  'Script Name: GUI Script
End Sub
```

5. Begin coding the GUI script.

For information about using the SQABasic scripting language, see the *SQABasic Language Reference*. (In Robot, click **Help** → **SQABasic Reference**.)

Testing Your Recorded Script

After you record a script, you can:

- ▶ Play it back using the same version of the application-under-test.
- ▶ Edit and compile it.
- ▶ Debug it.

These steps are described briefly in the following sections.

Playing Back the Script

After you record a script, play it back to verify that it works as intended. Use the same build of the application-under-test that you used to record the script. After you play back the script, Robot writes the results to a log. Use the Rational LogViewer to view the log. The results should validate the baseline of expected behavior for the application-under-test.

For more information, see Chapter 9, *Playing Back GUI Scripts* and Chapter 10, *Reviewing Logs with the LogViewer*.

Editing and Compiling the Script

After you play back a script, you may decide to edit the script to make it more usable. For example, you may want to insert a new verification point or change some text of the script. You may also want to print your script or compile changes.

For more information, see Chapter 7, *Editing, Compiling, and Debugging Scripts*.

Debugging the Script

You may need to debug your script to locate errors. Robot includes a complete, built-in debugging environment to assist you during the development phase of your GUI script.

For more information, see *Debugging GUI Scripts* on page 7-9.

Creating Shell Scripts to Play Back Scripts in Sequence

After you have created each GUI script and verified that it performs as intended, you may want to group the scripts into a shell script. A **shell script** is a script that plays back other scripts in sequence.

For example, you could have:

- ▶ One script that starts your application.
- ▶ A second that searches for and opens a particular file.
- ▶ A third that modifies the file.
- ▶ A fourth that closes the application and returns to the starting point.

Combined into a single shell script, scripts can run in unattended mode and perform comprehensive test coverage. The results from all scripts are stored in the same log, which simplifies results analysis in the LogViewer.

For unattended testing, each shell script should return to a common point in the application-under-test. This common point could be a main menu, a specific window or dialog box, or even the Windows desktop. This assures that script playback remains synchronized with the application-under-test.

Before creating a shell script, you must have already recorded the individual scripts that you intend to include.

Creating a Shell Script

To create a shell script:

1. Click **File** → **New** → **GUI Shell Script**.
2. Type a name (40 characters maximum).
3. Optionally, type a description.
4. Click **OK**.

5. To add scripts, select one or more scripts in the **Available** list and click > or > > . Robot plays back scripts in the same order in which they appear in the **Selected** list.
6. Click **OK**.

The shell script contains a `CallScript` command followed by the name of each script that you included.

Playing Back a Shell Script

You play back a shell script just like any other script. For information, see Chapter 9, *Playing Back GUI Scripts*.

For unattended playback, however, do the following before you play back a shell script:

1. Click **Tools** → **GUI Playback Options**.
2. In the **Playback** tab, clear the **Acknowledge results** check box.

This prevents a pass/fail result message box from appearing for each verification point. You can still view the results in the log after playback.
3. Set the other options in the tabs as appropriate. For information, see *Setting GUI Playback Options* on page 9-4.
4. Click **OK**.

When you play back the shell script, the results from all scripts are stored in the same log, which simplifies results analysis in the LogViewer. For information, see Chapter 10, *Reviewing Logs with the LogViewer*.

Adding Features to GUI Scripts

This chapter describes the features that you can add to GUI scripts. It includes the following topics:

- ▶ Starting an application
- ▶ Inserting a call to another script
- ▶ Inserting verification points
- ▶ Inserting timers
- ▶ Inserting comments
- ▶ Inserting log messages
- ▶ Inserting delay values
- ▶ Using the Insert menu
- ▶ Customizing SQABasic scripts

Starting an Application

While recording or editing a GUI script, you can start applications or other executable programs by using one of the Start buttons on the GUI Insert toolbar, or one of the Start commands on the Insert menu.

NOTE: To successfully test the objects in Oracle Forms, HTML, Java, C++ , and Visual Basic 4.0 applications, you need to enable the applications before you start recording your scripts. For information, see *Enabling IDE Applications for Testing* on page 4-5.

To start an application or executable program while recording or editing a script:



1. Do one of the following:
 - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.

2. Do one of the following:



- To start most applications, click the **Start Application** button. (For exceptions, see the following two bullets.)



- To start an HTML application, click the **Start Browser** button. (For more information, see *Enabling HTML Testing in Robot* on page 19-3.)



- To start a Java application that you plan to play back under Quantify or PureCoverage, click the **Start Java Application** button. (For more information, see *Setting Diagnostic Tools Options* on page 9-11.)

3. Fill in the dialog box and click **OK**.

For details about any item in the dialog box, click the question mark in the upper-right corner and then click the item.

4. Continue recording or editing the script.

During playback, Rational Robot starts the specified application when it reaches that command in the script.

NOTE: Do not use the Windows desktop (such as the Start button) to start an application.

Inserting a Call to Another Script

While recording or editing a GUI script, you can insert a call to a previously recorded GUI script. This lets you avoid repetitive actions in the application-under-test by taking advantage of scripts that already exist.

To insert a call to a previously recorded script while recording or editing:



1. Do one of the following:

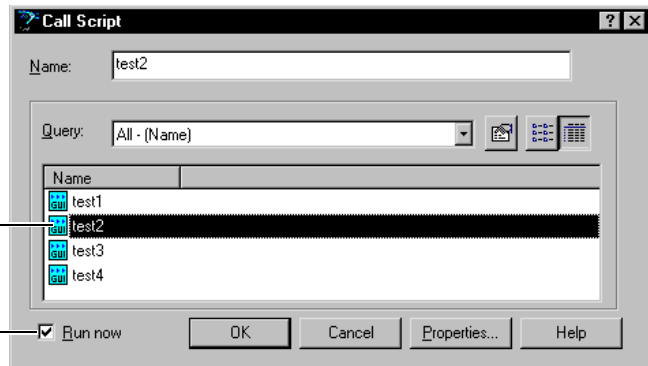
- If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
- If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.



2. Click the **Call Script** button on the GUI Insert toolbar.

Select the script to call.

Select to run the called script when you click OK.



3. Select a GUI script from the list.

The listed scripts have been recorded in Robot or generated in TestFactory. To change the list, select a query from the **Query** list. The query lets you narrow down the list, which is useful in projects with hundreds of scripts. You create queries in TestManager, and you modify queries in TestManager or Robot. (For information about queries, see Chapter 15, *Querying the Rational Repository*.)

4. Do one of the following:
 - Select **Run Now** if the script being recorded depends on the state in which the called script leaves the application-under-test. If this check box is selected, Robot adds the script call to the recording script and immediately plays back the called script when you click **OK**.
 - Clear **Run Now** if the called script starts and ends at the same point in the application-under-test, so that the script being recorded does not depend on the called script. If this check box is cleared, Robot adds the script call to the recording script but does not play back the called script when you click **OK**.
5. Click **OK** to continue recording or editing.

You can also group your scripts into a shell script. For information, see *Creating Shell Scripts to Play Back Scripts in Sequence* on page 4-26.

Inserting Verification Points

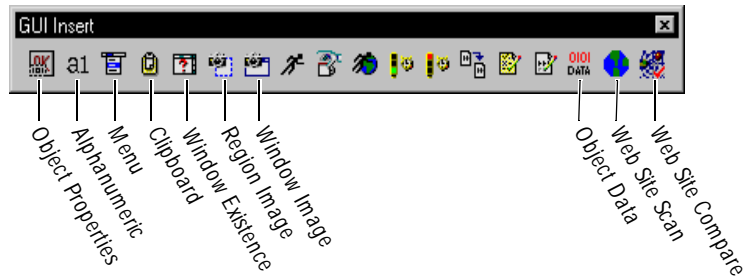
A **verification point** is a point in a script that you create to confirm the state of an object across builds. During recording, the verification point captures object information and stores it as the baseline. During playback, the verification point recaptures the object information and compares it with the baseline.

NOTE: This section gives an overview of how to insert a verification point. For detailed information about verification points, see Chapter 6, *Creating Verification Points in GUI Scripts*.

To insert a verification point while recording or editing a script:



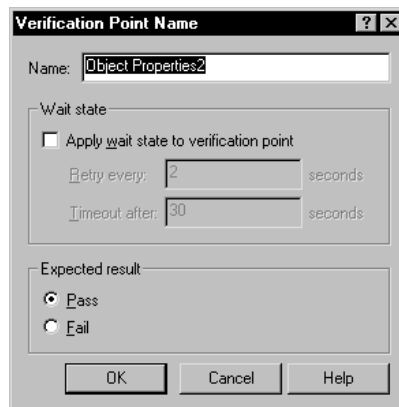
1. Do one of the following:
 - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.
2. Click a verification point button on the GUI Insert toolbar.



NOTE: To insert a File Comparison, File Existence, or Module Existence verification point, open the Robot window (click the **Open Robot Window** button on the GUI Record toolbar). Click **Insert** → **Verification Point** and the appropriate menu command.

3. In the Verification Point Name dialog box, edit the name of the verification point as appropriate.

Robot automatically names the verification point with the verification point type, and adds a number if there is more than one of the same type in the script.



4. Optionally, set the **Wait state** options.

The wait state specifies how often Robot should retry the verification point until it passes or times out, and how long Robot should keep trying the verification point before it times out. (For more information, see *Setting a Wait State for a Verification Point* on page 6-8.)

5. Optionally, set the **Expected result** option.

When you create a verification point, the expected result is usually that the verification point will pass — for example, that a window *does* exist during playback. However, you can also indicate that you expect the verification point to fail — for example, that a window does *not* exist during playback. (For more information, see *Setting the Expected Result for a Verification Point* on page 6-9.)

6. Click **OK**.

Inserting Timers

Robot lets you insert start timer and stop timer commands to record and write to the log the duration of events in a script. A **timer** measures the time it takes to perform an activity. For example, you may want to record the time required to perform a database transaction on a remote server, or how long it takes the same verification point to execute on client machines with different hardware configurations.

You can insert any number of timers with different names into the same script to measure a variety of separate tasks. You can nest timers within other timers (starting and stopping the second timer before stopping the first timer), and you can overlap timers (stopping the second timer after stopping the first timer). However, you should stop a timer before starting that same timer over again. If you start the same timer twice without stopping it, Robot terminates the first occurrence when it starts the second.

If you do not explicitly stop a timer, the timer is stopped automatically at the end of the transaction.

When you play back a script that includes timers, you can view the elapsed time in the log. For more information, see *Playing Back a Script that Includes Timers* on page 5-8.

Uses for Timers

You can use timers to measure general application performance and specific task performance.

Measuring General Application Performance

For general application performance, start a timer, perform a series of actions and create verification points with the application-under-test, and then stop the timer.

When you play back the script, the timer measures the amount of time it took for the application to complete all of the actions. The log shows the timing results.

Measuring Specific Task Performance

For specific task performance, you often use timers with verification points that have wait state values. (For more information, see *Setting a Wait State for a Verification Point* on page 6-8.) You use the wait state value to detect the completion of a task before stopping the timer.

The following is an example of using timers for specific task performance testing:

1. During recording, start a timer.
2. Start an application task or transaction (for example, open an application or start a database query).
3. Insert a verification point with a wait state.
For example, insert a Window Existence verification point that waits up to 30 seconds for a window that indicates the task is complete.
4. Stop the timer.
5. Continue recording other actions or stop the recording.

After you play back the script, the log shows the timing results.

Inserting a Timer

To insert a timer while recording or editing a script:



1. Do one of the following:
 - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.



2. Click the **Start Timer** button on the GUI Insert toolbar.
3. Type a timer name (40 characters maximum) and click **OK**. If you start more than one timer, make sure you give each timer a different name.
4. Perform the timed activity.



5. Immediately after performing the timed activity, click the **Stop Timer** button on the GUI Insert toolbar.
6. Select a timer name from the list of timers you started and click **OK**.

Playing Back a Script that Includes Timers

Do the following before you play back a script that include timers:

1. Click **Tools** → **GUI Playback Options**.

2. In the **Playback** tab, clear **Acknowledge results**.

This prevents a pass/fail result message box from appearing for each verification point. You can still view the results in the log after playback.

3. In the **Playback** tab, set the **Delay between commands** value to 0.

This removes any extra Robot timing delays from the performance measurement. If you need a delay before a single command, click **Insert** → **Delay** and type a delay value.

4. Click **OK**.

When you play back the script and view the log in the LogViewer, the elapsed time is displayed for each Stop Timer event. For more information, see Chapter 9, *Playing Back GUI Scripts* and Chapter 10, *Reviewing Logs with the LogViewer*.

Inserting Comments

During recording or editing, you can insert lines of comment text into a GUI script. Comments are helpful for documenting and editing scripts. Robot ignores comments at compile time.

To insert a comment into a script during recording or editing:



1. Do one of the following:

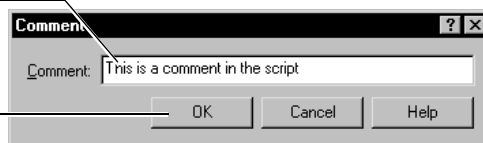
- If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
- If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.



2. Click the **Comment** button on the GUI Insert toolbar and then do the following:

Type a comment
(60 characters maximum).

Click OK to continue
recording or editing.



Robot inserts the comment into the script (in green by default) preceded by a single quotation mark. For example:

```
'This is a comment in the script
```

To change lines of text into comments or to uncomment text:

1. Highlight the text.
2. Click **Edit** → **Comment Line** or **Edit** → **Uncomment Line**.

Inserting Log Messages

During recording or editing, you can insert a log message, description, and result into a GUI script. During playback, Robot inserts this information into the log. You can use log messages to document your script for the playback process.

To insert a log message into a script during recording or editing:



1. Do one of the following:
 - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.



2. Click the **Write to Log** button on the GUI Insert toolbar and then do the following:

Type a message
(60 characters maximum).

Optionally, type a description
(60 characters maximum).

Select a result.

Click **OK** to continue
recording or editing.

After playback, you can view logs and messages using the LogViewer. The message appears in the Log Event column. The result appears in the Result column.

To view the description, select the log event and click **View** → **Log Event Properties**. Click the **Result** tab.

Inserting Delay Values

During playback of a GUI script, Robot adds a delay value between each user action command and between each verification point command. You can set this value in the **Playback** tab of the GUI Playback Options dialog box. (For more information, see *Setting Wait State and Delay Options* on page 9-7.)

At times during playback, you may need to have Robot pause for a specific amount of time before executing a particular command. For example, an additional delay may be necessary if the application accesses a network server, printer, or other remote system. In these cases, if script playback does not wait, it can become out-of-sync with the application by executing script commands before the application is ready for them.

When you insert a delay value into a script, the script waits for the amount of time you specified before playback continues. This delay is useful when you can calculate the amount of time needed for a process to finish before playback resumes.

NOTE: If you are testing an application in which time estimates are not predictable, you can define a wait state for a verification point instead of inserting a delay value. With a wait state, playback waits based on specific conditions rather than on absolute time. For more information, see *Setting a Wait State for a Verification Point* on page 6-8.

To insert a delay value into a script during recording or editing:

1. Do one of the following:

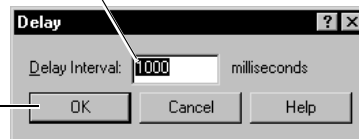


- If recording, click the **Open Robot Window** button on the GUI Record toolbar.
- If editing, position the pointer in the script.

2. Click **Insert** → **Delay** and then do the following:

Type the delay interval in milliseconds. For example:
1 second = 1000
1 minute = 60,000
1 hour = 3,600,000

Click **OK** to continue recording or editing.



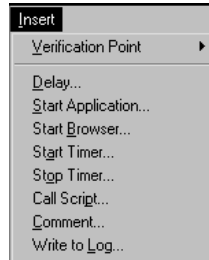
Using the Insert Menu

The preceding sections of this chapter describe how to use the GUI Insert toolbar to add features to scripts. You can also use the Robot **Insert** menu to add these features.

If Robot is minimized while you are recording:



- ▶ Click the **Open Robot Window** button on the GUI Record toolbar. This button restores the Robot window, letting you use the **Insert** menu.



Customizing SQABasic Scripts

In addition to editing a recorded script, you can customize SQABasic scripts in the following ways:

- ▶ By adding your own SQABasic sub procedures and functions either directly to script files or to included **library source files**. The custom procedures you add to library source files can be called from procedures in other files (scripts and other library source files).
- ▶ By using **SQABasic header files** to declare custom procedures, constants, and variables. Items declared in an SQABasic header file are available to multiple script and library source files.
- ▶ By using a script **template**. The template contains information that you want to appear in every new script and .rec library file that you create.

This section describes the basic information you need to know to use Robot to create and edit library source files and SQABasic header files. For syntax and other detailed information about using these files, see the *SQABasic Language Reference*.

Information about using the template appears at the end of this section.

Library Source Files

You can use Robot to create and edit two types of SQABasic library source files:

- ▶ **.sbl** – These have repository-wide scope, but they do not support verification points. They are stored in the SQABas32 folder of the repository.
- ▶ **.rec** – These have project-wide scope, and they do support verification points. They are stored in the Script folder of the project.

The **.rec** files are also used as GUI scripts.

Library source files are useful for storing custom procedures that multiple scripts need to access. If a custom procedure needs to be accessed by just a single script, consider adding the procedure to the script rather than to a library source file.

NOTE: You can also call procedures in **.dll** files from SQABasic scripts and library files. However, you cannot use Robot to create and edit **.dll** files as you can **.sbl** and **.rec** files.

Creating and Editing **.sbl** Library Source Files

To create a new **.sbl** library source file:

1. Click **File** → **New** → **SQABasic File**.
2. Click **Library Source File**, and then click **OK**.

You name the file (or accept the default name) the first time you save it.

A library file cannot have the same name as a script file that calls it. For instance, **MyScript.rec** cannot call a function in **MyScript.sbl**.

To edit an existing **.sbl** library source file:

1. Click **File** → **Open** → **SQABasic File**.
2. In **Files of type**, select **Library Source Files (*.sbl)**.
3. Click the file to edit, and then click **Open**.

Creating and Editing **.rec** Library Source Files

To create a new **.rec** library file:

1. Click **File** → **New** → **Script**.
2. Type the name of the file to create and optionally, a description.
3. Click the file type **GUI** if it is not already selected.
4. Click **OK**.

To edit an existing .rec library file:

1. Click **File** → **Open** → **Script**.
2. Click the name of the file to edit, and then click **OK**.

Adding Procedures to the Global Library Source File

For your convenience, Robot provides a blank library source file called Global.sbl. You can add procedures to this library source file and/or create your own.

To open Global.sbl:

1. Click **File** → **Open** → **SQABasic File**.
2. Set the file type to **Library Source Files (*.sbl)**.
3. Select **global.sbl**, and then click **Open**.

Using Library Source Files

To use an SQABasic library file at runtime, you must:

- ▶ Add custom procedures to the library source file.
- ▶ Compile the file. Both types of SQABasic library source files (extensions .sbl and .rec) compile to a .sbx runtime file.
- ▶ Declare the file in an SQABasic header file or directly in a script or library file that will call the custom procedures.

Here is an example of declaring the sub procedure myProc in the library file Mylibrary.sbx:

```
Declare Sub myProc BasicLib "Mylibrary" (arg as Integer)
```

And here is an example of declaring the function myFunc in the .dll file Mylibrary.dll:

```
Declare Function myFunc Lib "Myblibrary" (ByVal PassVar) as Integer
```

For information about adding custom procedures to SQABasic library files and about declaring library files (including .dll files), see the *SQABasic Language Reference*.

For information about compiling SQABasic library source files, see *Compiling Scripts and SQABasic Library Source Files* on page 7-7.

SQABasic Header Files

Header files let you declare custom procedures, constants, and variables that you want to make available to multiple script and library source files. You can use Robot to create and edit the following kinds of SQABasic header files:

- ▶ **Header files** – These files are stored in the SQABas32 folder of the repository. They can be accessed by all modules within the repository.
- ▶ **Project header files** – These files are stored in the Script folder of the project. They can be accessed by all modules within the project.

Both types of SQABasic header files have the extension .sbh.

Creating and Editing Repository-Wide Header Files

To create a new header file that can be accessed by any module in the repository:

1. Click **File** → **New** → **SQABasic File**.
2. Click **Header File**, and then click **OK**.

You name the file (or accept the default name) the first time you save it.

To edit an existing repository-wide header file:

1. Click **File** → **Open** → **SQABasic File**.
2. In **Files of type**, select **Header Files (*.sbh)**.
3. Click the file to edit, and then click **Open**.

Creating and Editing Project Header Files

To create a new project header file:

- ▶ Click **File** → **New** → **Project Header File**.

You name the file (or accept the default name) the first time you save it.

To edit an existing project header file:

1. Click **File** → **Open** → **Project Header File**.
2. Click the file to edit, and then click **Open**.

Adding Declarations to the Global Header File

For your convenience, Robot provides a blank header file called Global.sbh. Global.sbh is a repository-wide header file stored in SQABas32 in the repository. You can add declarations to this global header file and/or create your own.

To open Global.sbh:

1. Click **File** → **Open** → **SQABasic File**.
2. Set the file type to **Header Files (*.sbh)**.
3. Select **global.sbh**, and then click **Open**.

Using SQABasic Header Files



After you finish adding global declarations to an SQABasic header file, save the file before you compile a script or library file that references the header file. Save the header file by clicking the **Save** toolbar button.

You do not compile SQABasic header files.

Header and Library Source File Examples

The following examples show how a script can reference:

- ▶ Variables and constants declared in a header file.
- ▶ Procedures declared in the header file and defined in a library source file.

To run the example, type the contents of each example file into an *empty* .rec script file, .sbh header file, and .sbl library source file. Before attempting to run the script, save the .sbh file and compile the .sbl file.

NOTE: These examples are also provided in the Robot Help. (See *header files* in the Help Index.) You can copy the examples from the Help into your own files.

The examples and the names you should assign the files are:

- ▶ Example Script – Assign any name to this script.
- ▶ Example Header File – Name the script **tstHeader.sbh**.
- ▶ Example Library Source File – Name the script **tstLibrary.sbl**.

Example Script

Run this example with the example library and header files:

```
'$Include "tstHeader.sbh"  
Option Explicit  
Sub Main  
'Initially Recorded: 01/17/00 18:12:16  
'Script Name: testscript  
  userInput = InputBox("Type a number: ")  
  Call compareNumbers(userInput,NMBR)  
End Sub
```

Example Library Source File (Tstlibrary.sbl)

Run this example with the example script and header files. Be sure to compile the library source file to an .sbx file before you run the script that calls the custom procedure defined in the library file:

```
Sub compareNumbers(inputVal as Integer, constVal as Variant)  
  Dim txt as String  
  If inputVal > constVal then  
    txt="You typed a number greater than "  
  ElseIf inputVal < constVal then  
    txt="You typed a number less than "  
  Else  
    txt="The number you typed equals "  
  End If  
  MsgBox txt + constVal  
End Sub
```

Example Header File (Tstheader.sbh)

Run this example with the example script and library files:

```
Global userInput as Integer  
Global Const NMBR as Variant = 10  
Declare Sub compareNumbers BasicLib "tstLibrary" (arg1 as Integer,  
arg2 as Variant)
```

The Template File

Robot provides a template file, Testproc.tpl, that you can use to automatically add comments or include statements in new GUI scripts. Any text that you add to Testproc.tpl automatically appears in each newly recorded script.

To edit Testproc.tpl:

1. Click **File** → **Open** → **SQABasic File**.
2. Set the file type to **Template Files (*.tpl)**.
3. Select **testproc.tpl** and click **Open**.

4. Type include statements, as in the following example:

```
'Include global declarations in all scripts  
'$Include "global.sbh"
```

The `$Include` metacommand begins with a single quotation mark (`'`). Although this normally indicates a comment, when followed by a dollar sign (`$`) it indicates a special SQABasic command.

5. Click **File** → **Save**.

▶▶▶ C H A P T E R 6

Creating Verification Points in GUI Scripts

This chapter provides conceptual information about verification points and tells you how to perform common operations associated with creating a verification point. It includes the following topics:

- ▶ About verification points
- ▶ Types of verification points
- ▶ Before you create a verification point
- ▶ Tasks associated with creating a verification point
- ▶ Working with the data in data grids
- ▶ Editing a verification point

NOTE: For detailed information about each verification point and how to create it, see the Robot Help.

About Verification Points

A verification point is a point in a script that you create to confirm the state of an object across builds of the application-under-test.

Verification Points and Data Files

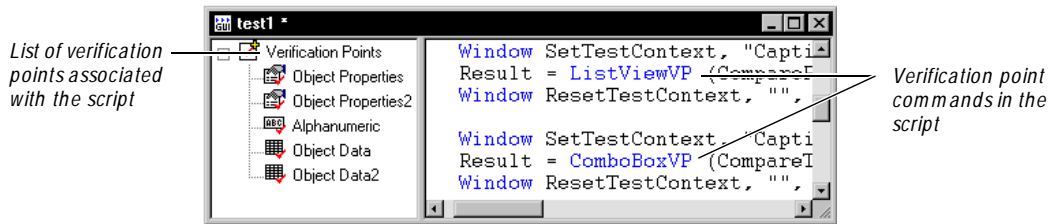
During recording, a verification point captures object information (based on the type of verification point) and stores it in a **baseline data file**. The information in this file becomes the baseline of the expected state of the object during subsequent builds.

When you play back the script against a new build, Rational Robot retrieves the information in the baseline file for each verification point and compares it to the state of the object in the new build. If the captured object does not match the baseline, Robot creates an **actual data file**. The information in this file shows the actual state of the object in the build.

After playback, the results of each verification point appear in the log in the LogViewer. If a verification point fails (the baseline and actual data do not match), you can double-click the verification point in the log to open the appropriate Comparator. The Comparator displays the baseline and actual files so that you can compare them.

Verification Points and Scripts

A verification point is stored in the repository and is always associated with a script. When you create a verification point, its name appears in the Asset (left) pane of the Script window. The verification point script command, which always begins with `Result =`, appears in the Script (right) pane.



NOTE: The following verification points are not stored in the repository and do not appear in the Asset pane: File Comparison, File Existence, Module Existence, Window Existence, and Alphanumeric (if the verification method is Numeric Equivalence or Numeric Range).

Because verification points are assets of a script, if you delete a script, Robot also deletes all of its associated verification points.

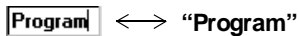
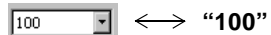
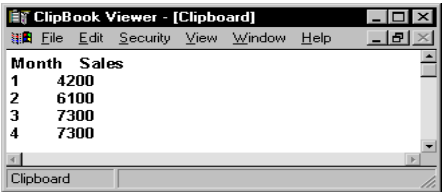

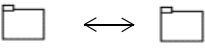
You can easily copy verification points to other scripts if you want to reuse them. For information, see *Copying a Verification Point* on page 6-25.

NOTE: You cannot play back a verification point that you have copied or typed into a .sbl library source file. The verification point must be in a script or a .rec library source file. For information about types of library files, see *Library Source Files* on page 5-12.

Types of Verification Points

The following table summarizes each Robot verification point.

NOTE: For detailed information about each verification point and how to create it, see the Robot Help.

Verification point type	Example
<p>Alphanumeric</p> <p>Captures and tests alphanumeric data in Windows objects that contain text, such as edit boxes, check boxes, group boxes, labels, push buttons, radio buttons, toolbars, and windows (captions). You can use the verification point to verify that text has not changed, to catch spelling errors, and to ensure that numeric values are accurate.</p>	 
<p>Clipboard</p> <p>Captures and compares alphanumeric data that has been copied to the Clipboard. To use this verification point, the application must supply a Copy or Cut capability so that you can place the data on the Clipboard. This verification point is useful for capturing data from spreadsheet and word processing applications as well as terminal emulators.</p>	
<p>File Comparison</p> <p>Compares two specified files during playback. The comparison is based on the contents of the files and their sizes, not on the file names or dates. When you create the verification point, you specify the drive, directory, and file names. During playback, Robot compares the files byte-for-byte.</p>	
<p>File Existence</p> <p>Verifies the existence of a specified file during playback. When you create the verification point, you specify the drive, directory, and file name for the required file. During playback, Robot checks to see if the file exists in the specified location.</p>	

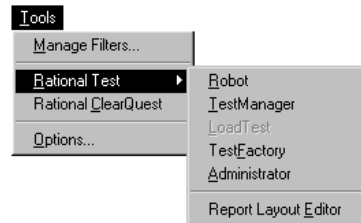
(Continued)

Verification point type

Example

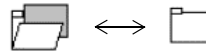
Menu

Captures and compares the menu title, menu items, shortcut keys, and the state of selected menus. Robot records information about the top menu and up to five levels of sub-menus. Robot treats menu items as objects within a menu and tests their content, state, and accelerator keys regardless of the menu item's location. (You can also use the Object Data verification point to test a menu.)



Module Existence

Verifies whether a specified module is loaded into a specified context (process), or is loaded anywhere in memory. Each process has its own context, which includes a set of loaded modules. When you create this verification point, you select the name of the module. You can also select the name of a context (process), in which case the verification point tests whether the module is loaded into that process. If no context is specified, the verification point tests whether the module is loaded anywhere in memory.



Object Data

Captures and compares the data inside standard Windows objects. Also provides specialized support for environment-specific objects such as Visual Basic Data controls, ActiveX controls, HTML and Java objects, PowerBuilder DataWindows, and Oracle Forms base-table blocks.

Robot provides many data tests that are used with the Object Data verification point. A data test is a mechanism for capturing the data of objects. For information about creating your own data tests, see Appendix B, *Working with Data Tests*.






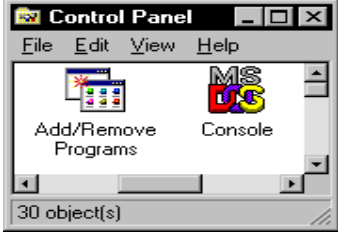
Customer Name	Address	City
ABC Manufacturing, Inc.	23 Broadway	Dayton
Acme Metal Working Corp.	909 East Greenway	Concord
A1 Woodworking Ltd.	542 Great Avenue	Needham
Barrymore Company	8431 Main Street	New York

Object Properties

Captures and compares the properties of standard Windows objects. Also provides specialized support for environment-specific objects such as Visual Basic Data controls, ActiveX controls, HTML and Java objects, PowerBuilder DataWindows, and Oracle Forms base-table blocks.

Properties:	
Name	Value
Alignment	1 - Right Justify
Appearance	1 - 3D
BackColor	RGB(255,255,255)
BackColor.Link	Window
BorderStyle	1 - Fixed Single
ClassName	TextBox
DataChanged	False

(Continued)

Verification point type	Example
<p>Region Image</p> <p>Captures a region of the screen as a bitmap. The captured region is a pixel-by-pixel representation that includes colors, height, and width.</p>	 Violin ↔ 
<p>Web Site Compare</p> <p>Captures a baseline of a Web site and compares it to the Web site at another point in time.</p> <p>Web Site Scan</p> <p>Checks the contents of a Web site with every revision and ensures that changes have not resulted in defects.</p>	
<p>Window Existence</p> <p>Verifies the existence and status of a specified window during playback. The status can be normal, minimized, maximized, or hidden.</p>	 
<p>Window Image</p> <p>Captures a window as a bitmap. The captured window is a pixel-by-pixel representation that includes colors, height, and width.</p>	

NOTE: You can also verify objects through your own custom procedures. You can then use the SQABasic verification point management commands to perform the same kind of verification and LogViewer tasks that Robot performs automatically. For more information, see the *SQABasic Language Reference*.

Before You Create a Verification Point

Before you create a verification point, consider the following:

- ▶ What feature in the application do you want to test?
Example: You want to verify that the **Cut** command places selected data on the Clipboard.
- ▶ To test the feature, what object or objects should you test?
Example: The objects that you should test are the **Cut** command on the Edit menu and the data on the Clipboard.
- ▶ What kind of verification points do you want to create?
Example: You create verification points to test that 1) the **Cut** command exists on the Edit menu and is enabled, and 2) the Clipboard contains the information cut to it.
- ▶ What type of verification points do you create to accomplish the kind of object testing that you want?
Example: You create a script that contains two verification points — an Object Data verification point to test that the **Cut** command exists on the Edit menu and that the state of the **Cut** command is enabled; a Clipboard verification point to test that the selected information is actually placed on the Clipboard.

Tasks Associated with Creating a Verification Point

The following table provides an overview of the major tasks that you perform when you create a verification point and where to look in this chapter for instructions. The specific steps depend on the type of verification point that you create.

Task	See
1. Start to create a verification point.	The next section, <i>Starting to Create a Verification Point</i>
2. Set a wait state.	<i>Setting a Wait State for a Verification Point</i> on page 6-8
3. Set the expected result.	<i>Setting the Expected Result for a Verification Point</i> on page 6-9
4. Select and identify the object to test.	<i>Selecting and Identifying the Object to Test</i> on page 6-10
5. Select a verification method.	<i>Selecting a Verification Method</i> on page 6-14

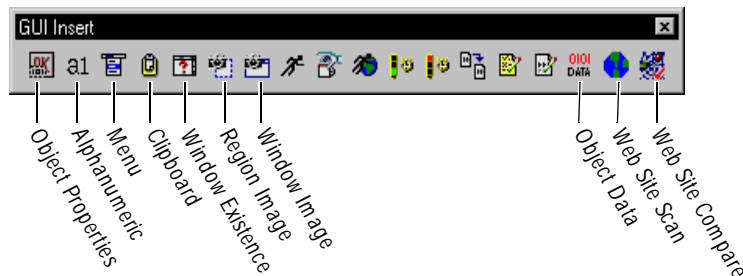
Task	See
6. Select an identification method.	<i>Selecting an Identification Method</i> on page 6-15
7. Select the data or properties to test.	<i>Selecting the Data to Test in a Data Grid</i> on page 6-19
8. Test column titles or menus (optional).	<i>Testing Column Titles or Top Menus in a Data Grid</i> on page 6-20
9. Edit the captured data (optional).	<i>Editing Captured Data in a Data Grid</i> on page 6-21

Starting to Create a Verification Point

The following is the basic procedure for starting to create a verification point:

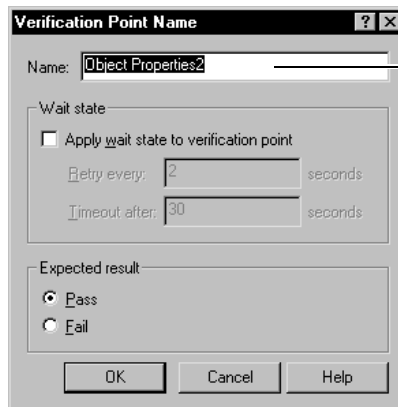


- Do one of the following:
 - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
 - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.
- Click a verification point button on the GUI Insert toolbar.



NOTE: To insert a File Comparison, File Existence, or Module Existence verification point, open the Robot window (click the **Open Robot Window** button on the GUI Record toolbar). Click **Insert** → **Verification Point** and the appropriate menu command.

3. In the Verification Point Name dialog box, edit the name as appropriate. The name can be a maximum of 20 characters.



Robot inserts the verification point type and adds a number if there is more than one of the same type in the script.

4. Optionally, set the **Wait state** options. For information, see the next section, *Setting a Wait State for a Verification Point*.
5. Optionally, set the **Expected result** option. For information, see *Setting the Expected Result for a Verification Point* on page 6-9.
6. Click **OK**.

The steps that you perform next depend on the type of verification point that you are creating. For a list of verification points, see *Types of Verification Points* on page 6-3. For detailed information about each verification point and how to create it, see the Robot Help.

Setting a Wait State for a Verification Point

When you create a verification point, you can add specific wait values to handle time-dependent test activities. Wait values are useful when the application requires an unknown amount of time to complete a task. Using a wait value keeps the verification point from failing if the task is not completed immediately or if the data is not accessible right away.

For example, suppose you create an Alphanumeric verification point that tests for a specific string in a text box. When you play back the script, Robot first looks for the text box. The verification point fails immediately if the box does not exist. If Robot finds the box, it checks for the string in the box. However, the string might not be in the box yet (your application might be running slowly and the box might not be updated yet). To solve this, include wait values so that Robot retries the test (checks for the string) every two seconds. If the content of the box does not match the string within 30 seconds, the verification point returns a failure indication to the script.

For verification points that verify the properties or data of an object, Robot must first find the specified object before it can perform the verification point. After it finds the object, the following happens:

- ▶ If no wait state is specified, the verification point passes or fails immediately.
- ▶ If a wait state is specified, then Robot does the following, as shown in this pseudo-code example:

```
loop until timeout period expires (as specified by Timeout After)
  wait for retry period (as specified by Retry Every)
  perform VP
  if it passes, exit loop, else loop back
end loop
```

To add a wait state when creating a verification point:

1. Start to create the verification point. (See *Starting to Create a Verification Point* on page 6-7.)
2. In the Verification Point Name dialog box, select **Apply wait state to verification point**.
3. Type values for the following options:

Retry every – How often Robot retries the verification point during playback. Robot retries until the verification point passes or until the timeout limit is reached.

Timeout after – The maximum amount of time that Robot waits for the verification point to pass before it times out. If the timeout limit is reached and the verification point has not passed, Robot enters a failure in the log. The script playback either continues or stops based on the setting in the **Error Recovery** tab of the GUI Playback Options dialog box.

Setting the Expected Result for a Verification Point

When you create a verification point, the expected result is usually that the verification point will pass. For example, if you create a Window Existence verification point, you are usually expecting that the window will exist during playback. If the window exists, the verification point passes.

However, suppose you want to test that a window does *not* exist during playback. This is useful when you want a script to wait for a window to disappear before continuing. In this example, you could create a Window Existence verification point with the following values:

- ▶ A timeout wait state value of 30 seconds
- ▶ An expected result of Fail

Because the expected result is a failure, you are telling Robot that you expect the window to *not* exist within 30 seconds. When you play back this verification point, if the window cannot be found at any time during the 30 seconds, the verification point passes. If the window is found during the 30 seconds, the verification point fails.

To set the expected result when creating a verification point:

1. Start to create a verification point. (See *Starting to Create a Verification Point* on page 6-7.)
2. In the Verification Point Name dialog box, click **Pass** or **Fail**.

You might also want to add wait state values to the verification point. (See *Setting a Wait State for a Verification Point* on page 6-8.)

Selecting and Identifying the Object to Test

When you create certain verification points, you need to select the object to test. You do this by pointing to the object with the Object Finder tool, or by selecting the object from a list of all objects on the Windows desktop.

When you point to an object, you can use one of several methods to visually identify the object before you actually select it.

Selecting the Object to Test

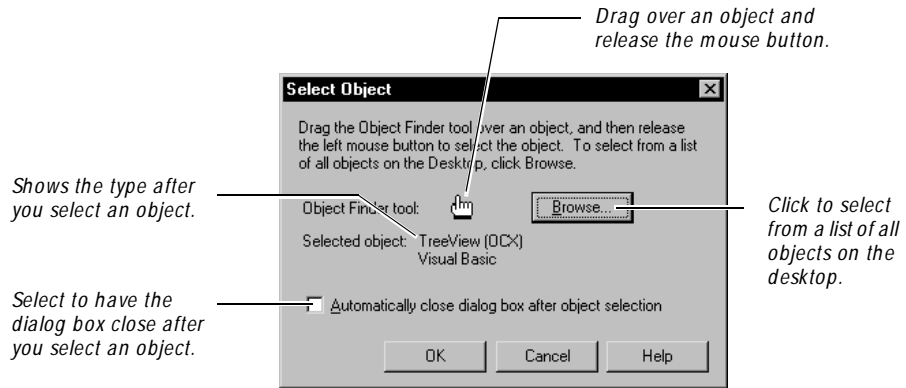
There are two ways to select the object to test:

- ▶ Point to it in the application. This is useful for selecting visible objects.
- ▶ Select it from a list of all objects on the desktop. This is useful for selecting hidden objects.

To select the object to test:

1. Start creating the verification point. (See *Starting to Create a Verification Point* on page 6-7.)

2. In the Verification Point Name dialog box, type a name and click **OK** to open the Select Object dialog box.



3. Do one of the following:
 - Select **Automatically close dialog box after object selection** to have the Select Object dialog box close after you select the object to test.
 - Clear **Automatically close dialog box after object selection** to have the Select Object dialog box reappear after you select the object to test. You will need to click **OK** to close the dialog box.

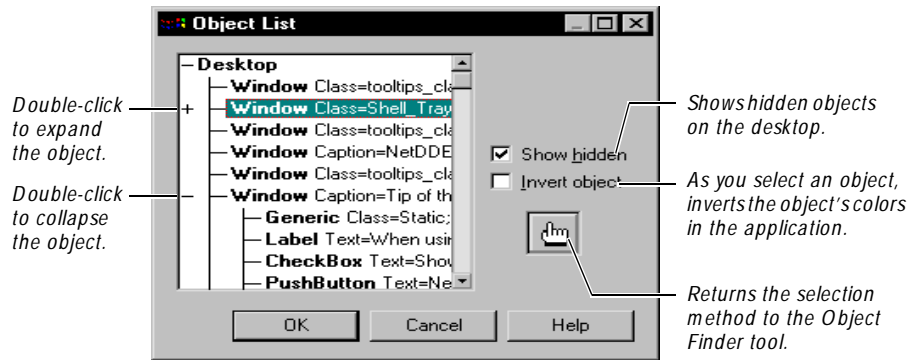
To select a visible object directly from the application, continue with step 4.
To select an object from a list of all objects on the desktop, skip to step 5.

4. To select a visible object directly from the application, drag the Object Finder tool over the object and release the mouse button.

When you drag the Object Finder tool, the Select Object dialog box disappears. When you release the mouse button, the Select Object dialog box reappears if you have cleared the **Automatically close dialog box after object selection** check box.

As you move the Object Finder tool over an object, the object type appears in a yellow TestTip. (For information about how to identify the object to test, see the next section, *Identifying the Object to Test*.)

5. To select a visible or hidden object from a list of all objects on the Windows desktop, click **Browse** to open the Object List dialog box. Select the object from the list and click **OK**.



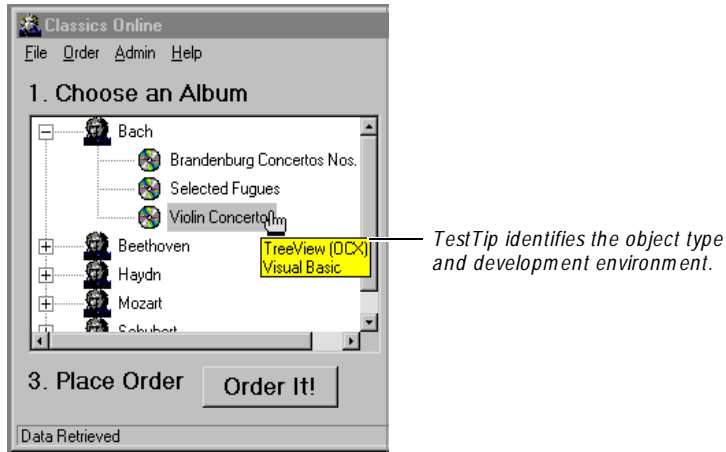
The Object List dialog box includes hidden objects that you cannot point to because they are not visible through the user interface, such as objects with the Visible property set to False and objects with no GUI component. This dialog box also includes objects that are direct children of the desktop, such as PowerBuilder DataStore controls.

When you select an object in the list and click **OK**, it is equivalent to pointing to the object with the Object Finder tool and releasing the mouse button.

NOTE: If you first select an object with the Object Finder tool (in step 4) and then click **Browse**, Robot highlights the selected object in the object list. The object's parent is expanded down to the level of the object. This is useful if there are many objects on the desktop. In this case, you would want to clear the **Automatically close dialog box after object selection** check box in the Select Object dialog box, so that it reappeared after you selected the object.

Identifying the Object to Test

When you point to an object in the application-under-test with the Object Finder tool, Robot displays a TestTip that identifies the object.



You can use one of several methods to visually identify an object. To set the method:

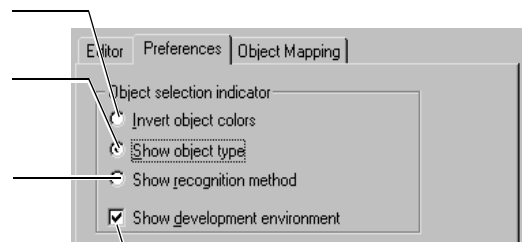


1. If recording, click the **Open Robot Window** button on the GUI Record toolbar to restore the Robot window.
2. Click **Tools** → **General Options**, and then click the **Preferences** tab.

Inverts screen colors as you point to an object.

Displays a TestTip that describes the object type as you point to an object (for example, PushButton).

Displays a TestTip that describes both the object type and the object recognition method as you point to an object.

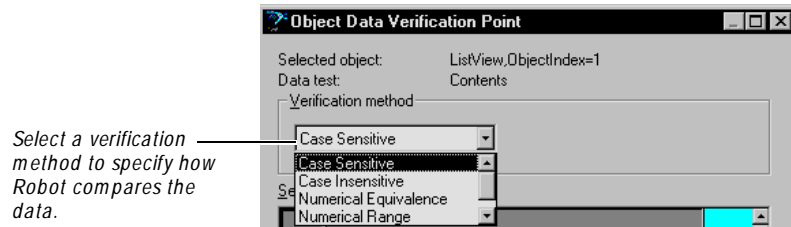


Displays the development environment name (if known) in the TestTip.

NOTE: To change the selection indicator temporarily while pointing to objects, press CTRL or SHIFT.

Selecting a Verification Method

When you create certain verification points, you can select a verification method. The **verification method** specifies how Robot compares the baseline data captured while recording with the data captured during playback.



The verification methods are:

Case-Sensitive – Verifies that the text captured during recording exactly matches the captured text during playback. For example, if you capture *Inches* during recording, the test fails during playback if the captured text is *inches* or if the text contains any other characters.

Case-Insensitive – Verifies that the text captured during recording matches the captured text during playback in content but not necessarily in case. For example, if you capture *Inches* during recording, the test passes during playback if the captured text is *inches*. If the text contains any other characters, the test fails.

Find Sub String Case-Sensitive – Verifies that the text captured during recording exactly matches a subset of the captured text during playback. For example, if you capture *Inches* during recording, the test passes during playback if the captured text is *Inches or Feet*, because *Inches* exists within the text. The test fails if the captured text contains *inches*, because the case is different.

Find Sub String Case-Insensitive – Verifies that the text captured during recording matches a subset of the captured text during playback in content but not necessarily in case. For example, if you capture *Inches* during recording, the test passes during playback if the captured text is *Inches or Feet*, because *Inches* exists within the text. The test also passes if the captured text contains *inches*, because the case does not have to match.

Numeric Equivalence – Verifies that the values of the data captured during recording exactly match the values captured during playback. For example, if you select *24.25* during recording, the test passes during playback only if the captured value is *24.25*.

Numeric Range – Verifies that the values of the data captured during recording fall within a specified range during playback. You specify the **From** and **To** values for the numeric range. During playback, the verification point verifies that the numbers are within that range. For example, you can capture a list containing a range of salaries and then set the high and low values of the range. The test passes during playback only if all of the salaries are within the set range.

User-Defined and **Apply a User-Defined DLL test function** – Passes text to a function within a dynamic-link library (DLL) so that you can run your own custom tests. You specify the path for the directory and name of the custom DLL and the function. The verification point passes or fails based on the result that it receives back from the DLL function. (Use the Apply a User-Defined DLL test function method with the Alphanumeric verification point. Use the User-Defined method with all other verification points.)

Verify that selected field is blank – Verifies that the selected field contains no text or numeric data. If the field is blank, the verification point passes. If the field contains any text or numeric value during playback, the verification point fails. You can use this method on a list if you do not highlight any of the items in the list. (This method is used only with the Alphanumeric verification point.)

Selecting an Identification Method

An **identification method** tells Robot how to identify the values to compare during record and playback.

For example, suppose you want to test that the values of one row in a table remain the same during record and playback. You could specify an identification method so that Robot can identify the values regardless of the location of the row in the table.

When you create certain verification points, you can select an identification method for data that appears in a data grid. A **data grid** shows data in rows and columns in a Robot dialog box. Data grids are used when you create a Clipboard, Menu, or Object Data verification point. You can also select an identification method for properties that have a **list** or **array** value when you create an Object Properties verification point.

If the data is displayed in a two-dimensional grid, you select two identification methods — one for columns and one for rows. If the data is displayed in a one-dimensional grid, you select only one identification method.

There are four identification methods: By Content, By Location, By Title, and By Key/Value. (For a complete list of the identification methods, see *List of Identification Methods* on page 6-18.)

By Content

Use this method to verify that the recorded values exist during playback. This method is location-independent. For example, if you record a value of 100, the verification point passes as long as the value 100 exists during playback.

The following figure shows baseline data captured using Items By Content. During playback, the verification point passes because the recorded value exists even though its location changes.

Corporate Training
Gizmo
Gizmo Deluxe
Mini-Widget
Network Services

Baseline

Corporate Training
Gizmo Deluxe
Gizmo
Mini-Widget
Network Services

Playback (Pass)

By Location

Use this method to verify that the recorded values exist in the same locations during playback. For example, when you test items in a menu, use By Location to verify that the locations of the recorded menu items remain the same during playback. You can also use By Location to verify that the locations of recorded column and row values remain the same during playback.

The following figure shows baseline data captured using Columns By Location and Rows by Location. During playback, the verification point passes because the locations of the recorded values remain the same.

CustomerName	Address	City
ABC Manufacturing, I	23 Broadway	Dayton
Acme Metal Working	909 East Green	Concord
A1 Woodworking Ltd.	542 Great Aven	Needham
Barrymore Companv	8431 Main Stre	New York

Baseline

CustomerName	Address	City
ABC Manufacturing, I	23 Broadway	Dayton
Acme Metal Working	909 East Green	Concord
A1 Woodworking Ltd.	542 Great Aven	Needham
Barrymore Companv	8431 Main Stre	New York

Playback (Pass)

By Title

Use this method to verify that the recorded values remain with their titles (names of menus or columns) during playback, even though the columns may have changed locations.

The following figure shows baseline data captured using By Title. During playback, the verification point passes because the recorded values under the menu title remain the same even though the File and Edit menus have changed positions.

File	Edit
New... Ctrl+N	Undo Ctrl+Z
Open... Ctrl+O	
Save Ctrl+S	Cut Ctrl+X
Save As...	Copy Ctrl+C
	Paste Ctrl+V

Baseline

Edit	File
Undo Ctrl+Z	New... Ctrl+N
	Open... Ctrl+O
Cut Ctrl+X	Save Ctrl+S
Copy Ctrl+C	Save As...
Paste Ctrl+V	

Playback (Pass)

By Key/Value

Use this method to verify that the recorded values in a row remain the same during playback. This method is location-independent. If rows are added or deleted, the verification point passes as long as the recorded values in the row remain the same.

This method also lets you add up to eight keys to the columns in the data grid. The keys function like a primary key in a database table. Each key uniquely identifies a column so that Robot can easily locate and retrieve the records you select. If you add a key to a column, Robot searches for the recorded values in the key column during playback. After Robot locates the values in the key column, it then verifies that the rest of the recorded values in each row have remained the same during playback.

To add or remove a key from a column, position the pointer anywhere in the column and click the right mouse button.

The following figure shows baseline data captured using Rows By Key/Value with a key in the customerid column. During playback, Robot searches only for the recorded value in the key column (for example, 2). After Robot locates the value in the key column, it then compares the recorded values with the baseline values. The verification point passes because the recorded values exist even though the row location changes because a new record was added to the database.

customername	state	customerid
ABC Manufacturing, Inc.	OH	1
Acme Metal Working Corp.	NH	2
A1 Woodworking Ltd.	MA	3
Barrymore Company	NY	4

Baseline

customername	state	customerid
ABC Manufacturing, Inc.	OH	1
Abbot Metals	MA	5
Acme Metal Working Corp.	NH	2
A1 Woodworking Ltd.	MA	3

Playback (Pass)

When you select Rows By Key/Value:

- ▶ Robot uses the Case-Sensitive verification method during playback to verify values in the columns that contain keys. If you select another verification method, it applies to the values in the non-key columns.
- ▶ If you select Numeric Range as the verification method, you must use at least one key. The key tells Robot how to locate a record. Then, Robot compares the data to the specified range of numbers.
- ▶ You can add or change a key in the baseline data file in the Grid Comparator and then recompare the baseline and actual data files. For information, see *Using Keys to Compare Data Files* on page 13-7.

List of Identification Methods

The following table lists the identification methods. The type of verification point that you are creating determines the available identification methods.

Use this method	To test on playback that
Columns By Location	The locations of recorded column values have not changed.
Columns By Title	The recorded values remain with their column titles even if column locations change.
Rows By Location	The locations of recorded row values have not changed.
Rows By Content	The recorded values in a row have not changed.
Rows By Key/Value	The recorded values in a row have not changed; the row may have changed location.
Top Menus By Location	The locations of recorded top menus have not changed.
Top Menus by Title	The recorded values remain with their menu titles even if menu locations change.
Menu Items By Location	The locations of recorded menu items have not changed.
Menu Items by Content	The values of recorded menu items have not changed.
Items by Location	The locations of recorded list items have not changed.
Items by Content	The values of selected list items have not changed.

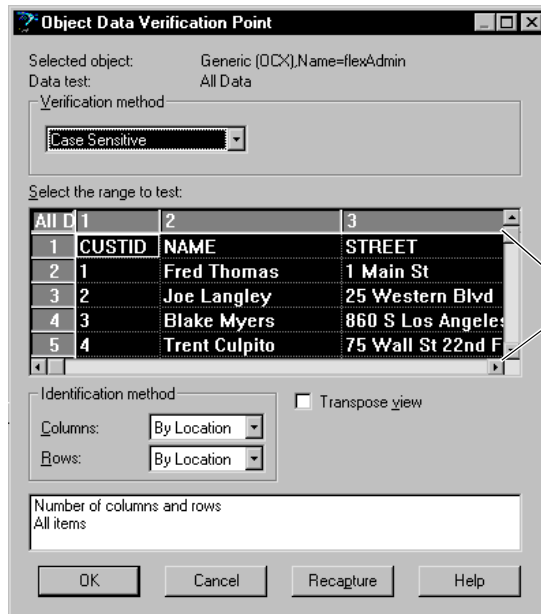
Working with the Data in Data Grids

When you create a Clipboard, Menu, or Object Data verification point and select an object, you are actually testing the object's data. This data appears in a Robot data grid, which shows data in rows and columns. You use the data grid to select and edit the data to test.

Selecting the Data to Test in a Data Grid

After selecting an object but before saving the verification point, you can select the data to test for the following verification points: Clipboard, Menu, and Object Data.

The values originally captured appear in a data grid.



Use the data grid to select a subset of the captured values.

Use any of the following methods to select data in the columns, rows, or cells of the data grid. The selected values become the baseline that Robot uses during playback to test the current build of the application.

To select	Do this
Range	Click and drag the pointer over a range of cells. ...or... Click the first cell, hold down the SHIFT key, and click the last cell in the range. ...or... Hold down the SHIFT key while pressing one of the arrow keys.
Non-contiguous cells	Make sure the captured values are deselected. Then press the CTRL key and click each cell. Clicking without the CTRL key cancels previous selections.
Entire column	Click a column title. Robot compares the data and the number of items in the column.
Entire row	Click a row number. Robot compares the data and the number of items in the row.
All cells	Click the box in the upper-left corner of the grid.

Testing Column Titles or Top Menus in a Data Grid

After you capture data using the Object Data, Menu, or Clipboard verification point, you can select **Move column titles to grid** or **Move top menus to grid** in the Verification Point dialog box.

If you select this check box, the titles move into the data grid and numbers replace the titles above the grid.

	1	2	3
1	customername	address	city
2	ABC Manufacturing, Inc.	23 Broadway	Dayton
3	Acme Metal Working Corp.	909 East Greenway	Concord
4	A1 Woodworking Ltd.	542 Great Avenue	Needham
5	Barrymore Company	8431 Main Street	New York

Column titles are moved into the grid for testing.

Use **Move column titles to grid** or **Move top menus to grid** to:

- ▶ Capture and test a title without its column data.
- ▶ Test the title like any of the other data in the grid.
- ▶ Edit a title by moving it to the grid, editing it, and moving it back to its position as a title.

If a verification point captures only column titles, Robot selects the **Move column titles to grid** check box. Titles are moved to the grid so that data exists in the grid for testing. This check box is not available for list boxes, combo list boxes, and combo boxes.

Editing Captured Data in a Data Grid

After selecting the data to test but before saving the verification point, you can edit the data to test for the following verification points: Clipboard, Menu, and Object Data.

You can edit the data in any cell of a data grid. Editing data is useful if you want to change the baseline for a verification point based on a new specification or anticipated changes to the application-under-test. By editing data before playback, you can often avoid a verification point failure.

Editing Data for a Clipboard or Object Data Verification Point

To edit the data for a Clipboard or Object Data verification point:

1. Double-click a cell in the data grid. The pointer changes to a text cursor.
2. Edit the data in the cell.
3. To accept the changes, press `ENTER`. To cancel the changes, press `ESC`.

NOTE: To edit the titles in a data grid, select the **Move column titles to grid** or **Move top menus to grid** check box.

Editing Data for a Menu Verification Point

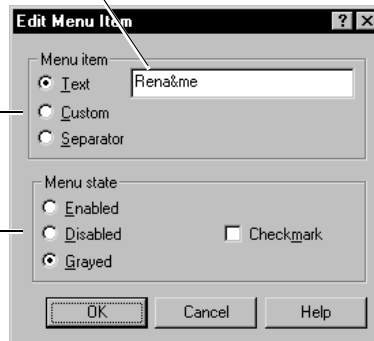
To edit the data for a Menu verification point:

- ▶ Double-click a cell in the data grid to open the Edit Menu Item dialog box.

Change a menu command name by editing its text. Type an ampersand (&) before the letter to be used as the mnemonic accelerator.

Change the item type as needed.

Change the menu state as needed.



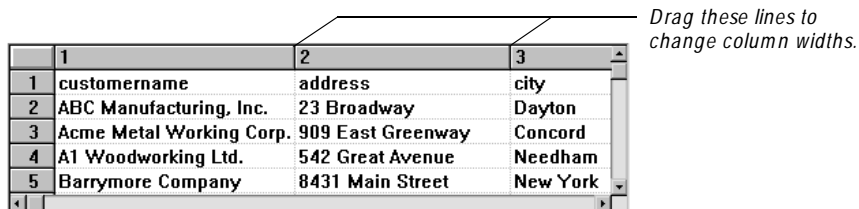
Restrictions on Editing Data

When you edit data in a data grid:

- ▶ You cannot edit column, row, or menu titles unless you use the **Move column titles to grid** or **Move top menus to grid** option.
- ▶ You cannot insert additional columns or rows.
- ▶ You cannot use the **Numeric Range** verification method, because this method does not compare the data to the values in the grid. Instead, it compares the data captured during script playback according to the **From** and **To** values that you specify. Editing the data in the grid has no effect.

Changing a Column Width in a Data Grid

The column widths in the data grid default to fit the longest data string. You can adjust the widths of any of the columns in the grid by dragging the lines between the columns.



Transposing Columns and Rows in a Data Grid

You can transpose the view of the data in the grid by selecting the **Transpose view** check box in the Verification Point dialog box.

	customername	address
1	ABC Manufacturing, Inc.	23 Broadway
2	Acme Metal Working Corp.	909 East Greenway
3	A1 Woodworking Ltd.	542 Great Avenue
4	Barrymore Company	8431 Main Street
5	Olympic Fabricators, Inc.	7483 West Way

When the view is not transposed, data appears in standard rows and columns. Column widths are adjusted according to the contents of each column.

	2	
customername	Acme Metal Working Corp.	A1 Woodwo
address	909 East Greenway	542 Great Av
city	Concord	Needham
state	NH	MA
zipcode	09834	02165

When the view is transposed, columns become rows and rows become columns. Column widths become the same size — the maximum size needed.

Transpose view is a display option only. It does not affect how Robot captures information.

Transpose view is not available for a menu because Robot treats each menu as a separate entity; rows of menu items are not recognized. For example, Robot does not treat the fourth menu item in one menu and the fourth menu item in another menu as though they were in the same row.

Editing a Verification Point

When you record a verification point in a script, the verification point is stored in the repository, along with any associated files. The verification point name appears in the Asset pane of the Robot Script window.

You can view and edit the baseline file of a verification point in one of the Comparators. You can rename, copy, or delete any verification point in a script.

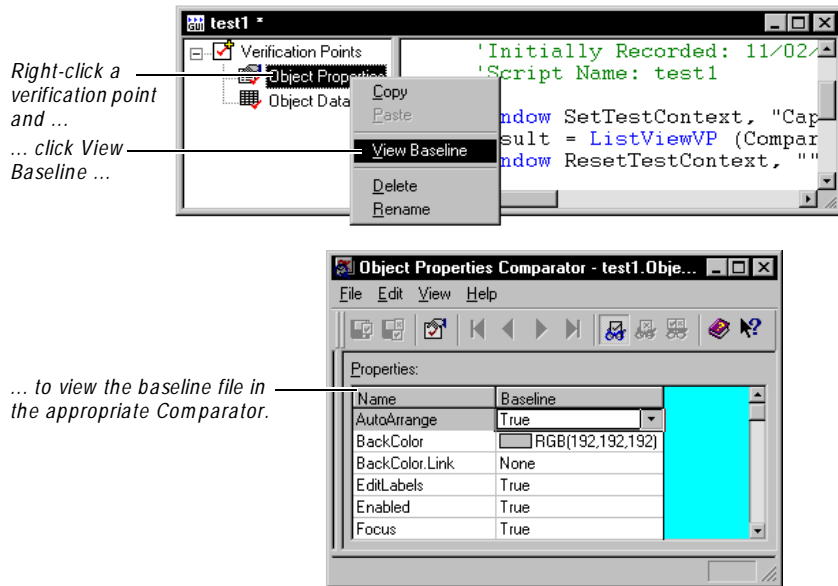
NOTE: The following verification points are not stored in the repository and do not appear in the Asset pane in the Script window: File Comparison, File Existence, Module Existence, Window Existence, and Alphanumeric (if the verification method is Numeric Equivalence or Numeric Range). You can rename, copy, or delete these verification points directly in the script.

Viewing a Baseline File

To view the baseline file of a verification point in a Comparator:

- ▶ In the Asset (left) pane in Robot, right-click the verification point name and click **View Baseline**, or double-click the name.

As the following figure shows, Robot opens the baseline file of an Object Properties verification point in the Object Properties Comparator.



Once the baseline file opens in the appropriate Comparator, you can view and edit the data. Editing data is useful if you want to change the baseline data for a verification point based on a new specification or anticipated changes to the application-under-test. By editing data before playback, you can often avoid a verification point failure. For information about the four Comparators, see Chapters 11, 12, 13, and 14.

NOTE: To compare the baseline and actual files, you must open the Comparator through the LogViewer. For information, see *Evaluating Verification Point Failures in a Comparator* on page 10-9.

Renaming a Verification Point

Renaming a verification point involves two tasks:

- ▶ Renaming the verification point in the Asset pane, which renames the verification point and its associated files in the repository.
- ▶ Renaming all references to that verification point in the script.



When you rename a verification point in the Asset pane, Robot does not automatically rename references to it in the script. If you play back a script that refers to a verification point with a name that is not in the Asset pane (and therefore not in the repository), the verification point and script will fail.

To rename a verification point and its associated files:

1. Right-click the verification point name in the Asset (left) pane and click **Rename**.
2. Type the new name and press **ENTER**.
3. Click the top of the script in the Script (right) pane.
4. Click **Edit** → **Replace**.
5. Type the old name in the **Find what** box. Type the new name in the **Replace with** box.
6. Click **Replace All**.

Copying a Verification Point

You can copy a verification point into the same script or into another script in the same project. Copying a verification point involves two tasks:

- ▶ Copying the verification point name in the Asset pane in one script, and pasting it into the Asset pane in the same script or a different script. This puts a copy of the verification point and its associated files in the repository.
- ▶ Copying the verification point command from the script and pasting it into the same script or a different script.

To copy a verification point:

1. Right-click the verification point in the Asset (left) pane and click **Copy**.
2. In the same script or in a different script (in the same project), right-click **Verification Points** in the Asset pane.
3. Click **Paste** to paste a copy of the verification point and its associated files into the repository.

If a verification point with that name already exists, Robot appends a unique number to the name.

You can also copy and paste by dragging the verification point to **Verification Points** in the Asset pane.

4. Click the top of the Script (right) pane of the original script.
5. Click **Edit** → **Find** and locate the line with the verification point name that you just copied.
6. Select the entire line, which starts with `Result=`.
7. Click **Edit** → **Copy**.
8. Return to the script that you used in step 2. Click the location in the script where you want to paste the line. Click **Edit** → **Paste**.
9. Change the name of the verification point to match the name in the Asset pane.

Deleting a Verification Point

Deleting a verification point involves two tasks:

- ▶ Deleting the verification point name from the Asset pane, which deletes the verification point and its associated files from the repository.
- ▶ Deleting the verification point command from the script.

When you delete a verification point from the Asset pane, Robot does not automatically delete references to that verification point from the script. If you play back a script that refers to a deleted verification point, the verification point and script will fail.

To delete a verification point and its associated files:

1. Right-click the verification point name in the Asset (left) pane and click **Delete**.
2. Click the top of the script in the Script (right) pane.
3. Click **Edit** → **Find**.
4. Type the name of the deleted verification point in the **Find what** box.
5. Click **Find Next**.
6. Delete the entire line, which starts with `Result=`.
7. Repeat steps 5 and 6 until you have deleted all references.

▶▶▶ C H A P T E R 7

Editing, Compiling, and Debugging Scripts

This chapter explains how to edit, print, and compile GUI and virtual user scripts, and how to debug GUI scripts. It includes the following topics:

- ▶ Editing the text of a script
- ▶ Adding a user action to an existing GUI script
- ▶ Adding a feature to an existing GUI script
- ▶ Working with low-level scripts
- ▶ Saving scripts and SQABasic files
- ▶ Printing a script or SQABasic file
- ▶ Compiling scripts and SQABasic library source files
- ▶ Debugging GUI scripts
- ▶ Deleting scripts

Editing the Text of a Script

You can edit the text of any open script. You might want to edit a script to change a command argument or to add conditional logic using the SQABasic language (for GUI scripts) or the VU language (for virtual user scripts). For information about these languages, see the *SQABasic Language Reference* and the *VU Language Reference*.

The Rational Robot **Edit** menu commands use standard Windows mouse and pointer techniques for selecting text. In addition, you can use standard Windows shortcut keys instead of the mouse to select menu commands. Shortcut keys are listed next to the corresponding **Edit** menu commands.

Before starting to edit, you must have a script open. The script can be:

- ▶ A script you have just recorded.
- ▶ A script you have opened.

To edit the text of a script, use the **Edit** menu commands or toolbar buttons.

Some of the **Edit** menu commands are disabled if you are debugging. To stop debugging, click **Debug** → **Stop**.

Adding a User Action to an Existing GUI Script

User actions are actions, such as keystrokes and mouse clicks, that help you navigate around the application. After you record a script, you might decide to add new user actions, such as selecting a menu command, to the script.

To add a new action to an existing script:

1. If necessary, open the script by clicking **File** → **Open** → **Script**.
2. If you are currently debugging, click **Debug** → **Stop**.
3. In the Script window, click where you want to insert the new actions. Make sure that the application-under-test is in the appropriate state to begin recording at the text cursor position.
4. Click the **Insert Recording** button on the Standard toolbar.



The Robot window minimizes by default, or behaves as specified in the GUI Record Options dialog box.

5. Continue working with the application-under-test as you normally do when recording a script.

Adding a Feature to an Existing GUI Script

Features you might want to add to an existing script include verification points, timers, and comments. You can easily add these features while you are recording a script or after you finish recording.

To add a feature to an existing GUI script:

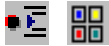
1. If necessary, open the script by clicking **File** → **Open** → **Script**.
2. If you are currently debugging, click **Debug** → **Stop**.

3. In the Script window, click where you want to insert the feature. Make sure that the application-under-test is in the appropriate state to insert the feature at the text cursor position.

4. Do one of the following:



- To add the feature without going into recording mode, click the **Display GUI Insert Toolbar** button on the Standard toolbar. The Robot Script window remains open.



- To start recording and add the feature, click the **Insert Recording** button on the Standard toolbar. The Robot window minimizes by default, or behaves as specified in the GUI Record Options dialog box. Click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.

5. Click the appropriate button on the GUI Insert toolbar.

NOTE: The following features are *not* on the GUI Insert toolbar: File Comparison, File Existence, Module Existence, and Delay. To add these features to your script, open the Robot window if necessary (by clicking the **Open Robot Window** button on the GUI Record toolbar). Click the **Insert** menu, and then click the appropriate command.

6. Continue adding the feature as usual.

For more information about the features you can add, see Chapter 5, *Adding Features to GUI Scripts*.

Working with Low-Level Scripts

As indicated in *Switching to Low-Level Recording* on page 4-22, Robot has two recording modes:

- ▶ Object-Oriented Recording mode
- ▶ Low-level recording mode

If you turn on low-level recording, Robot tracks detailed mouse movements and keyboard actions by screen coordinates and exact timing. Robot records these low-level actions in a binary script file. You can view an ASCII version of this binary file. You can also rename, copy, or delete the file.

A low-level script is stored in the repository and is always associated with a Robot script. When you create a low-level script, its name appears in the Asset pane of the Script window. If you delete a Robot script, its associated low-level scripts are also deleted.

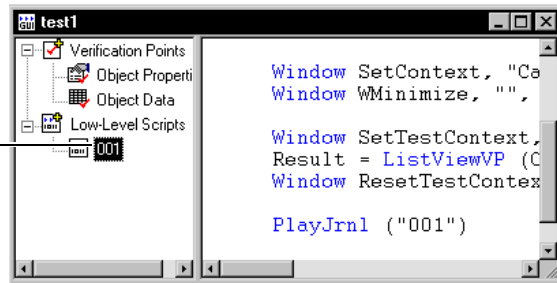
Viewing Low-Level Scripts

You cannot edit the low-level binary file, but you can use Notepad to view an ASCII version of the binary file.

To view the low-level script file:

1. In the Asset (left) pane of the Script window, expand **Low Level Scripts** if necessary by clicking the plus sign (+).
2. Double-click the number of the low-level script that you want to view in Notepad.

Double-click the low-level script in the Asset pane to view an ASCII version of the binary file.



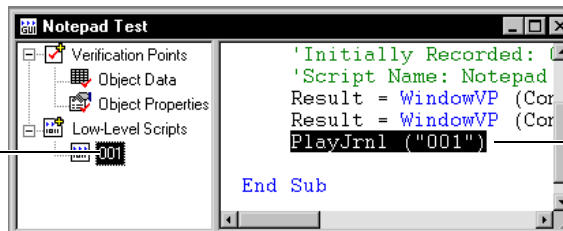
The low-level ASCII file lists the actions that occurred during low-level recording. For information about the contents of this file, see *low-level recording* in the Robot Help Index.

Renaming a Low-Level Script

When you record a low-level script, it is stored in the repository. You can rename the low-level script if needed. Renaming a low-level script involves two tasks:

- ▶ Renaming the low-level script in the Asset pane, which renames it in the repository.
- ▶ Renaming all references to that low-level script in the script.

If you rename the low-level script in the Asset pane ...



... you also need to rename references to it in the script.

When you rename a low-level script in the Asset pane, Robot does not automatically rename references to it in the script. If you play back a script that refers to a low-level script with a name that is not in the Asset pane (and therefore is not in the repository), the script will fail.

To rename a low-level script:

1. Right-click the low-level script name in the Asset (left) pane and click **Rename**.
2. Type the new name and press **ENTER**.
3. Click the top of the script in the Script (right) pane.
4. Click **Edit** → **Replace**.
5. Type the old name in the **Find what** box. Type the new name in the **Replace with** box.
6. Click **Replace All**.

Copying a Low-Level Script

You can copy a low-level script to the same script or to a different script in the same project. Copying a low-level script involves two tasks:

- ▶ Copying the low-level script name in the Asset pane in one script, and pasting it into the Asset pane in the same script or a different script. This puts a copy of the low-level script in the repository.
- ▶ Copying the low-level script command from the script and pasting it into the same script or a different script.

To copy a low-level script:

1. Right-click the low-level script in the Asset (left) pane and click **Copy**.
2. In the same script or in a different script (in the same project), right-click **Low-Level Scripts** in the Asset pane.
3. Click **Paste** to paste a copy of the low-level script into the repository.

If a low-level script with that name already exists, Robot appends a unique number to the name.

You can also copy and paste by dragging the low-level script to **Low Level Scripts** in the Asset pane.

4. Click the top of the Script (right) pane of the original script.
5. Click **Edit** → **Find** and locate the line with the low-level script name that you just copied.
6. Select the entire line, which starts with `PlayJrn1`. Click **Edit** → **Copy**.
7. Return to the script that you used in step 2. Click the location in the script where you want to paste the line, and then click **Edit** → **Paste**.
8. Change the name of the low-level script to match the name in the Asset pane.

Deleting a Low-Level Script

If you no longer need a low-level script, you can delete it. Deleting a low-level script involves two tasks:

- ▶ Deleting the low-level script name in the Asset pane (left pane), which deletes the low-level script from the repository.
- ▶ Deleting the low-level script command from the script.

When you delete a low-level script in the Asset pane, Robot does not automatically delete references to it from the script. If you play back a script that refers to a deleted low-level script, the script will fail.

To delete a low-level script:

1. Right-click the low-level script name in the Asset (left) pane and click **Delete**.
2. Click the top of the script in the Script (right) pane.
3. Click **Edit** → **Find**.
4. Type the name of the deleted low-level script in the **Find what** box.
5. Click **Find Next**.
6. Delete the entire line, which starts with `PlayJrn1`.
7. Repeat steps 5 and 6 until you have deleted all references.

Saving Scripts and SQABasic Files

Robot saves a script after you define it or record it. You can also save any open script or SQABasic file manually.

To save open scripts or SQABasic files, do one of the following:

To save	Do this
The active script or file	Click File → Save .
The active script or file with a new name	Click File → Save As . Type the new name and click OK .
All open scripts and files	Click File → Save All .

You can save only within the current repository and project.

Printing a Script or SQABasic File

To print an open script or SQABasic file:

1. If necessary, click **File** → **Page Setup** to set up the format of printed output.
To add information to the page header or footer, you need to use print codes. For a description of these codes, click the **Help** button in the Page Setup dialog box.
2. Click **File** → **Print**.
3. Set the print options as needed and click **OK**.

Robot uses standard Windows Print Setup dialog boxes. For more information, see your Windows documentation.

Compiling Scripts and SQABasic Library Source Files

When you play back a GUI script or virtual user script, or when you debug a GUI script, Robot compiles the script if it has been modified since it last ran.

You can also compile scripts and SQABasic library source files manually.

Compiling One or All Scripts and Library Source Files

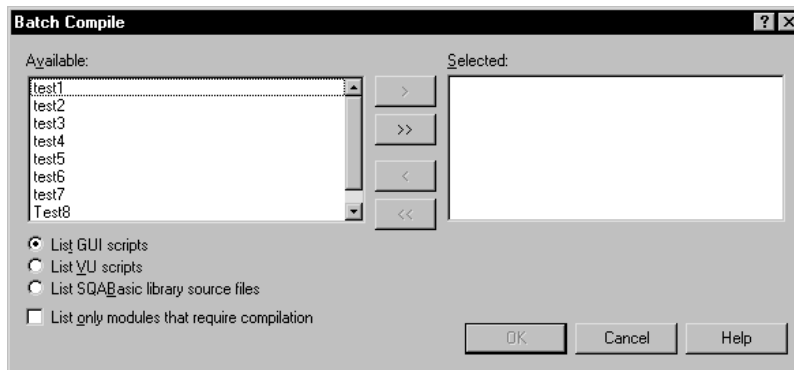
You can compile the active script or file, or you can compile all scripts and files in the current project.

To	Do this
Compile the active script or library source file	Click File → Compile .
Compile all scripts and library source files in the current project	Click File → Compile All . Use this if, for example, you have made changes to global definitions that may affect all of your SQABasic files.

Batch Compiling Scripts and Library Source Files

To batch compile scripts and library source files:

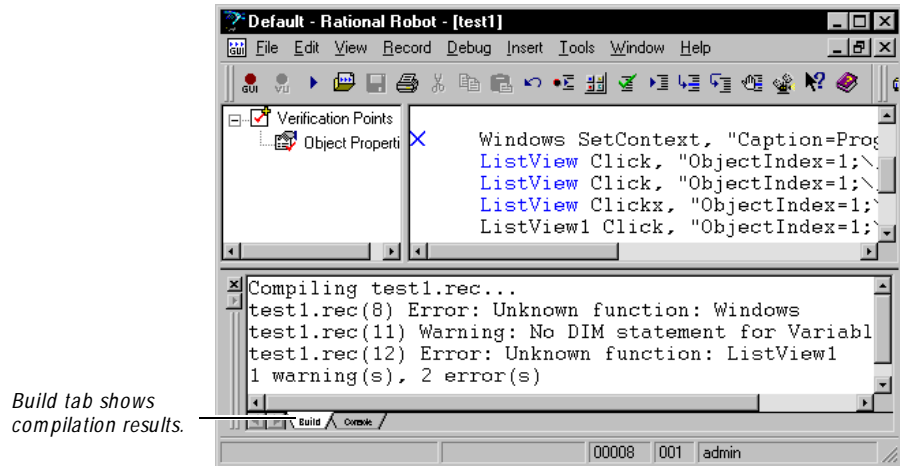
1. Click **File** → **Batch Compile**.



2. Select an option to filter the type of scripts or files you want to appear in the **Available** list: GUI scripts, VU scripts, or SQABasic library source files.
3. Optionally, select **List only modules that require compilation** to display only those files that have not yet been compiled or that have changed since they were last compiled.
4. Select one or more files in the **Available** list and click **>** or **>>**. Robot compiles the files in the same order in which they appear in the **Selected** list.
5. Click **OK** to compile the selected files.

Locating Compilation Errors

During compilation, the **Build** tab of the Output window displays compilation results and error messages with line numbers for all scripts and library source files.



To locate compilation errors in the Script window, do one of the following:

- ▶ Double-click the error or warning in the **Build** tab. Robot moves the cursor to the beginning of the line and inserts an X in the left margin or highlights the line.
- ▶ Click **Edit** → **Next Error** or **Edit** → **Previous Error**. Robot moves the cursor to the beginning of the line and inserts an X in the left margin or highlights the line.
- ▶ Click **Edit** → **Go to Line**, type the line number, and click **OK**. Robot moves the cursor to the beginning of the line.

Debugging GUI Scripts

Robot includes a complete, built-in debugging environment to assist you during the development phase of your GUI scripts.






NOTE: Robot does not have a debugging environment for virtual user scripts.




Before you start to debug, you must have an open GUI script. The script can be:

- ▶ A script that you have just recorded.
- ▶ A script that you have opened by clicking **File** → **Open** → **Script**.

To debug a GUI script, use the **Debug** menu commands or toolbar buttons on the Standard toolbar or Playback toolbar. When you start to debug, Robot automatically compiles the script if it has changed since it last ran, and displays the Playback toolbar.

The following table describes the commands on the **Debug** menu.

Debug command	Description
<p>Go</p> 	<p>Plays back the currently open script.</p> <p>Executes until either the next breakpoint or the end of the script, whichever comes first.</p>
<p>Go Until Cursor</p> 	<p>Plays back the currently open script, stopping at the text cursor position.</p> <p>Executes until either the next breakpoint or the end of the script, whichever comes first.</p>
<p>Animate</p> 	<p>Plays back the currently open script, displaying a yellow arrow in the left margin of each line (or highlighting the line) as it executes.</p> <p>Executes until either the next breakpoint or the end of the script, whichever comes first.</p>
<p>Pause</p> 	<p>Pauses playback. To resume playback, click Debug → Pause.</p>
<p>Stop</p> 	<p>Stops playback.</p>
<p>Set or Clear Breakpoint</p>	<p>Sets or clears a breakpoint at the cursor position.</p> <p>If you set a breakpoint, Robot inserts a solid red circle in the left margin or highlights the line.</p> <p>If you clear a breakpoint, Robot removes the circle or highlighting.</p>

Debug command	Description
Clear All Breakpoints	Clears all breakpoints in the script.
Step Into 	Begins single-step execution. (The subprogram you initially step into is <code>Ma.i.n.</code>) Executes one command at a time.
Step Over 	Enabled after you step into a script. Executes a single command line within a script. If the command calls another script, Robot executes the called script as if it were a single instruction and moves to the command immediately following the script call. If the command is the last line in a called script, Robot returns to the calling script and stops at the command immediately following the script call.
Step Out 	Enabled after you step into a script. Steps out of the called script and returns to the calling script. Execution stops at the command immediately following the script call. Step Out is equivalent to Go Until Cursor with the text cursor placed in the calling script in the command line immediately following the script call.

You can also use the **Next Error** and **Previous Error** commands on the **Edit** menu. These commands move the text cursor to the line containing the next or previous compiler error, and add an X in the left margin or highlight the line.

Setting and Clearing Breakpoints

Robot lets you set any number of breakpoints in a script. A **breakpoint** is a location in a script where you want execution to stop.

When execution stops at a breakpoint, you can examine the value of a variable or check the state of an object before it is modified by a subsequent command. You can then resume execution until the next breakpoint or the end of the script.

To set and clear breakpoints:

1. If necessary, open a script by clicking **File** → **Open** → **Script**.
2. Place the pointer on the line where you want to set a new breakpoint or clear an existing breakpoint.

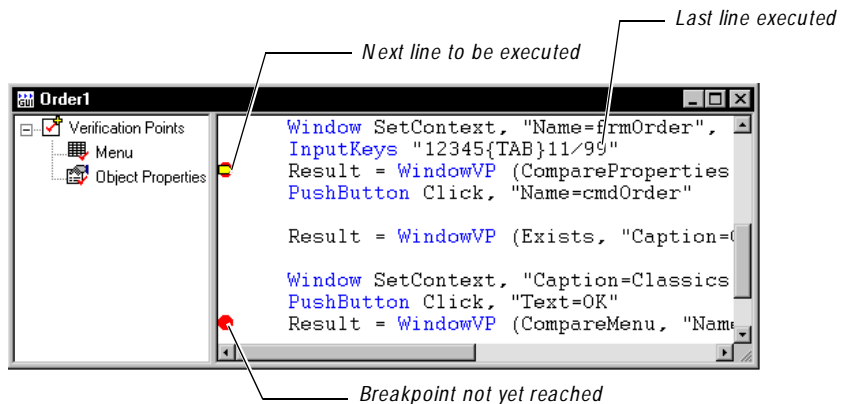
You can only place a breakpoint on a line where an SQABasic command is executed. Breakpoints on comments, labels, and blank lines are not supported. Also, there are a very few commands that do not support breakpoints (for example, `Dim` and `Sub`).

3. Click once to insert a blinking text cursor. (You can also highlight the entire line or any part of the line.)
4. Click **Debug** → **Set or Clear Breakpoint**.

If you set a breakpoint, Robot inserts a solid red circle in the left margin or highlights the line. If you clear a breakpoint, Robot removes the circle or highlighting.

5. If you set a breakpoint, click **Debug** → **Go**.

Robot executes as far as the breakpoint, and then displays a yellow arrow in the left margin of that line or highlights the line.



If you attempt to assign a breakpoint to a line of code that does not support breakpoints, Robot does the following:

- ▶ If you attempt an unsupported breakpoint assignment before you execute the script, the assignment appears to be successful, and no warning message appears. However, when script execution begins, Robot automatically removes invalid breakpoint assignments.

- ▶ If you attempt an unsupported breakpoint assignment during the execution of a script (for example, while execution is stopped at a breakpoint), the warning message *This is not an executable line of code* appears in the status bar.

Executing to a Selected Line

To stop execution at a selected line in a script without setting a breakpoint:

1. If necessary, open a script by clicking **File** → **Open** → **Script**.
2. Place the cursor on the line where you want execution to stop.
3. Click once to insert a blinking text cursor. (You can also highlight the entire line, or any part of the line.)
4. Click **Debug** → **Go Until Cursor**.

Robot executes as far as the line with the text cursor, and displays a yellow arrow in the left margin of that line or highlights the line.

Executing in Animation Mode

To play back a script in animation mode, so you can see each line as it executes:

1. If necessary, open a script by clicking **File** → **Open** → **Script**.
2. Move and resize the Robot window so that it does not cover the application-under-test but so that you can still see the Script window.
3. Click **Debug** → **Animate**.

As Robot plays back the script, it displays a yellow arrow in the left margin of the currently executing line or highlights the line.

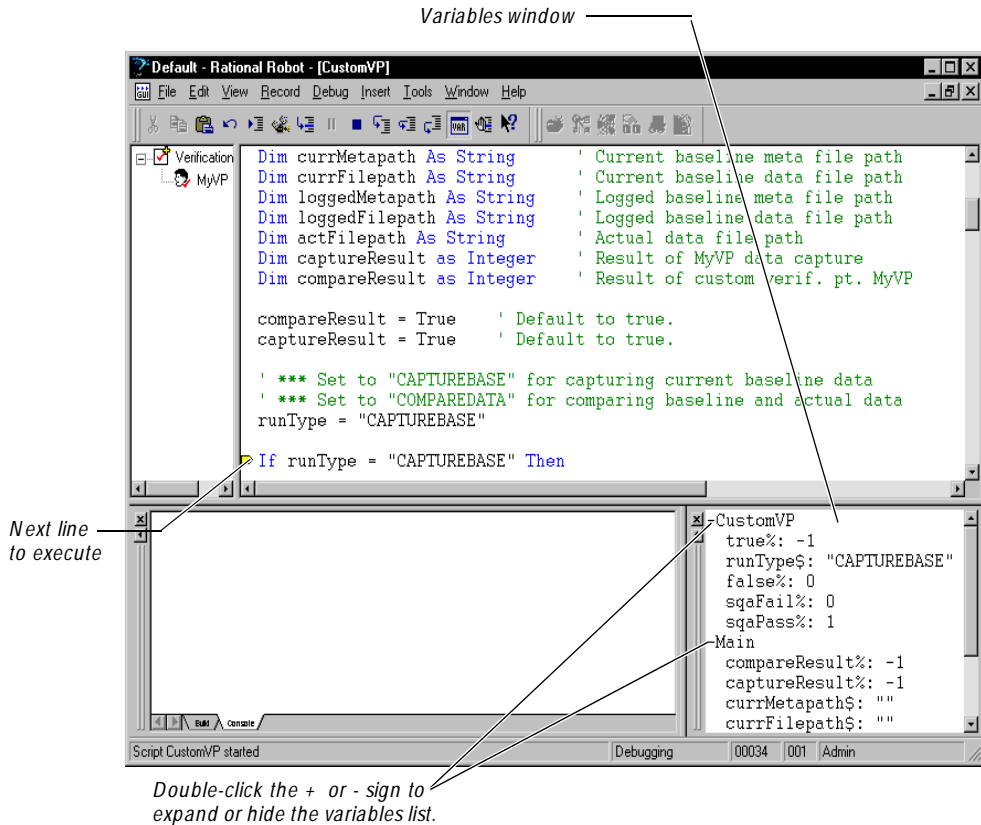
When playing back a script in animation mode, you may want to increase the delay between commands. This slows down the execution of user action commands and verification point commands so you can view line-by-line animation more clearly. (For information, see *Setting Delay Options for Commands and Keystrokes* on page 9-8.)

Examining Variable Values

You can examine variable and constant values in the Variables window as you play back scripts during debugging.

The Variables window appears in the lower-right corner of the Robot main window. If the Variables window is not open, click **View** → **Variables** to open it.

Editing, Compiling, and Debugging Scripts



The Variables window contains lists of the variables that are assigned values during playback and the constants that are referred to during playback.

Variable and constant values are updated each time execution pauses during playback — for example, at a breakpoint, or as you step through the script line by line. Variable and constant values are also updated during animation mode when each statement is executed.

The data type of each variable and constant listed in the Variables window is indicated by a type-declaration character at the end of the variable or constant name.

Variables and constants are grouped according to scope. For example, in the previous figure:

- ▶ The variables listed under **Main** are local variables that are visible only to the **Main** sub procedure.
- ▶ The variables listed under **Custom VP** are module-level variables that are visible to all the sub procedures in the script **Custom VP**.

Variables and constants that are visible to all modules are listed under the heading *Globals*.

Deleting Scripts

To delete scripts from the repository:

1. Click **File** → **Delete**.
2. Select one or more scripts from the list.

To change the list of scripts, select a query from the **Query** list.

3. Click **Delete**. Click **OK** to confirm the deletion.
4. Click **Close**.

Deleting a GUI script from the repository also deletes its corresponding script file (.rec), executable file (.sbx), verification points, and low-level scripts.

Deleting a virtual user script deletes the .s file and its properties, but not the associated watch file (.wch).

For information about deleting scripts from TestManager, see *Deleting Scripts* on page 2-18.

Working with Datapools

This chapter describes how to create and manage datapools. It includes the following topics:

- ▶ What is a datapool
- ▶ Planning and creating a datapool
- ▶ Data types
- ▶ Using datapools with GUI scripts
- ▶ Managing datapools
- ▶ Managing data types
- ▶ Generating and retrieving unique datapool rows
- ▶ Creating a datapool outside Rational Test
- ▶ Creating a column of values outside Rational Test

Before you begin to work with datapools, you should familiarize yourself with the concepts and procedures in this chapter

NOTE: This chapter describes datapool access from GUI scripts played back in Rational Robot. If you have Rational Suite PerformanceStudio installed, see the datapools chapter in the *Using Rational LoadTest* manual for information about accessing datapools from GUI and virtual user scripts played back in a LoadTest schedule.

What Is a Datapool?

A **datapool** is a test dataset. It supplies data values to the variables in a script during script playback.

Datapools automatically pump a different set of test data to a script each time a script sends data to the server during the playback of a test. Consider using a datapool whenever multiple records are being sent to the server in a single playback, and you want to send a different record each time — for example:

- ▶ If a script transaction sends a record to the server, and the script repeats the transaction multiple times through a loop (such as a `For...Next` loop).
- ▶ If a script transaction sends a record to the server, and the script is executed multiple times through a `CallScript` command in a loop.
- ▶ If multiple scripts are executed consecutively through a shell script, and each script sends the server one or more records of the same type.

If you do not use a datapool, the same literal values (the values that were captured when you recorded the script) are sent to the server each time a record is sent to the server during script playback.

Datapool Tools

You use Robot to manually add datapool commands to GUI scripts. You use TestManager to create and manage datapools — for example:

- ▶ Create a datapool and automatically generate datapool values.
- ▶ Edit datapool column definitions.
- ▶ Edit datapool values.
- ▶ Create and edit datapool data types.
- ▶ Perform datapool management activities such as copying and renaming datapools.
- ▶ Import and export datapools.
- ▶ Import data types.

This chapter describes how to perform all of these tasks.

Managing Datapool Files

A datapool consists of two files:

- ▶ Datapool **values** are stored in a text file with a .csv extension.
- ▶ Datapool **column names** are stored in a specification(.spc) file. The software for Rational Robot or TestManager is always responsible for creating and maintaining this file. You should never edit this file directly.

.csv and .spc files are stored in the Datapool directory of your Robot project.

Unless you import a datapool, the Robot or TestManager software automatically creates and manages the .csv and .spc files based on instructions that you provide through the user interface.

If you import a datapool, you are responsible for creating the .csv file and populating it with data. However, the Rational Test software is still responsible for creating and managing the .spc file for the imported datapool.

For information about importing datapools, see *Importing a Datapool* on page 8-28 and *Creating a Datapool Outside Rational Test* on page 8-38.

NOTE: LoadTest automatically copies a .csv file to each Agent computer that uses it. If an Agent's .csv file becomes out-of-date, LoadTest automatically updates it.

Datapool Cursor

The datapool **cursor**, or row-pointer, is automatically reset after you run a test in Robot. In other words, if the last datapool row accessed at the end of a test run is row 100, access resumes with row 1 the first time the datapool is accessed in a new test. Access does not resume with row 101.

Row Access Order

Row access order is the sequence in which datapool rows are accessed at test runtime.

With GUI scripts, you can control the row access order of the datapool cursor through the *sequence* argument of the `SQABasic SQADatapoolOpen` command.

Datapool Limits

A datapool can have up to 150 columns if the Rational Test software automatically generates the data for the datapool, or 32,768 columns if you import the datapool from a database or other source. Also, a datapool can have up to 2,147,483,647 rows.

What Kinds of Problems Does a Datapool Solve?

If a single record is sent to the server during script playback, the script probably does not need to access a datapool.

But sometimes, script playback involves sending multiple records to the server — for example, if the same transaction is repeated multiple times during playback.

If the values used in each transaction are the same literal values — the values you provided during recording — you might encounter problems at test runtime.

Here are some examples of problems that datapools can solve:

- ▶ *Problem:* During recording, you create a personnel file for a new employee, using the employee's unique social security number. Each time the transaction is repeated, there is an attempt to create the same personnel file and supply the same social security number. The application rejects the duplicate requests.

Solution: Use a datapool to send different employee data, including unique social security numbers, to the server each time the transaction is repeated.

- ▶ *Problem:* You delete a record during recording. During playback, each transaction attempts to delete the same record, and “Record Not Found” errors result.

Solution: Use a datapool to reference a different record in the deletion request each time the transaction is repeated.

Planning and Creating a Datapool

The following is a summary of the stages involved in preparing a datapool for use in testing. Stages two and three can be performed in any order:

1. Plan the datapool.

Determine the datapool columns you need. In other words, what kinds of values (names, addresses, dates, and so on) do you need to retrieve from the datapool and send to the server?

Typically, you need a datapool column for each script variable that you plan to assign datapool values to during recording.

For example, suppose your client application has a field called **Order Number**. During recording, you type a value for that field. Later, you edit the script and substitute the literal value that you recorded with a variable. During playback, the variable can be assigned unique order numbers from a datapool column.

This stage requires some knowledge of the client application and the kinds of data that it processes.

2. Add datapool code to the script.

To access a datapool at runtime, a script must contain datapool commands, such as commands for opening the datapool and fetching a row of data.

With GUI scripts, you manually insert the datapool commands into the script during editing — for example, substituting a variable for a literal value you recorded, and then matching up the variable with a datapool column through the `SQADatapoolValue` command. For information about using this and other datapool commands, see *Using Datapools with GUI Scripts* on page 8-12.

3. Create and populate the datapool.

You create and populate the datapool in the TestManager Datapool Specification dialog box. The following is a summary of these basic tasks:

- Define the datapool columns that you determined you needed during the planning stage. For example, for an Order Number column, you can specify the maximum number of characters that an order number can have, and whether the Order Number column must contain unique values. For information about defining datapool columns in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 8-20 and *Example of Datapool Column Definition* on page 8-23.

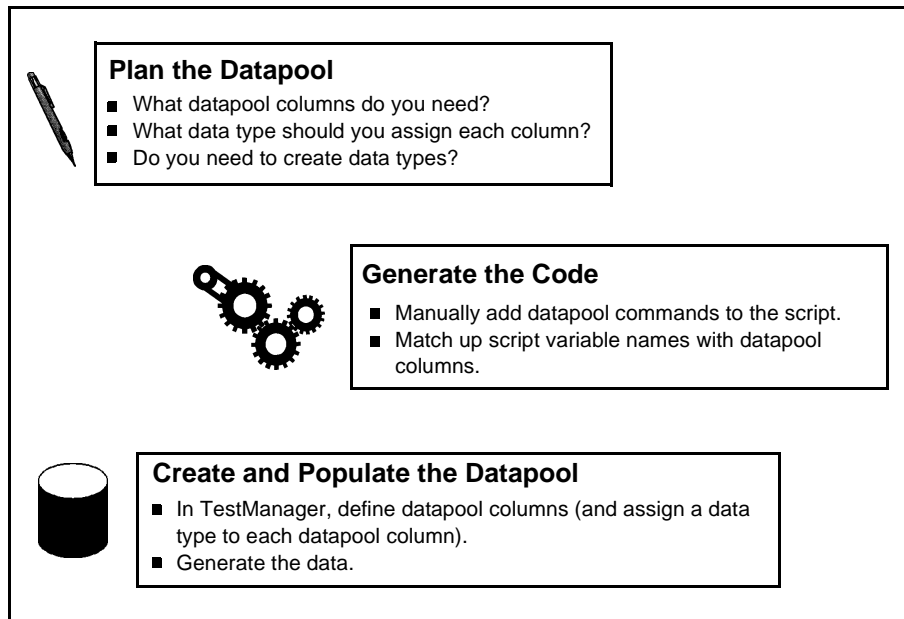
You also assign a data type to each datapool column. Data types supply a datapool column with its values. For information about data types, see *Data Types* on page 8-6.

- Generate the data. Once you define the datapool columns, you populate the datapool simply by clicking **Generate Data**.

For more information, see *Creating a Datapool with TestManager* on page 8-18.

NOTE: You can also create and populate a datapool file manually and import it into the repository. For more information, see *Creating a Datapool Outside Rational Test* on page 8-38.

The following figure illustrates the three stages of datapool creation:

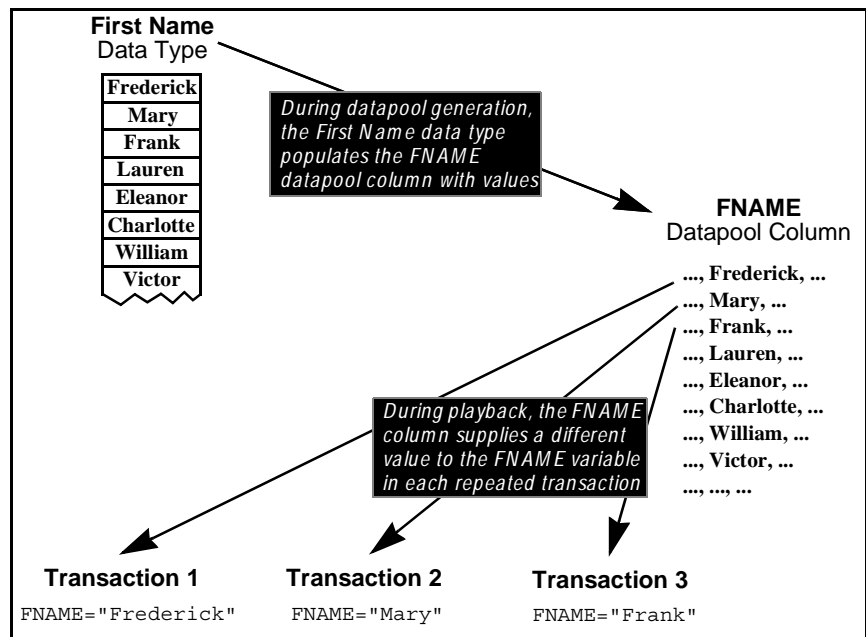


Data Types

A datapool **data type** is a source of data for one datapool column.

For example, the `Names - First` data type (shipped with Rational Test as a standard data type) contains a list of persons' first names. Suppose that you assign this data type to the datapool column `FN AME`. When Robot automatically generates the datapool, it populates the `FN AME` column with all of the values in the `Names - First` data type.

The following figure illustrates the relationship between data types, datapool columns, and the values assigned to script variables during playback:



Standard and User-Defined Data Types

There are two kinds of datapool data types, as follows:

- ▶ **Standard data types** that are included with Rational Test. These data types include commonly used, realistic sets of data in categories such as first and last names, company names, cities, and numbers.

For a list of the standard data types, see Appendix C.

- ▶ **User-defined data types** that you create. You must create a data type if none of the standard data types contain the kind of values you want to supply to a script variable.

User-defined data types are useful in situations such as the following:

- When a field accepts a limited number of valid values.

For example, suppose a datapool column supplies data to a script variable named *color*. This variable provides the server with the color of a product being ordered. If the product only comes in the colors red, green, blue, yellow, and brown, these are the only values that *color* can be assigned. No standard data type contains these exact values.

You could ensure that the *color* variable is assigned a valid value from the datapool by doing the following:

1. Before you create the datapool, create a data type named Colors that contains the five supported values (Red, Green, Blue, Yellow, Brown).
 2. When you create the datapool, assign the Colors data type to the datapool column COLOR. The COLOR column will supply values to the script's *color* variable.
- When you need to generate data that contains multi-byte characters, such as those used in some foreign-language character sets. For more information, see the section *Generating Multi-Byte Characters* on page 8-11.

Before you create a datapool, find out which standard data types you can use as sources of data and which user-defined data types you need to create. Although it is possible to create a data type while you are creating the datapool itself, the process of creating a datapool will be smoother if you create the user-defined data types first. For more information, see *Creating User-Defined Data Types* on page 8-9.

Finding Out What Data Types You Need

To decide whether to assign a standard data type or a user-defined data type to each datapool column, you need to know the kinds of values that will be supplied to script variables during playback — for example, whether a variable will contain names, dates, order numbers, and so on.

After recording a script, search the script for each value that you provided to the application during recording. Later, you will replace these literal values with variables. During playback, the variables will be supplied values from the datapool.

For information about replacing the recorded value with a variable that is supplied values from a datapool, see *Substituting Variables for Literal Values* on page 8-14.

Finding Values in GUI Scripts

The following are two examples of literal values in GUI scripts. The values are in bold type:

```
'Credit Card Type
ComboBox Click, "ObjectIndex=1", "Coords=104,7"
ComboBox Click, "ObjectIndex=1", "Text=Discover"

'Credit Card Expiration Date
EditBox Left_Drag, "ObjectIndex=4", "Coords=19,13,16,12"
InputKeys "12/31/99"
```

To simplify the task of searching for values, insert a descriptive comment into the script before providing a value to the client application during recording.

NOTE: The only values that Robot records are those that you specifically provide during recording. If you accept a default, Robot does not record that value.

Creating User-Defined Data Types

If none of the standard data types can provide the correct kind of values to a script variable, create a user-defined data type.

To create a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click **New**.
3. Type a name for the data type (40 characters maximum) and optionally, a description (255 characters maximum).
4. Click **OK**.
5. Click **Yes** when prompted to enter data values now.

The Edit Data Type dialog box appears. This dialog box supports Input Method Editor (IME) modes for typing multi-byte characters.

6. Type a data type value on the first blank line in the list.

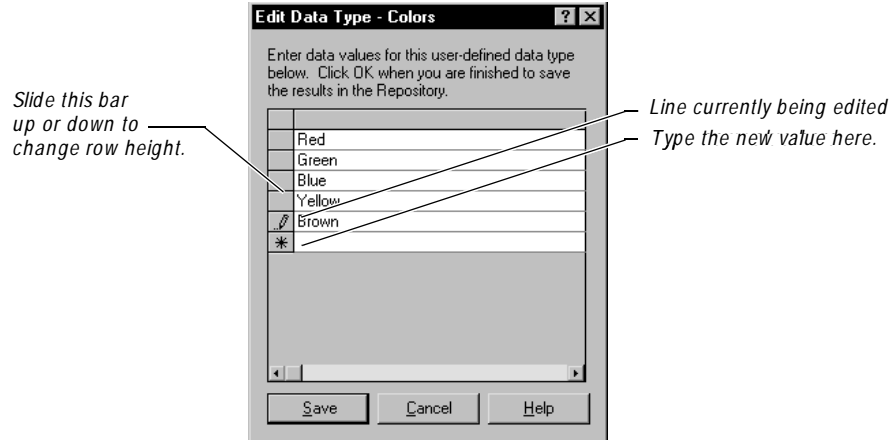


When you start typing the value, a pencil icon appears, indicating editing mode.



7. To type a new value, place the insertion point on the blank line next to the asterisk icon, and then type the value.
8. Repeat steps 6 and 7 until you have added all of the values.
9. Click **Save**.

The following figure shows the data type Colors being populated with five values:



When you create a user-defined data type, it is listed in the **Type** column of the Datapool Specification dialog box (where you define datapool columns). **Type** also includes the names of all the standard data types. User-defined data types are flagged in this list with an asterisk (*).

NOTE: You can assign data from a standard data type to a user-defined data type. For information, see *Editing Standard Data Type Values* on page 8-32.

Generating Unique Values from User-Defined Data Types

You may want a user-defined data type to supply unique values to a script variable during playback. To do so, the user-defined data type must contain unique values.

In addition, when you are defining the datapool in the Datapool Specification dialog box, make the following settings for the datapool column associated with the user-defined data type:

- ▶ Set **Sequence** to Sequential.
- ▶ Set **Repeat** to 1.
- ▶ Make sure that the **No. of records to generate** value does not exceed the number of unique values in your user-defined data type.

For information about the values you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 8-20.

Generating Multi-Byte Characters

If you want to include multi-byte characters in your datapool (for example, to support Japanese and other languages that include multi-byte characters), you can do so in one of the following ways:

- ▶ Through a user-defined data type. (For information, see *Creating User-Defined Data Types* on page 8-9.)

The editor provided for you to supply the user-defined data fully supports Input Method Editor (IME) operation. An IME lets you type multi-byte characters, such as Kanji and Katakana characters as well as multi-byte ASCII, from a standard keyboard. An IME is included in the Japanese version of Microsoft Windows.

- ▶ Through the Read From File data type. (For information, see *Creating a Column of Values Outside Rational Test* on page 8-42.)

Using Datapools with GUI Scripts

A GUI script can access a datapool when it is played back in Robot.

The following are the general tasks involved in providing access to a datapool from a GUI script. These tasks are not in a fixed order — you can create the datapool at any point:

- ▶ Record the GUI script.
- ▶ Add datapool commands to the script.
- ▶ Create the datapool.

The following sections provide information about recording a GUI script for datapool access and adding datapool commands to a GUI script.

For information about creating a datapool with TestManager, see *Creating a Datapool with TestManager* on page 8-18.

Recording a GUI Script

During GUI recording, as you provide values to the client application, follow the guidelines below. These guidelines will simplify the task of adding datapool commands to the script after you record it.

- ▶ Before you provide a value to the client application, insert a comment that describes the value you are providing. Later, when you are editing the script, comments will simplify the task of searching for the values you provided during recording.



To insert a comment, click the **Comment** button on the GUI Insert floating toolbar.

- ▶ Specify a value for each application field that is to be supplied with a datapool value during script playback. Do this even for fields that contain default values.

Remember, during GUI recording, that Robot records your GUI actions. If you do not act on a field that contains a default value, that field object and its default value will not appear in the script. You will either have to re-record that portion of the script or add the information to the script manually.

Adding Datapool Commands to a GUI Script

Once you have recorded values for all of the fields in the client application that require values from the datapool, edit the script and perform the following basic tasks:

- ▶ Reference the `sqautil.sbh` header file.
- ▶ Substitute variables for the literal values that you provided during recording.
- ▶ Add datapool commands that open the datapool, fetch a row of data from the datapool, retrieve the individual values in the fetched row, and assign each value to a script variable.

The following code fragment highlights the role of the primary datapool commands:

```
'$Include "sqautil.sbh"
Sub Main

    ... Declare variables with Dim statements

    ' Open a datapool named CD Orders
    dp=SQADatapoolOpen("CD Orders")

    ' Perform the transaction 100 times, using a new
    ' set of data from the datapool each time

    For x = 1 to 100

        ' Fetch a row from the datapool
        Call SQADatapoolFetch(dp)

        ' Begin the transaction

        ' Credit Card Number
        Window SetContext, "Caption=Make An Order", ""
        EditBox Click, "ObjectIndex=3", "Coords=13,11"
        ' Assign ccNum a value from datapool column #4
        Call SQADatapoolValue(dp,4,ccNum)
        InputKeys ccNum ' Pass the datapool value to the application

        ... ' Assign other datapool values to other variables

    Next x

    Call SQADatapoolClose(dp)

End Sub
```

For details about using these datapool commands and the `SQADatapoolRewind` command, see the *SQABasic Language Reference*.

Substituting Variables for Literal Values

The values that you provided during recording are included in the script as literal values. If you do not substitute a variable for a literal value, the literal value is sent to the server each time the transaction is executed.

The recorded literal values are represented in the script in various ways. For example, if you type a value into an edit box, the `InputKeys` command specifies the characters you typed. If you click an item in a combo list box, the value is specified in the `parameters` argument of `ComboBox`.

Edit Box Example

The following is an example of how Robot records the value `Fred` as it is typed into an edit box:

```
'Customer's First Name  
EditBox Click, "ObjectIndex=5", "Coords=104,12"  
InputKeys "Fred"
```

And the following is an example of replacing that literal value with the variable `fName`:

```
'Customer's First Name  
EditBox Click, "ObjectIndex=5", "Coords=104,12"  
Call SQADatapoolValue(dp,1,fName)  
InputKeys fName
```

Combo List Box Example

The following is an example of how Robot records the value `Discover` as it is selected from a list of credit card types:

```
'Credit Card Type  
ComboBox Click, "ObjectIndex=1", "Coords=104,7"  
ComboBox Click, "ObjectIndex=1", "Text=Discover"
```

And the following is an example of replacing that literal value with the variable `ccType`:

```
'Credit Card Type  
ComboBox Click, "ObjectIndex=1", "Coords=104,7"  
Call SQADatapoolValue(dp,5,ccType)  
ComboBox Click, "ObjectIndex=1", "Text=" + ccType
```

Assigning Datapool Values to Variables

Once you substitute variables for the literal values that you recorded, you assign datapool values to the variables. You do so through the `SQADatapoolFetch` and `SQADatapoolValue` commands. Use these commands as follows:

- ▶ Call `SQADatapoolFetch` to retrieve an entire row of values (also called a record) from the datapool.
- ▶ Call `SQADatapoolValue` to retrieve an individual value from the fetched datapool row and assign it to a script variable.

For example, suppose a datapool row consists of three columns of values: Part Number, Part Name, and Unit Price.

1. At the beginning of the transaction, just before the lines of code where Robot recorded these three values, call `SQADatapoolFetch`.
2. Next, call `SQADatapoolValue` three times — once for each of the three datapool columns that you are accessing in the fetched row. `SQADatapoolValue` retrieves a value from the specified column in the fetched row and assigns the value to a script variable.

In the following example, `SQADatapoolValue` retrieves a value from the first column in the fetched datapool row and assigns the value to the variable `fName`:

```
'Customer's First Name
EditBox Click, "ObjectIndex=5", "Coords=104,12"
Call SQADatapoolValue(dp,1,fName)
InputKeys fName
```

Optionally, `SQADatapoolValue` can refer to the column by column name rather than by column number. In the following example, the datapool column name `fName` matches the variable name `fName`:

```
Call SQADatapoolValue(dp,"fName",fName)
```

If you refer to the datapool column by name, the reference must match the datapool column name exactly, including a case match.

Datapool column names and column numbers are indicated in the TestManager Datapool Specification dialog box and in the TestManager Edit Datapool dialog box.

Example GUI Script

The following GUI script was edited to access the CD Orders datapool illustrated in *Example of Datapool Column Definition* on page 8-23:

```
'$Include "squtil.sbh"

Sub Main
  Dim Result As Integer

  'Initially Recorded: 05/06/98 17:56:15
  'Script Name: CD Order

  Dim x as Integer
  Dim dp as Long           ' Reference to datapool

  'Variables to be assigned data from datapool
  Dim ccNum as String
  Dim ccType as String
  Dim ccExpDate as String
  Dim fName as String
  Dim lName as String
  Dim custID as String

  ' Open a datapool named CD Orders
  dp=SQADatapoolOpen("CD Orders")

  ' Execute transaction 100 times.
  For x = 0 to 99

    ' Fetch a row from the datapool
    Call SQADatapoolFetch(dp)

    'Begin the order
    Window SetContext, "Caption=Classics Online;Class=ThunderForm", ""
    PushButton Click, "Text=Order It!"
    Window SetContext, "Caption=Classics Login", ""
    PushButton Click, "Text=OK"

    ' The following section uses data from the CD Orders datapool

    'Credit Card Number
    Window SetContext, "Caption=Make An Order", ""
    EditText Click, "ObjectIndex=3", "Coords=13,11"
    Call SQADatapoolValue(dp,4,ccNum)
    InputKeys ccNum

    'Credit Card Type
    ComboBox Click, "ObjectIndex=1", "Coords=104,7"
    Call SQADatapoolValue(dp,5,ccType)
    ComboBox Click, "ObjectIndex=1", "Text=" + ccType

    'Credit Card Expiration Date
    EditText Left_Drag, "ObjectIndex=4", "Coords=19,13,16,12"
    Call SQADatapoolValue(dp,6,ccExpDate)
    InputKeys ccExpDate

    'Customer's First Name
    EditText Click, "ObjectIndex=5", "Coords=104,12"
    Call SQADatapoolValue(dp,1,fName)
    InputKeys fName
```

```

'Customer's Last Name
EditBox Left_Drag, "ObjectIndex=6", "Coords=67,4,-309,15"
Call SQADatapoolValue(dp,2,lName)
InputKeys lName

'Customer's ID
EditBox Left_Drag, "ObjectIndex=7", "Coords=115,11,-305,20"
Call SQADatapoolValue(dp,3,custID)
InputKeys custID

'Place the order
PushButton Click, "Text=Place Order"

'Acknowledge the placement of the order
Window SetContext, "Caption=Classics Online;Class=#32770", ""
PushButton Click, "Text=OK"

Next x          ' End the current transaction

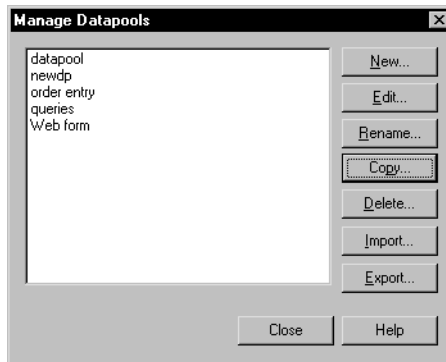
Call SQADatapoolClose(dp)

End Sub

```

Managing Datapools

You manage datapools in the TestManager Manage Datapools dialog box.



The activities that you perform in this dialog box affect the datapools stored in the repository. For information about where datapools are stored, see *Datapool Location* on page 8-29.

Creating a Datapool with TestManager

To create and automatically populate a datapool with TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Click **New**.
3. Type a name for the datapool (40 characters maximum) and optionally, a description (255 characters maximum).
4. Click **OK**.
5. Click **Yes** to acknowledge that you want to define the datapool now.

The Datapool Specification dialog box appears. This dialog box lets you define the columns in the datapool file. Datapool column definitions are listed as rows in this dialog box. Datapool columns are also called *fields*.

6. Click **Insert before** or **Insert after** to add a datapool column to the datapool.
7. Type a name for the new datapool column (40 characters maximum).
8. Assign a data type to the datapool column, and define any other properties as needed.

For information about the properties that you can define for a datapool column, see *Defining Datapool Columns* on page 8-20.

9. Repeat steps 6 through 8 until you have defined all of the columns in the datapool.

To see an example of datapool columns defined in the Datapool Specification dialog box, see *Example of Datapool Column Definition* on page 8-23.

10. Type a number in the **No. of records to generate** box.

Alternatively, if you do not want to generate any data now, click **Save** to save your datapool column definitions, and then click **Close**.

11. When finished defining datapool columns, click **Generate Data**.

You cannot generate data for a datapool that has more than 150 columns.

12. Optionally, click **Yes** to see a brief summary of the generated data.

If There Are Errors

If the datapool values are not successfully generated, you are prompted to see an error report rather than a summary of the generated data. To correct the errors:

1. Click **Yes** to see the error report.
2. After viewing the cause of the errors, click **Cancel**.
3. Correct the errors in the **Datapool Fields** grid.

Viewing Datapool Values

To see the generated values, close the Datapool Specification dialog box. In the Manage Datapools dialog box, select the datapool that you just created, click **Edit**, and then click **Edit Datapool Data**.

If a datapool includes complex values (for example, embedded strings, or field separator characters included in datapool values), you should view the datapool values to make sure that the contents of the datapool are as you expect.

Making the Datapool Available to a Script

For a script to be able to access the datapool you create with TestManager, the script must contain datapool commands, such as commands for opening the datapool and fetching values.

For information about adding datapool commands to a GUI script, see *Using Datapools with GUI Scripts* on page 8-12.

Defining Datapool Columns

The following table provides information to help you define datapool columns in the Datapool Specification dialog box.

Grid column	Description
Name	<p>The name of a datapool column (and its corresponding script variable).</p> <p>If you change the name of a datapool column, be sure the new name matches all instances of its corresponding script variable.</p> <p>If you create a datapool outside of the Rational Test environment and then import it, TestManager automatically assigns default names to the datapool columns. Use Name to match the imported datapool column names with their corresponding script variables. Names are case-sensitive.</p> <p>You can use an IME to type multi-byte characters in datapool field names. (With GUI scripts, you can associate datapool column names and script variables through column position rather than column name. For more information, see the description of the <code>SQADatapoolValue</code> command in the <i>SQABasic Language Reference</i>.)</p>
Type	<p>The standard or user-defined data type that supplies values to the datapool column in Name. User-defined data types are marked with an asterisk (*).</p> <p>Specify the data type to assign to the datapool column, as follows:</p> <ul style="list-style-type: none"> ▶ To select a standard data type or an existing user-defined data type, click the currently displayed data type name, and then select the new data type from the drop-down list: <div style="border: 1px solid black; padding: 2px; margin: 5px 0;">Random alphanumeric str1 ▾</div> <p>See Appendix C for a description of the standard data types.</p> <p>If you type rather than select the name of a user-defined data type, enter an asterisk before the user-defined data type name. For example, to specify the user-defined data type MyData, type:</p> <p style="margin-left: 20px;">*MyData</p> ▶ To create a new user-defined data type, enter the data type name (without the asterisk) in the field, and then press RETURN. After you click Yes to confirm that you want to create a user-defined data type, the Data Type Properties - Edit dialog box appears. <p>For information about creating a data type, see <i>Creating User-Defined Data Types</i> on page 8-9.</p>

(Continued)

Grid column	Description
Sequence	<p>The order in which the values in the data type specified in Type are written to the datapool column. Select one of the following options from the drop-down list:</p> <ul style="list-style-type: none"> ▶ Random – Writes numeric and alphanumeric values to the datapool column in any order. ▶ Sequential – Writes numeric values sequentially (for example, 0, 1, 2...). With decimal numbers, the sequence is based on the lowest possible decimal increment (for example, with a Decimals value of 2, the sequential values are 0.00, 0.01, 0.02, ...). <p>Sequential is only supported for numeric values (including date and time values) and values generated from user-defined data types.</p> <p>When you select Sequential with numeric data types, and you specify a Minimum and Maximum range, Interval must be greater than 0.</p> <ul style="list-style-type: none"> ▶ Unique – With data type Integers - Signed, ensures that numbers written to the datapool column are unique. Also, set Repeat to 1, and define a Minimum and Maximum range.
Repeat	<p>The number of times a given value can appear in a datapool column. Repeat cannot be set to 0.</p> <p>To make values unique with Integers - Signed data types and user-defined data types, set Repeat to 1. For unique Integers - Signed values, also set Sequence to either Sequential or Unique.</p> <p>When defining unique values, make sure that the number of rows you are generating is not higher than the range of possible unique values.</p>
Length	<p>The maximum number of characters that a value in the datapool column can have. If the datapool column contains numeric values, Length specifies the maximum number of characters a number can have, <i>including</i> a decimal point and minus sign, if any.</p> <p>For example, for decimal numbers as high as 999.99, set Length to 6. For decimal numbers as low as -999.99, set Length to 7.</p> <p>Length cannot be 0.</p>
Decimals	<p>Specifies the maximum number of decimal places that floating point values can have. The maximum setting is 6 decimal places.</p>

(Continued)

Grid column	Description
Interval	<p>Writes a sequence of numeric values to the datapool column. The sequence increments by the Interval you set. For example, if Interval is 10, the datapool column contains 0, 10, 20, and so on. If Interval is 10 and Decimal is 2, the datapool column contains 0.00, 0.10, 0.20, and so on.</p> <p>Minimum interval is 1. Maximum interval is 999999.</p> <p>With numeric data types (including dates and times), when Sequence is set to Sequential and you specify a Minimum and Maximum range, Interval must be greater than 0.</p> <p>Use Interval only with numeric values (including dates and times).</p>
Minimum	<p>Specifies the lowest in a range of numeric values. For example, if the datapool column supplies order number values, and the lowest possible order number is 10000, set Type to Integer - Signed, Minimum to 10000, and Maximum to the highest possible order number.</p> <p>Use Minimum only with numeric values (including dates and times).</p>
Maximum	<p>Specifies the highest in a range of numeric values. For example, if the datapool column supplies values to a variable named <i>ounces</i>, set Type to Integer - Signed, Minimum to 0, and Maximum to 16.</p> <p>Use Maximum only with numeric values (including dates and times).</p>
Seed	<p>The number that Rational Test uses to compute random values. The same seed number always results in the same random sequence. To change the random sequence, change the seed number.</p> <p>When Type is String Constant, use Seed to provide the alphanumeric value of the constant. For example, if you want to insert the constant “Rational Software” into every row of a datapool column, set Type to String Constant and type Rational Software into Seed.</p>
Data File	<p>The path to the user-defined data type file. The path is automatically inserted for you. This field is not modifiable.</p> <p>Data type files are stored in the Datatype directory of your project. You never have to modify these files directly.</p>

Some items might not be modifiable, depending on the data type that you select. For example, if you select the Names - First data type, you cannot modify **Decimals**, **Interval**, **Minimum**, or **Maximum**.

If you are generating unique values for an Integers - Signed data type, **Length**, **Minimum**, **Maximum**, and **No. of records to generate** must be consistent. For example, if you want unique numbers from 0 through 999, errors may result if you set **Length** to 1, **Maximum** to 5000, and/or if you set **No. of records to generate** to a number greater than 1000.

NOTE: You can use an IME to type multi-byte characters into the **Name** column only. The IME is automatically disabled when you are editing any other column.

Example of Datapool Column Definition

Suppose you want to record a transaction in which a customer purchase is entered into a database. During recording, you supply the client application with the following information about the customer:

- ▶ Customer name
- ▶ Customer ID
- ▶ Credit card number
- ▶ Credit card type
- ▶ Credit card expiration date

After you record the script and edit it for datapool access, you are ready to create the datapool. Following the instructions on page 8-18, you define the datapool's columns in the Datapool Specification dialog box, as illustrated below:

Float data type with 0 decimals is used for credit card numbers requiring 16 digits

Customer ID is unique

Date range

Datapool column 1

Generate 1000 datapool rows

The only user-defined data type needed

		Datapool Fields							
Name	Type	Sequence	Repeat	Length	Decimals	Interval	Minimum	Maximum	
iName	Names - First	Random	1	20	0	0	0	0	
iName	Names - Last	Random	1	20	0	0	0	0	
custID	Integer - Signed	Unique	1	7	0	0	1000000	9999999	
ccNum	Float - XXXXX	Random	1	16	0	0	1000000000000000	9999999999999999	
ccType	*Credit Card Type	Random	1	15	0	0	0	0	
ccExpDate	Date - MM/DD/YYYY	Random	1	15	0	0	07011998	12312002	

No. of records to generate:

In this example, note the following highlights of the datapool column definition:

- ▶ **fName** column. The standard data type `Names - First` supplies this datapool column with masculine and feminine first names.
- ▶ **Iname** column. The standard data type `Names - Last` supplies this datapool column with surnames.
- ▶ **custID** column. The standard data type `Integer - Signed` supplies ID numbers to this datapool column. Because all customer IDs in this example consist of seven digits, the **Minimum** and **Maximum** range is set from 1000000 through 9999999. Also, because all IDs must be unique, **Sequence** is set to `Unique`.

NOTE: **Sequence** can only be set to `Unique` for `Integer - Signed` data types.

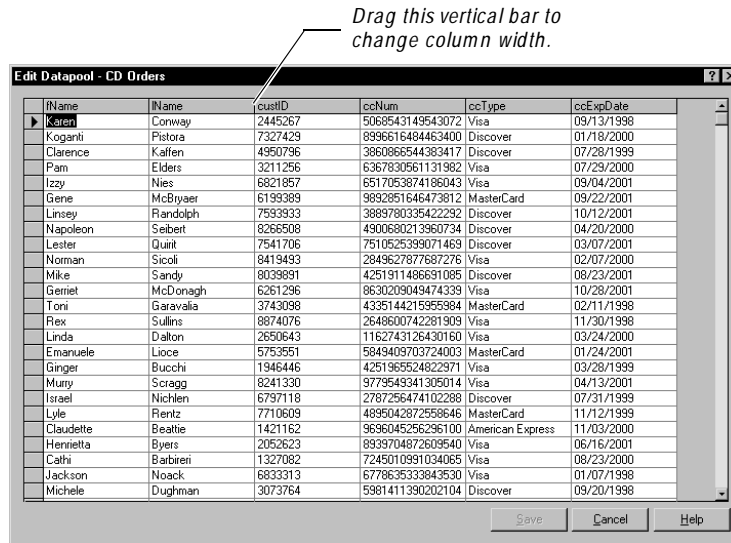
- ▶ **ccNum** column. The `Integer - Signed` data type generates numbers up to nine digits. Because credit card numbers contain more than nine digits, the standard data type `Float X.XXX` is used to supply credit card numbers to this datapool column. **Decimals** is set to 0 so that only whole numbers are generated. **Sequence** is set to `Random` to generate random card numbers. To generate unique numbers, **Repeat** is set to 1.
- ▶ **ccType** column. This is the only datapool column that needs to have values supplied from a user-defined data type. The user-defined data type `Credit Card Type` contains just four values — American Express, Discover, MasterCard, and Visa.
- ▶ **ccExpDate** column. The standard data type `Date - MM/DD/YYYY` supplies credit card expiration dates to this datapool column. The range of valid expiration dates is set from July 1, 1998 through December 31, 2002. **Sequence** is set to `Random` to generate random dates.

Example of Datapool Value Generation

After you define datapool columns in the Datapool Specification dialog box, click **Generate Data** to generate the datapool values. To see the values that you generated:

1. Click **Close**.
2. Click **Edit Datapool Data**.

This is what you see:



Editing Datapool Column Definitions with TestManager

To edit the definitions of the columns in an existing datapool with TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Select the datapool to edit, and then click **Edit**.
3. Click **Define Datapool Fields**.

The Datapool Specification dialog box appears. This dialog box lets you define the columns in the datapool file. Datapool column definitions are listed as rows in this dialog box. Datapool columns are also called *fields*.

4. Edit one or more datapool column definitions, as described in the table in *Defining Datapool Columns* on page 8-20.
5. When you have finished editing datapool column definitions, take one of the following actions:
 - To save the datapool column definitions but not generate any data, click **Save**, and then click **Close**.
 - To save the datapool column definitions and generate data, type a number in the **No. of records to generate** field, and then click **Generate Data**. Optionally, click **Yes** to see a brief summary of the generated data.

NOTE: To see the generated values, close the Datapool Specification dialog box. In the Datapool Properties - Edit dialog box, click **Edit Datapool Data**.

If There Are Errors

If the datapool values are not successfully generated, you are prompted to see an error report rather than a summary of the generated data. To correct the errors:

1. Click **Yes** to see the error report.
2. After viewing the cause of the errors, click **Cancel**.
3. Correct the errors in the **Datapool Fields** grid.

Deleting a Datapool Column

Datapool column definitions are listed as rows in the Datapool Specification dialog box.

To delete a datapool column definition from the list:

1. Click anywhere in the row to be deleted.
2. Click the gray box to the left of the datapool column name. This action selects the entire row.
3. Press the `DELETE` key.

You are not prompted to confirm the deletion.

Editing Datapool Values with TestManager

To view or edit the values in an existing datapool with the TestManager editor:

1. Click **Tools** → **Manage Datapools**.
2. Select the datapool to edit, and then click **Edit**.
3. Click **Edit Datapool Data** in the Datapool Properties - Edit dialog box.

The Edit Datapool dialog box appears.

4. Modify the datapool values as necessary. Note that:

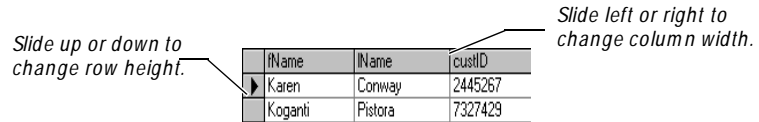


- When you click a value to edit it, an arrow icon appears to the left of the row you are editing.



- When you begin to edit the value, a pencil icon appears to the left of the row, indicating editing mode.
- To undo the changes that you just made to a value, press `CTRL + Z` *before* you move the insertion point out of the field.

- To see the editing menu, select the text to edit, and then right-click the mouse.
- To increase the width of a column, move the bar that separates column names. To increase the height of a row, move the bar that separates rows:



5. To delete an entire datapool row:
 - a. Click the gray box to the left of the first value in the row that you are deleting. This action selects the entire row.
 - b. Press the DELETE key.
6. When you have finished modifying datapool values, click **Save**, and then click **Close**.

For an example of the datapool values that TestManager generates, see *Example of Datapool Value Generation* on page 8-24.

Canceling Your Edits

To abandon all of the edits that you made in the Edit Datapool dialog box, click **Cancel** or the ESC key. With either action, all your edits are abandoned, and the Edit Datapool dialog box closes.

Renaming a Datapool

To rename a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to rename.
3. Click **Rename**.
4. Type the datapool's new name (40 characters maximum).
5. Click **OK**, and then click **Close**.

Copying a Datapool

To copy a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to copy.

3. Click **Copy**.
4. Type a name for the new datapool (40 characters maximum).
5. Click **OK**, and then click **Close**.

Deleting a Datapool

To delete a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to delete.
3. Click **Delete**.
4. Click **Yes** to confirm the deletion, and then click **Close**.

Deleting a datapool removes the datapool .csv and .spc files plus all references to the datapool from the repository.

Importing a Datapool

Robot automatically creates and populates datapools for you. However, it is possible for you to create and populate a datapool yourself, using a tool such as Microsoft Excel. For example, you might want to export data from your database into a .csv file, and then use that file as your datapool.

If you create a datapool yourself, you need to import it into the same repository as the scripts that will access it. Use TestManager to import a datapool .csv file.

To import a datapool .csv file using TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Click **Import**.
3. In the **Look in** box, specify the directory where the datapool you created is located.
4. In **File name**, specify the name of the datapool .csv file.
5. Click **Open**.

The Datapool Properties - Import dialog box appears and displays the datapool name.

6. Accept the default datapool name or type a new one (40 characters maximum).

7. In **Field Separator**, make sure that the character(s) displayed are the same as the field separator used in the .csv file that you are importing as a datapool.

For information about field separators in datapools, see *Datapool Structure* on page 8-39.

8. Optionally, type a description of the datapool (255 characters maximum).
9. Click **OK**, and then click **Close**.

When you import a datapool, you often have to change the names of the datapool columns to match the names of the corresponding script variables. For more information, see *Matching Datapool Columns with Script Variables* on page 8-42.

Datapool Location

When you import a datapool, TestManager copies the datapool's .csv file to the Datapool directory associated with the current repository and project.

For example, if the current repository is MyRepo, and if the current project directory is MyProject, the datapool is stored in the following directory:

```
C:\MyRepo\Project\MyProject\Datapool
```

This directory also includes the datapool's specification (.spc) file. When you create and then import a .csv file, TestManager automatically creates the .spc file for you. You should never edit the .spc file directly.

NOTE: After you import a datapool, the original file that you used to populate the datapool remains in the directory that you specified when you saved it. The Rational Test software has no further need for this file.

Importing a Datapool from Another Project

Use the TestManager Import feature to copy a datapool that you created for one project into another. When you import a datapool into a new project, the source datapool is still available to the original project.

NOTE: If the datapool that you are importing includes user-defined data types, import the data types *before* you import the datapool. For information, see *Importing a User-Defined Data Type* on page 8-35.

To import a datapool into a new project:

1. Run TestManager. (By default, click **Start** → **Programs** → *Rational product name* → **Rational Test 7** → **TestManager**.)
2. In the Rational Repository Login dialog box, select the repository and project that you are importing the datapool to.
3. Click **File** → **Import Test Assets**. The Import Test Assets wizard appears.
4. Select **Datapool** in the **Asset Type** list.
5. Click the appropriate overwrite option. This option determines whether to overwrite an existing datapool with the same name as the one you are importing.
6. Click **Next**.
7. In the Login dialog box, provide the path, project name, and your login information for the repository that contains the datapool you are importing.

If you are importing between projects in the same repository, you do not need to provide a user ID and password.
8. Click **Next**.
9. Select one or more datapools to import. (To select multiple datapools, hold down the CTRL key while clicking each datapool to import.)
10. Click **Next**.
11. Click **Finish** after reading the information summarizing the import operation.

Exporting a Datapool

Use the TestManager Export feature to copy a datapool to any directory on your computer. When you export a datapool, the original datapool remains in its repository and project.

Do not attempt to export a datapool to another Rational Test project. Instead, use the import feature to import the datapool into the new project. For more information, see *Importing a Datapool* on page 8-28.

To export a datapool to a location on your computer:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the datapool to export.
3. Click **Export**.
4. In **Save In**, specify the directory where you want to copy the datapool.

5. Click **Save**.
6. Click **OK** to acknowledge that the datapool was exported to the correct location.

Managing Data Types

You can use TestManager to manage data types, as follows:

- ▶ With user-defined data types, you can edit data type values and data type definitions. You can also import, rename, copy, and delete user-defined data types.

For information about creating user-defined data types, see *Creating User-Defined Data Types* on page 8-9.

- ▶ With standard data types, you can only edit data type values, and you can only do so indirectly, through a user-defined data type.

Editing User-Defined Data Type Values

If you want to add, remove, or modify data type values, or if you just want to modify the optional description, edit the data type. You can edit only user-defined data types, not standard data types.

To edit a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Select the data type to edit, and then click **Edit**.
3. Optionally, modify the data type description.
4. Click **Edit Data Type Data**. A dialog box appears with the name of the data type you are editing.
5. The arrow icon points to the line that is currently available for editing. Edit that line, or place the insertion point in another line.



You can use the keyboard's up and down arrow keys to move the insertion point.

6. To see the editing menu, select the text to edit, and then right-click the mouse.
7. To delete a value, place the insertion point anywhere in the value, click the gray box to the left of the value, and then press the DELETE key.
8. When you have finished making changes, click **Save**.
9. Click **OK** in the Data Type Properties - Edit dialog box, and then close the Manage Data Types dialog box.

Editing Standard Data Type Values

You cannot edit the values in a standard data type directly. However, you can accomplish the same task by copying the values in a standard data type to a user-defined data type, and then modifying the values in the user-defined data type.

The following procedure shows how you can modify the values in the standard data type **Cities – U.S.**:

1. Click **Tools** → **Manage Data Types**, and then click **New**.
2. Click the name of your new data type (in this case, **Custom US Cities**). Optionally, type a description, and then click **OK**.
3. Click **No** when prompted to enter data into the new data type now.
4. Select the data type you just created, and then click **Edit**.
5. Click **Define Data Type Field**.
6. In the **Type** column, use the drop-down list to select the name of the standard data type that contains the values you want to modify (in this case, **Cities – U.S.**).
7. In **No. of records to generate**, type the number of values to copy from the standard data type to the new data type, and then click **Generate Data**.
8. Click **Close** in the Datapool Specification dialog box, and then click **Edit Data Type Data**.
9. Add, delete, and modify the standard data type values as appropriate.
10. When you have finished making changes, click **Save**.
11. Click **OK** in the Data Type Properties - Edit dialog box, and then close the Manage Data Types dialog box.

To use the modified values when assigning a data type to a datapool column in the Datapool Specification dialog box, assign the user-defined data type that you just created, not the standard data type.

Editing User-Defined Data Type Definitions

Like all data types, a user-defined data type is essentially a one-column datapool. The single column contains the values that you type into the user-defined data type.

You can edit the default definition of the data type column in the Datapool Specification dialog box, just as you edit the default definition of datapool columns.

If you edit the definition of a user-defined data type, and then generate values for the data type, you overwrite any existing definitions and values for the data type.

How to Edit User-Defined Data Type Definitions

To edit the definition of a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Select the data type to edit, and then click **Edit**.
3. Click **Define Data Type Field**.

The Datapool Specification dialog box appears. Note that the **Insert Before** and **Insert After** buttons are not present; this is because a user-defined data type file can have only one column of values.

4. Define the fields in the data type. Use the table in *Defining Datapool Columns* on page 8-20 to help you define the fields.
5. When you have finished defining the data type, click **Generate Data**.
Optionally, click **Yes** to see a brief summary of the generated data.
6. Click **Close** in the Datapool Specification dialog box, and then click **Edit Data Type Data**.
7. Make any necessary modifications to the generated values.
8. When you have finished modifying values, click **Save**.
9. Click **OK** in the Data Type Properties - Edit dialog box, and then close the Manage Data Types dialog box.

Generating Values for a User-Defined Data Type

You can add values to a user-defined data type by supplying it with values from a standard data type. Doing so can reduce the typing that you need to perform when adding values to the user-defined data type.

For example, suppose you want to create a user-defined data type containing a list of valid product IDs. The valid ID numbers range from 1000001 through 1000100. However, there is a dash between the fourth and fifth digits (such as 1000-001).

Rather than type in all 100 numbers, with dashes, you can have TestManager generate the numbers and assign them to a user-defined data type. Then, all you then have to do is edit the data type values and add the dash to each ID.

Use the following steps to guide you through the process:

1. In TestManager, click **Tools** → **Manage Data Types**.
2. Click **New**.
3. Type a name for your user-defined data type (for example, Item ID).
4. Click **OK**.
5. Click **No** when prompted to enter data type values now.
6. Click the name of your new data type, and then click **Edit**.
7. Click **Define Data Type Field**.
8. Set the following column values in the grid (or accept the defaults):
 - **Type** = Integers - Signed
 - **Sequence** = Sequential
 - **Repeat** = 1
 - **Length** = 7
 - **Interval** = 1
 - **Minimum** = 1000001
 - **Maximum** = 1000100
9. Type 100 in **No. of records to generate**.
10. Click **Generate Data**.
11. Click **No** to decline to see data generation details, and then click **Close**.
12. Click **Edit Data Type Data** in the Data Type Properties dialog box.
13. Type a dash character between the fourth and fifth characters of each value.

NOTE: You can also assign the standard data type Read From File to a user-defined data type. For information about using the Read From File data type, see *Creating a Column of Values Outside Rational Test* on page 8-42.

Importing a User-Defined Data Type

You can import a user-defined data type from one project into another. When you import a user-defined data type into a new project, the source data type is still available to the original project.

To import a user-defined data type into a new project:

1. Run TestManager. (By default, click **Start** → **Programs** → *Rational product name* → **Rational Test 7** → **TestM anager**.)
2. In the Rational Repository Login dialog box, select the repository and project that you are importing the user-defined data type to.
3. Click **File** → **Import Test Assets**. The Import Test Assets wizard appears.
4. Select **Data type** in the **Asset Type** list.
5. Click the appropriate overwrite option. This option determines whether to overwrite an existing datapool with the same name as the one you are importing.
6. Click **Next**.
7. In the Login dialog box, provide the path, project name, and your login information for the repository that contains the data type you are importing.

If you are importing between projects in the same repository, you do not need to provide a user ID and password.
8. Click **Next**.
9. Select one or more data types to import. (To select multiple data types, hold down the CTRL key while clicking each data type to import.)
10. Click **Next**.
11. Click **Finish** after reading the information summarizing the import operation.

Renaming a User-Defined Data Type

To rename a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to rename.
3. Click **Rename**.
4. Type the data type's new name (40 characters maximum).
5. Click **OK**, and then click **Close**.

Copying a User-Defined Data Type

To copy a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to copy.
3. Click **Copy**.
4. Type the name of the new data type (40 characters maximum).
5. Click **OK**, and then click **Close**.

Deleting a User-Defined Data Type

To delete a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to delete.
3. Click **Delete**.
4. Click **Yes** to confirm the deletion, and then click **Close**.

Generating and Retrieving Unique Datapool Rows

Many database tests work best when each row of test data is unique. For example, if a test involves repeating a transaction that adds customer orders to a database, each new order has to be unique — in other words, at least one field in the new record has to be a “key” field containing unique data.

When you are defining datapool columns in the Datapool Specification dialog box, you specify whether a given datapool column should contain unique data. If you specify that one or more columns should contain unique data, the datapool that the Rational Test software generates is guaranteed to contain unique rows.

However, even when a datapool contains all unique rows, it is possible for duplicate rows to be supplied to a script at test runtime.

To generate and retrieve unique datapool rows, you need to perform a few simple tasks when you define the datapool, as described in the next section.

What You Can Do to Guarantee Unique Row Retrieval

To ensure that a datapool supplies only unique rows to scripts at runtime, use the guidelines in the following table.

What to do	How to do it
Specify at least one column of unique data.	<p>In the Datapool Specification dialog box, specify that at least one datapool column should contain unique data. Unique data can be supplied through the Integers - Signed data type, through the Read From File data type, and through user-defined data types.</p> <p>With the Integers - Signed data type, take all of the following actions:</p> <ul style="list-style-type: none"> ▶ Set Sequence to Unique or Sequential. ▶ Set Repeat to 1. ▶ If Sequence= Unique, set an appropriate range in Minimum and Maximum. ▶ Make sure that the values of Length and No. of records to generate are appropriate for the set of numbers to generate. <p>For information about the Read From File data type, see <i>Generating Unique Values</i> on page 8-44.</p> <p>For information about user-defined data types, see <i>Generating Unique Values from User-Defined Data Types</i> on page 8-10.</p>
Generate enough datapool rows.	<p>Generate at least as many unique datapool rows as the number of times the datapool will be accessed during a test.</p> <p>For example, if a script repeats a transaction 100 times, and if each transaction sends one data record to the server, the datapool must contain at least 100 rows.</p> <p>You specify the number of rows to generate in the No. of records to generate field of the Datapool Specification dialog box.</p>
Disable cursor wrapping.	<p>If the datapool cursor wraps after the last row in the datapool has been accessed, previously fetched rows are fetched again.</p> <p>When editing a GUI script in Robot, disable cursor wrapping by setting the <i>wrap</i> argument of the <code>SQADatapoolOpen</code> command to False.</p>

(Continued)

What to do	How to do it
Use sequential or shuffle access order.	<p>With sequential or shuffle access, each datapool row is referenced in the row access order just once. When the last row is retrieved, the datapool cursor either wraps or datapool access ends.</p> <p>With random access, rows can be referenced in the access order multiple times. Thus, a given row can be retrieved multiple times.</p> <p>When editing a GUI script in Robot, set the <i>sequence</i> argument of the <code>SQADatapoolOpen</code> command to either <code>SQA_DP_SEQUENTIAL</code> or <code>SQA_DP_SHUFFLE</code>.</p>
Do not rewind the cursor during a test.	If you rewind the datapool cursor during a test (through the <code>SQABasicSQADatapoolRewind</code> command), previously accessed rows will be fetched again.

NOTE: Rational Test can guarantee that a datapool contains unique rows only when you generate datapool data through TestManager.

Creating a Datapool Outside Rational Test

To create a datapool file and populate it with data, you can use any text editor (such as Windows Notepad) or any application (such as Microsoft Excel or Microsoft Access) that can save data in .csv format.

For example, you can create a datapool file and type in the data, row by row and value by value. Or, you can export data from your database into a .csv file that you create with a tool such as Excel.

After you create and populate a datapool, use TestManager to import the datapool into the repository. For information about importing a datapool, see *Importing a Datapool* on page 8-28.

Datapool Structure

A datapool is stored in a text file with a .csv extension. The file has the following characteristics:

- ▶ Each row contains one record.
- ▶ Each record contains datapool field values delimited by a field separator. Any character can be used for the field separator. Some common field separators are:
 - Comma (,). This is typically the default in the United States and the United Kingdom.
 - Semi-colon (;). This is typically the default in most other countries.
 - Colon (:).
 - Pipe (|).
 - Slash (/).

The field separator can consist of up to three single-byte ASCII characters or one multi-byte character.

NOTE: To view or change the field separator, click **Start** → **Settings** → **Control Panel**, double-click the **Regional Settings** icon, and then click the **Number** tab. **List separator** contains the separator character(s).

- ▶ Each column in a datapool file contains a list of datapool field values.
- ▶ Field values can contain spaces.
- ▶ A single value can contain a separator character if the value is enclosed in double quotes. For example, “Jones, Robert” is a single value in a record, not two.

The quotation marks are used only when the value is stored in the datapool file. The marks are not part of the value that is supplied to your application.
- ▶ A single value can contain embedded strings. For example, “Jones, Robert “Bob”” is a single value in a record, not two.
- ▶ Each record ends with a line feed.
- ▶ Datapool column names are stored in a .spc file. (Robot and TestManager edit the .spc file. You should never edit the .spc file directly.)
- ▶ The datapool name that is stored in the repository is the same as the root datapool file name (without the .csv extension). The maximum length of a datapool name is 40 characters.

Example Datapool

The following is an example of a datapool file with three rows of data. In this example, field values are separated by commas:

```
John,Sullivan,238 Tuckerman St,Andover,MA,01810  
Peter,Hahn,512 Lewiston Rd,Malden,MA,02148  
Sally,Sutherland,8 Upper Woodland Highway,Revere,MA,02151
```

Example Using Microsoft Excel

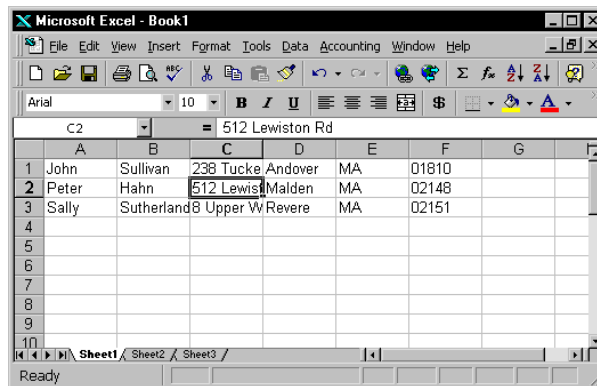
To create and populate a datapool using Microsoft Excel:

1. Run Microsoft Excel.
2. Click **File** → **New** to create a new Excel workbook.
3. Click the **Workbook** icon, and then click **OK**.
4. Type a datapool record into row 1. To do so, type each value in the record into separate columns, beginning with column A.

When using Microsoft Excel to populate a datapool, do not separate values with the Windows separator character (see page 8-39). Excel automatically inserts the separator character when you save the datapool in .csv format.

5. Continue populating the datapool by typing records into the subsequent rows.

The following is an example of how a datapool might look as it is being populated with data in Microsoft Excel:



Note that:

- ▶ Each column represents a datapool field.
- ▶ Each row is an individual datapool record containing datapool field values.

Saving the Datapool in Excel

When you finish adding rows of values to the datapool, save the datapool to .csv format. To do so using Microsoft Excel:

1. Click **File** → **Save As**.
2. In the **Save in** field, specify the directory where you want to save the datapool.

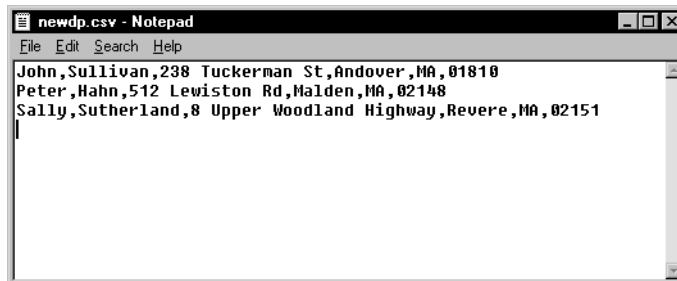
Do not specify the Datapool directory in the repository. When you later import the datapool using the TestManager Import feature, TestManager automatically copies the datapool to the Datapool directory in the current repository and project.

3. In the **File name** field, type the datapool name (maximum 40 characters). This is the name that the script and the repository use to identify the datapool.
4. In the **Save as type** list, select the entry **CSV (Comma delimited) (*.csv)**.
5. Click **Save**.

If you are prompted that the .csv file format does not support multiple workbook sheets, click **OK** to save the datapool and abandon the other (empty) worksheets.

Upon closing Excel, if you are prompted to decide whether to save the .csv file in Microsoft Excel format, click **No**.

If you use Windows Notepad to open the datapool file you just created and saved, this is how it looks:



Matching Datapool Columns with Script Variables

When you create a .csv file and then import it as a datapool, TestManager automatically assigns column names (that is, datapool field names) to each datapool column.

Datapool column names must match the names of the script variables that they supply with data (including a case match). But most likely, when you create and import a datapool, the column names that TestManager assigns will not match the names of the associated script variables. As a result, you need to edit the column names that TestManager automatically assigns during the import. You do so by modifying a column's **Name** value in the Datapool Specification dialog box.

For information about how to open the Datapool Specification dialog box during datapool editing, see *Editing Datapool Column Definitions with TestManager* on page 8-25.

NOTE: With GUI scripts, you can associate datapool column names and script variables through column position rather than column name. For more information, see the description of the `SQADatapoolValue` command in the *SQABasic Language Reference*.

Maximum Number of Imported Columns

You can import a datapool that contains up to 32,768 columns. If you open an imported datapool in the Datapool Specification dialog box, you can view and edit all datapool column definitions up to that limit.

A datapool is subject to a 150-column limit only if you generate data for the datapool from the Datapool Specification dialog box.

Creating a Column of Values Outside Rational Test

A datapool that you create with Rational Test can include a column of values supplied by an ASCII text file. You could use this feature, for example, if you wanted the datapool to include a column of values from a database.

Populating a datapool column with values from an external file requires two basic steps:

1. Create the file containing the values.
2. Assign the values in the file to a datapool column through the standard data type Read From File.

Step 1. Create the File

To use a file as a source of values for a datapool column, the file must be a standard ASCII text file. The file must contain a single column of values, with each value terminated by a carriage return.

You can create this text file any way you like — for example, you can use either of these methods:

- ▶ Type the list of values in Microsoft Notepad.
- ▶ Export a column of values from a database to a text file.

Step 2. Assign the File's Values to the Datapool Column

Once the file of values exists, you assign the values to a datapool column just as you assign any set of values to a datapool column — through a data type. In this case, you assign the values through the Read From File data type. To do so:

1. Open the Datapool Specification dialog box:
 - To open this dialog box during datapool creation, see *Creating a Datapool with TestManager* on page 8-18.
 - To open this dialog box during datapool editing, see *Editing Datapool Column Definitions with TestManager* on page 8-25.
2. In the **Type** column, select the data type Read From File for the datapool column being supplied the values from the external text file.
3. Tab out of the column. The Open dialog box appears.
4. In **Look in**, specify the directory where the text file that you created is located.
5. In **File name**, specify the name of the text file.
6. Click **Open**.

You can use the Read From File data type to assign values to multiple columns in the same datapool.

Generating Unique Values

You can use the Read From File data type to generate unique values to a datapool column that you create outside Rational Test.

To generate unique values through the Read From File data type, the file that the data type accesses must contain unique values.

In addition, when you are defining the datapool in the Datapool Specification dialog box, change the following settings for the datapool column associated with the Read From File data type:

- ▶ Set **Sequence** to Sequential.
- ▶ Set **Repeat** to 1.
- ▶ Make sure the **No. of records to generate** value does not exceed the number of unique values that you are accessing through the Read From File data type.

For information about the values that you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 8-20.

▶ ▶ ▶ Part IV

Playing Back Scripts and Analyzing Results

Playing Back GUI Scripts

This chapter explains how to play back GUI scripts. It includes the following topics:

- ▶ Playback phases
- ▶ Restoring the test environment before playback
- ▶ Setting GUI playback options
- ▶ Playing back a GUI script
- ▶ Viewing results in the Rational LogViewer
- ▶ Analyzing verification point results with the Comparators

NOTE: For information about playing back virtual user scripts, see the *Using Rational LoadTest* manual.

Playback Phases

When you play back a script, Rational Robot repeats your recorded actions and automates the software testing process. With automation, you can test each new build of your application faster and more thoroughly than by testing it manually. This decreases testing time and increases both coverage and overall consistency.

There are two general phases of script playback:

- ▶ Test development phase
- ▶ Regression testing phase

These phases are described in the following sections.

Test Development Phase

During the test development phase, you play back scripts to verify that they work as intended, using the same version of the application-under-test that you used to record. This validates the baseline of expected behavior for the application.

The following table shows the general process for the test development phase.

Task	See
1. Prepare for playback by restoring the test environment and setting the playback options.	<i>Restoring the Test Environment Before Playback</i> on page 9-3 <i>Setting GUI Playback Options</i> on page 9-4
2. Play back each script against the same version of the application-under-test that was used for recording to verify that it performs as intended.	<i>Playing Back a GUI Script</i> on page 9-18
3. Analyze the results using the LogViewer.	Chapter 10, <i>Reviewing Logs with the LogViewer</i>
4. Use the appropriate Comparator to determine the cause of verification point failures.	Chapters 11, 12, 13, 14
5. If the script fails, edit, debug, or rerecord the script so that it runs as required.	Chapter 7, <i>Editing, Compiling, and Debugging Scripts</i>
6. Group individual scripts into a comprehensive shell script. Play back the shell script to verify that the scripts work properly. If necessary, edit, debug, or rerecord the scripts.	<i>Creating Shell Scripts to Play Back Scripts in Sequence</i> on page 4-26

Regression Testing Phase

During the regression testing phase, you play back scripts to compare the latest build of the application-under-test to the baseline established during the test development phase. Regression testing reveals any differences that may have been introduced into the application since the last build. You can evaluate these differences to determine whether they are actual defects or deliberate changes.

The following table shows the general process for the regression testing phase.

Task	See
1. Prepare for playback by restoring the test environment and setting the playback options.	<i>Restoring the Test Environment Before Playback</i> on page 9-3 <i>Setting GUI Playback Options</i> on page 9-4
2. Play back each script against a new build of the application-under-test.	<i>Playing Back a GUI Script</i> on page 9-18
3. Analyze the results using the LogViewer.	Chapter 10, <i>Reviewing Logs with the LogViewer</i>
4. Use the appropriate Comparator to determine the cause of verification point failures. If failed verification points are the result of intentional changes to the application-under-test, update the baseline data using the appropriate Comparator.	Chapters 11, 12, 13, 14
5. Use the LogViewer to enter defects.	<i>Entering and Modifying Defects</i> on page 10-14
6. If necessary, revise the scripts to bring them up-to-date with new features in the application-under-test. Play back the revised scripts against the current build and then reevaluate the results.	Chapter 7, <i>Editing, Compiling, and Debugging Scripts</i>

Restoring the Test Environment Before Playback

The state of the Windows environment as well as your application-under-test can affect script playback. If there are differences between the recorded environment and the playback environment, playback problems can occur.

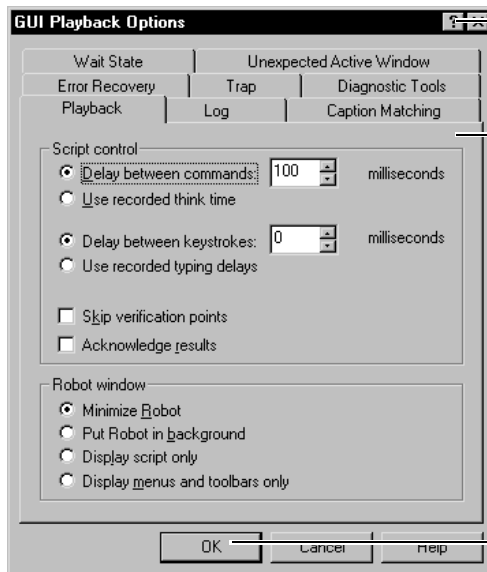
Before playing back a script, be sure that your application-under-test is in the same state it was in when you recorded the script. Any applications and windows that were open, active, or displayed when you started recording the script should be open, active, or displayed when you start playback. In addition, be sure that any relevant network settings, active databases, and system memory are in the same state as when the script was recorded.

Setting GUI Playback Options

GUI playback options provide instructions to Robot about how to play back scripts. You can set these options either before you begin playback or early in the playback process.

To set GUI playback options:

- ▶ Open the GUI Playback Options dialog box by doing one of the following:
 - Before you start playback, click **Tools** → **GUI Playback Options**.
 - Start playback by clicking the **Playback Script** button on the toolbar. In the Playback dialog box, click **Options**.



For detailed information about an item, click the question mark, and then click the item.

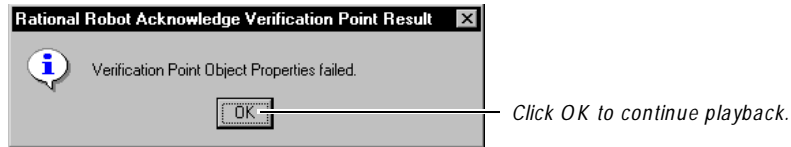
Set the options on each tab.

Click OK or change other options.

Acknowledging the Results of Verification Point Playback

By selecting the **Acknowledge results** check box, you can have Robot display a results message box each time it plays back a verification point.

For example, in the following figure, the message box indicates that the verification point named Object Properties failed during playback. You must click **OK** before playback continues. During the test development phase, this lets you interactively view the playback results of each verification point.



During the regression testing phase, you usually play back scripts in unattended mode. By clearing the **Acknowledge results** check box, you can prevent Robot from displaying this message box. After the script plays back, you can view the results of all verifications points in the LogViewer. (For more information, see Chapter 10, *Reviewing Logs with the LogViewer*.)

To set this option:

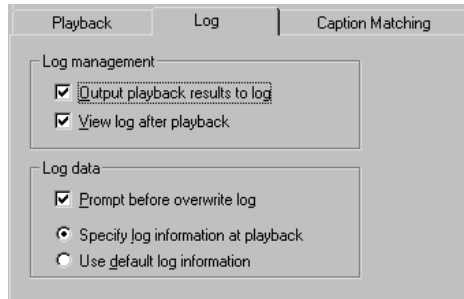
1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. In the **Playback** tab, do one of the following:
 - Select **Acknowledge results** to have Robot display a pass/fail result message box for each verification point. You must click **OK** before playback continues.
 - Clear **Acknowledge results** so that Robot does not interactively display pass/fail results.
3. Click **OK** or change other options.

Setting Log Options for Playback

A **log** is a file that contains the record of events that occur while a script is playing back. A log includes the results of the script and of all verification points. You view logs in the LogViewer. (For more information, see Chapter 10, *Reviewing Logs with the LogViewer*.)

To set the log options:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Log** tab.



3. To output the playback results to the log so you can view them in the LogViewer, select **Output playback results to log**.

If you clear this, you cannot to view the playback results in the log.

4. To have the LogViewer appear automatically after playback is complete, select **View log after playback**.

If you clear this, you can still view the log after playback by clicking **Tools** → **Rational Test** → **LogViewer**, and then opening the log.

5. To have Robot prompt you before it overwrites a log, select **Prompt before overwrite log**.

6. Click one of the following:

Specify log information at playback – At playback, displays the Specify Log Information dialog box so that you can specify the build, log folder, and log. You can define these in TestManager, Robot, or TestFactory.

Use default log information – At playback, uses the same build and log folder that was used during the last playback. Uses the script name as the log name.

For information about builds, log folders, and logs, see Chapter 3, *Managing Builds, Log Folders, and Logs*.

7. Click **OK** or change other options.

Setting Wait State and Delay Options

In most cases, it is important for the playback of a GUI script to be synchronized with the application-under-test, so that Robot executes commands in the script only after the application is ready to receive them. Robot attempts to maintain this synchronization automatically for you using several techniques.

You can refine the synchronization by setting the following options in the GUI Playback Options dialog box:

- ▶ Wait states for windows
- ▶ Delays between commands
- ▶ Delays between keystrokes

These options are described in the following sections.

NOTE: If a script needs to wait before executing a particular command, you can insert a delay for just that command. (For information, see *Inserting Delay Values* on page 5-10.) If you are testing an application in which time estimates are not predictable, you can define a wait state for a verification point so that playback waits based on specific conditions rather than on absolute time. (For information, see *Setting a Wait State for a Verification Point* on page 6-8.)

Setting Wait State Options

During playback, Robot waits for windows (including dialog boxes) to appear before executing a user action or verification point command. You can specify how often Robot checks for the existence of a window and how long it waits before it times out.

For example, suppose that Robot is playing back a script with the following lines:

```
StartApplication "MyVBApp.exe"
Window SetContext, "Name=Form1", ""
Pushbutton Click, "Name=Command5"
```

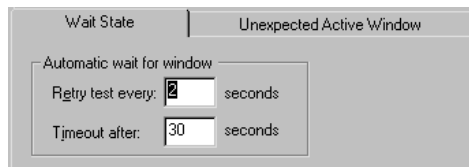
This example specifies that Robot should start an application, find a window on the desktop named “Form1”, find a pushbutton named “Command5”, and generate a click on that button. However, suppose Robot gets to the `SetContext` line in the script and fails to find a window named “Form1”. This may not necessarily be an error — the application may not yet have started up and created the window. In this case, Robot keeps looking for the window for a specified period of time.

By default, if Robot cannot find a window during playback, it waits for 2 seconds and then looks for it again. If it still cannot find the window after 30 seconds, it times out and returns a command failure indication to the script. Script execution continues or stops based on the **On script command failure** setting in the **Error Recovery** tab of the GUI Playback Options dialog box.

You can change the default values for the retry time and the timeout by changing the wait state options.

To set the wait state options:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Wait State** tab.



3. To specify how often Robot checks for the existence of a window, type a number in the **Retry test every** box.
4. To specify how long Robot waits for a window before it times out, type a number in the **Timeout after** box.
5. Click **OK** or change other options.

NOTE: This synchronization is used only in Object-Oriented Recording. In contrast, low-level scripts are processed in real time. They play back at the same speed at which they were recorded and do not use automatic wait settings.

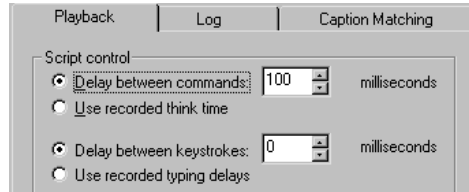
Setting Delay Options for Commands and Keystrokes

By default, Robot pauses 100 milliseconds between each user action command and between each verification point command during playback. If you find that Robot consistently gets ahead of your application-under-test during playback, you can increase the time that Robot waits between these commands.

Also, if you find that your application-under-test does not see all of the keystrokes that Robot sends it, you can have Robot wait between sending keystrokes to the application.

To set the delay options for commands and keystrokes:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Playback** tab.



3. Click **Delay between commands**. Type the delay value.

This is the delay between each user action command and between each verification point command during playback.

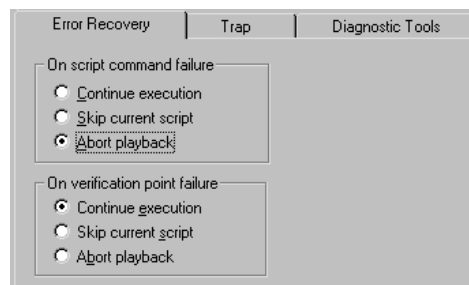
4. Click **Delay between keystrokes**. Type the delay value.
5. Click **OK** or change other options.

Setting Error Recovery Options

Use the error recovery options to specify how Robot handles script command failures and verification point failures.

To set the error recovery options:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Error Recovery** tab.



3. To specify what Robot should do if it encounters a failure, click one of the following options under both **On script command failure** and **On verification point failure**:

Continue execution – Continues playback of the script.

Skip current script – Terminates playback of the current script. If the script with the failure was called from another script, playback resumes with the command following the `CallScript` command.

Abort playback – Terminates playback of the current script. If the script with the failure was called from another script, the calling script also terminates.

4. Click **OK** or change other options.

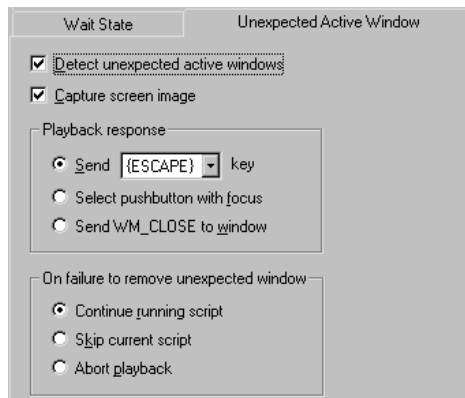
Failures are stored in the log. (For information about logs, see Chapter 10, *Reviewing Logs with the LogViewer*.)

Setting Unexpected Active Window Options

An **unexpected active window** is any unplanned window that appears during script playback that prevents the expected window from being made active (for example, an error message from the network or application-under-test). These windows can interrupt playback and cause false failures.

To set options to specify how Robot responds to unexpected active windows:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Unexpected Active Window** tab.



3. To have Robot detect unexpected active windows and capture the screen image for viewing in the Image Comparator, select **Detect unexpected active windows** and **Capture screen image**. (For information, see *Viewing Unexpected Active Window* on page 14-18.)
4. Specify how Robot should respond to an unexpected active window:
 - Send key** – Robot sends the specified keystroke: ENTER, ESCAPE, or any alphabetic key (A through Z).
 - Select pushbutton with focus** – Robot clicks the push button with focus.
 - Send WM_CLOSE to window** – Robot sends a Windows WM_CLOSE message. This is equivalent to clicking the Windows Close button.
5. Specify what Robot should do if it cannot remove an unexpected active window:
 - Continue running script** – Robot continues script playback with the next command in the script after the one being processed when the unexpected active window appeared. Playback continues even if the unexpected active window cannot be removed. This may result in repeated script command failures.
 - Skip current script** – Robot halts playback of the current script. If the script that detected the unexpected active window was called from within another script, playback resumes with the script command following the `CallScript` command.
 - Abort playback** – Robot halts playback completely. If the script that detected the unexpected active window was called from within another script, the calling script also stops running.
6. Click **OK** or change other options.

Setting Diagnostic Tools Options

You can use the Rational diagnostic tools — Rational Purify, Quantify, and PureCoverage — to collect diagnostic information during playback of a Robot script.

After playback, Robot can integrate the diagnostic tool's results into the Robot log, so that you can view all of the playback results in one place. You can choose to show any combination of errors, warnings, and informational messages. You can then double-click a result in the log to open the script in Robot and the appropriate file in the diagnostic tool.

About Purify and Robot

Robot with playback under Purify works with Visual C/C++ applications on Windows NT 4.0 and Windows 2000.

Purify detects and diagnoses memory access errors and memory leaks. Without Purify, the visible symptoms (crashes, malfunctions, or incorrect results) of these kinds of errors often do not show up until long after the erroneous code was executed, and in a short test, often not at all. Purify detects and pinpoints the cause of the error as the code is executed.

Where applicable, Purify adds significant value to Robot, because it finds many otherwise hidden defects in the application code.

About Quantify and Robot

Robot with playback under Quantify works with Visual C/C++ , Visual Basic, and Java applications on Windows NT 4.0 and Windows 2000.

Quantify profiles the time spent in each module, function, line, and block of code, and detects performance bottlenecks within an application. Once bottlenecks are identified, you can focus on the inefficient parts of the code, and substitute alternative implementations or algorithms to improve performance.

By using a Robot script to drive the application, in conjunction with Quantify, you ensure that a repeatable test is measured for each iteration of performance improvement. This minimizes the risk of comparing different things when contrasting a run before and after a possible performance-enhancing code-change.

About PureCoverage and Robot

Robot with playback under PureCoverage works with Visual C/C++ , Visual Basic, and Java applications on Windows NT 4.0 and Windows 2000.

PureCoverage is a code coverage analyzer that reports which modules, functions, and lines of code were and were not executed in any run or collection of runs. Using PureCoverage to monitor a script reveals how comprehensively that script exercises the application-under-test, and can provide helpful information about which code paths are taken under particular scenarios.

How the Diagnostic Tools Work with Robot

For Visual C/C++ and Visual Basic applications, these diagnostic tools work best when the applications have been compiled with debug information (in other words, when a .pdb file is available). These tools use the Rational Object Code Insertion (OCI) technology to insert instrumentation probes into the executable program for the application before it runs. When you select a diagnostic tool in the GUI Playback Options dialog box, you instruct Robot to call that tool to instrument the application that is to be started, and then run the instrumented application in place of the original.

For Java applications, Quantify and PureCoverage put the JVM into a special mode to enable event monitoring when the application runs during playback. When you select a diagnostic tool in the GUI Playback Options dialog box, you instruct Robot to call that tool to enable event monitoring.

To use any of the diagnostic tools with Robot, start the application as follows when recording:

- ▶ Start Visual C/C++ and Visual Basic applications with the **Start Application** button or command.
- ▶ Start Java applications with the **Start Java Application** button or command.

For more information, see *Starting an Application* on page 5-1.

NOTE: There is no support for using Robot to play back Java applets under these diagnostic tools. For information about the environments that are supported directly in these tools, see the documentation for the appropriate product.

Setting the Options

To set options to specify the diagnostic tool to be used during playback:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)

- Click the **Diagnostic Tools** tab. Then do the following:

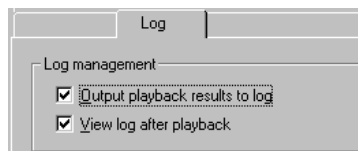
a. Click the diagnostic tool under which the application should run. The options are enabled if you have the tools installed.

b. Optionally, select this and type a value. This multiplies wait state and delay values.

c. Select the type of information to show in the log.

For Visual C/C++ and Visual Basic applications, the instrumentation added by these tools causes the application-under-test to run considerably slower than the uninstrumented program. For Java applications, the event monitoring causes the application to run slower than usual. If you select **Use timeout multiplier**, Robot compensates for this effect by multiplying wait state values for windows, wait state values for verification points, and delay values (between user action and verification point commands, and between keystrokes) by the specified value. If you need to fine-tune the wait states and delays further, see *Setting Wait State and Delay Options* on page 9-7.

- If you selected any of the log check boxes in step 2c, click the **Log** tab and select both **Log management** check boxes.



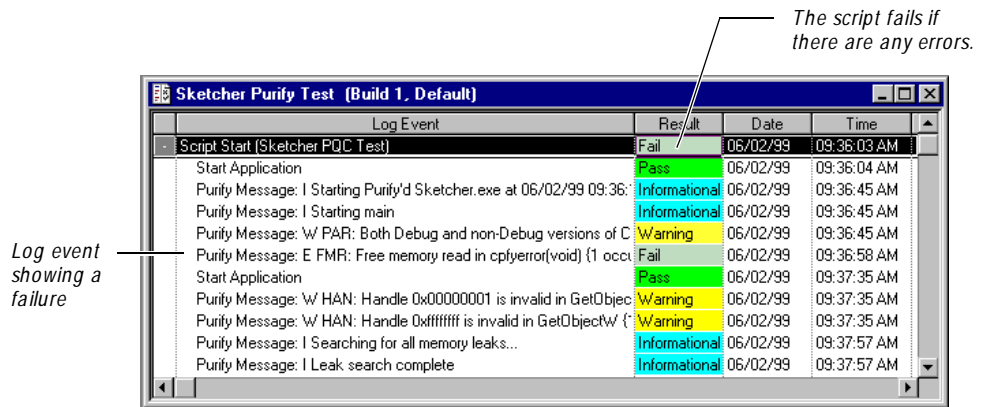
If **Output playback results to log** is cleared, the diagnostic tool opens (instead of the log) after you play back the script. (For more information about the log options, see *Setting Log Options for Playback* on page 9-5.)

- Click **OK** or change other options.

You are now ready to play back the script. Keep in mind that the script will run slower than usual because of the instrumentation or event monitoring added by these tools.

Viewing the Playback Results

After you play back the script, the results appear in the log.



A summary line at the end of the log indicates the total number of errors, warnings, and informational messages.



Double-click a log event to open up both:

- ▶ The script in Robot, near the line that was executing when the error was reported.
- ▶ The appropriate file in the diagnostic tool, so that you can view the details.

If you double-click the summary line, and if there are multiple files involved in the summary, then all of the files open. This happens if the script contains more than one `StartApplication` or `StartJavaApplication` command.

For more information about the log, see Chapter 10, *Reviewing Logs with the LogViewer*.

Setting the Trap Options to Detect GPFs

Robot uses the Trap utility to detect the occurrence of General Protection Faults (GPF) and the location of offending function calls during playback. If a GPF is detected, Robot updates a log file that provides information about the state of the Windows session that was running.

Important Notes

- ▶ To use the Trap utility, you must include Common Object File Format (COFF) information in your application when you link. For instructions, see the documentation for your development environment.
- ▶ Trap, Visual C++ , and Dr. Watson (from Microsoft), WinSpector (from Borland), and Crash Analyzer (from Symantec) use the same Windows system calls to trap faults. You cannot use more than one error trapping program at the same time.

Uses for Trap

The occurrence of a GPF usually results in a crash of the running application and may also result in a loss of data. Using Trap, you can:

- ▶ Capture information about GPFs.
- ▶ Write the state of your environment to a log file when a GPF is detected.
- ▶ Specify the type of information to write to the log file.
- ▶ Automatically restart Windows or call your own error handling sub procedure before performing any other action.
- ▶ Save an audit of the function where the fault occurred in the failing program.

The Trap utility detects and traps the following events during playback:

- ▶ UAE: General Protection Fault # 13
- ▶ Stack Overflow: Fault # 12
- ▶ Invalid Op Code: Fault # 6
- ▶ Divide by Zero: Fault # 0

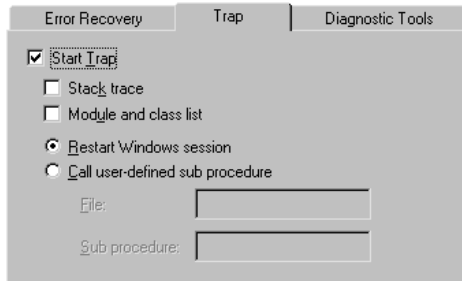
If one of these errors occurs, Trap appends the error data to the existing Sqatrap.log file in the Rational installation directory, or creates a new file if one does not exist. (For more information, see *Analyzing Results in the Sqatrap.log File* on page 9-17.)

Starting Trap

NOTE: Before you start Trap, see *Important Notes* on page 9-16.

To automatically start Trap during playback:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-4.)
2. Click the **Trap** tab.



3. Select **Start Trap** to enable the other options.
4. To include the contents of the stack for non-current tasks, select **Stack trace**.
5. To include the modules and class list information, select **Module and class list**.
6. Click one of the following to specify what Trap should do after detecting a GPF:

Restart Windows session – Trap restarts Windows.

Call user-defined sub procedure – Trap calls the sub-procedure in the module that you specify. Select this option to specify your own custom SQABasic error handling. Type the names of the library source file (with an.sbl extension) and the sub-procedure.

7. Click **OK** or change other options.

Analyzing Results in the Sqatrap.log File

If you select the **Start Trap** option and an error occurs during playback, Robot appends the error data to the existing Sqatrap.log file in the Rational installation directory, or creates a new file if one does not exist. (To start with a clean Sqatrap.log file, delete the old file.) This file provides information about the state of the Windows session that was running.

The Sqatrap.log file can contain a variety of information about failure events. The following failure information is always written to Sqatrap.log:

- ▶ Contents of the stack for the current task
- ▶ Names of functions that were called just before the error occurred
- ▶ Contents of CPU registers
- ▶ Date/Time stamp and Fault Number

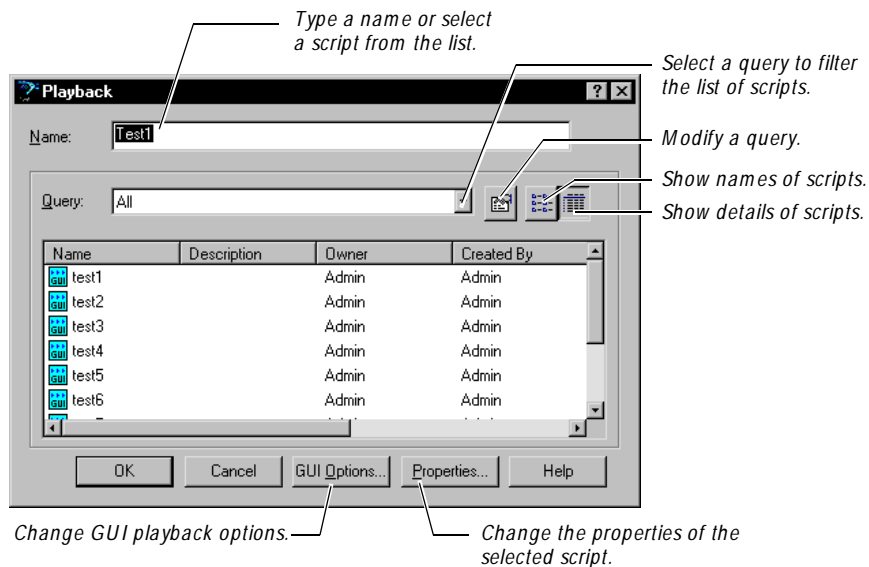
Other information can be written to Sqatrap.log depending on settings in the **Trap** tab of the GUI Playback Options dialog box.

To see a sample Sqatrap.log file, see *Sqatrap.log file* in the Robot Help Index.

Playing Back a GUI Script

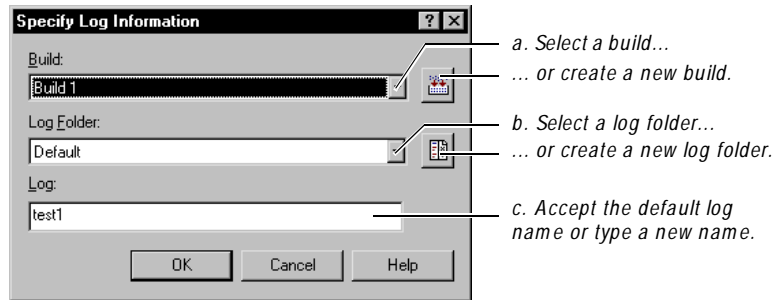
To play back a GUI script:

1. Prepare for playback by restoring the test environment. (For information, see *Restoring the Test Environment Before Playback* on page 9-3.)
2. Set your playback options. You can also set these options after you start playback. (For instructions, see *Setting GUI Playback Options* on page 9-4.)
3. Click the **Playback Script** button on the toolbar.



4. Type a name or select it from the list.
To change the list, select a query from the **Query** list.
5. To change the playback options, click **GUI Options**. When finished, click **OK**.
6. Click **OK** to continue.
7. If the Specify Log Information dialog box appears, fill in the dialog box and click **OK**.

This dialog box appears if you selected **Specify log information at playback** in the **Log** tab of the GUI Playback Options dialog box.



For information about builds, log folders, and logs, see Chapter 3, *Managing Builds, Log Folders, and Logs*.

8. If a prompt appears asking if you want to overwrite the log, do one of the following:
 - Click **Yes** to overwrite the log.
 - Click **No** to return to the Specify Log Information dialog box. Change the build, log folder, and/or log information.
 - Click **Cancel** to cancel the playback.

This prompt appears if you selected **Prompt before overwrite log** in the **Log** tab of the GUI Playback Options dialog box.

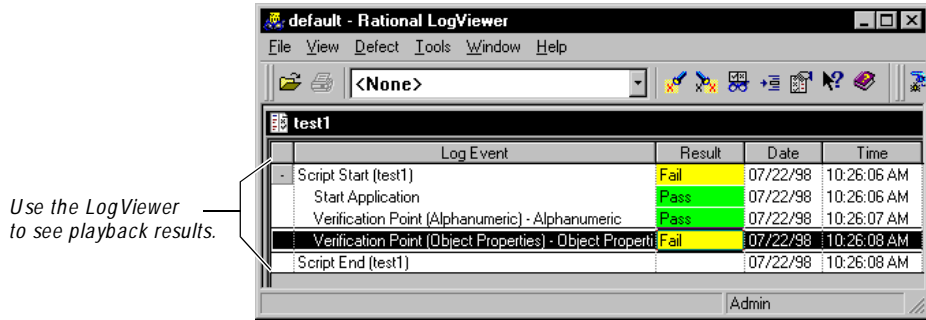
When you begin playback, the Robot main window is minimized by default. You can change this behavior in the **Playback** tab of the GUI Playback Options dialog box.

NOTE: To stop playback of a script, press the F11 key. Robot recognizes the F11 key only when playing back object-oriented commands. The F11 key does not stop playback during low-level actions.

After playback, you can see the results in the LogViewer, as described in the next section.

Viewing Results in the Rational LogViewer

After playback finishes, you can use the LogViewer to view the playback results, including verification point failures, procedural failures, aborts, and any additional playback information.



The following table gives you more information about the LogViewer.

To	Do this	For information, see
Control the log information and display of the LogViewer	Set options in the Log tab of the GUI Playback Options dialog box.	<i>Setting Log Options for Playback</i> on page 9-5
Play back a script under Purify, Quantify, or PureCoverage, and see the results in the log	Set options in the Diagnostic Tools tab of the GUI Playback Options dialog box.	<i>Setting Log Options for Playback</i> on page 9-5
Analyze a failure in a Comparator	Double-click a verification point failure in the LogViewer.	The next section, <i>Analyzing Verification Point Results with the Comparators</i>
Enter defects into Rational ClearQuest from the LogViewer	Select the failed event in the log and click Defect → Generate .	<i>Entering and Modifying Defects</i> on page 10-14

For detailed information about the LogViewer, see Chapter 10, *Reviewing Logs with the LogViewer*.

Analyzing Verification Point Results with the Comparators

Use the Comparators to analyze differences between the baseline verification point data (the data captured when you created the verification point) and the actual verification point data (the data captured when you played back the verification point). The Comparators help you determine whether a failure is a defect or an intentional change to the application-under-test.

There are four Comparators, as follows:

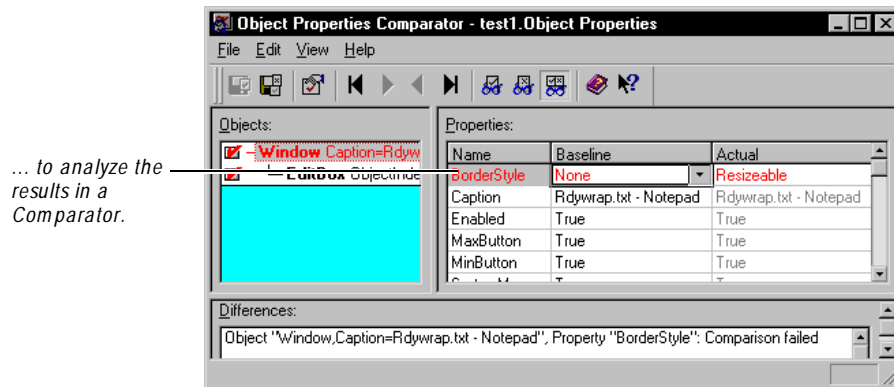
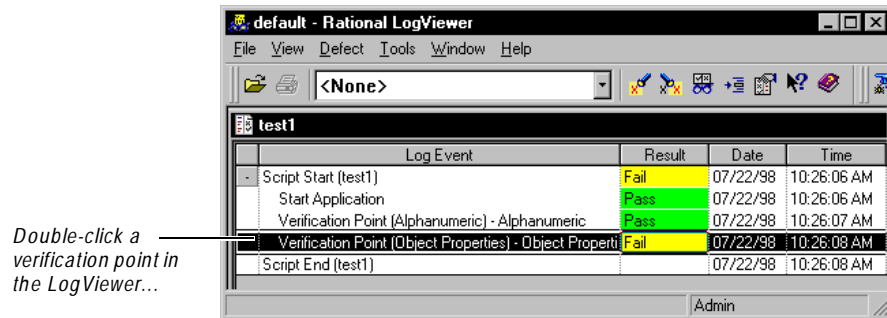
Comparator	Description	For information see
Object Properties	Compares the baseline data to the data that caused a failure for the Object Properties verification point.	Chapter 11, <i>Using the Object Properties Comparator</i>
Text	Compares the baseline data to the data that caused a failure for the Alphanumeric verification point.	Chapter 12, <i>Using the Text Comparator</i>
Grid	Compares the baseline data to the data that caused a failure for the following verification points: Clipboard, Menu, and Object Data.	Chapter 13, <i>Using the Grid Comparator</i>
Image	Compares the baseline image to the image that caused a failure for the Window Image or Region Image verification points. Also lets you view unexpected active windows that cause a failure during playback.	Chapter 14, <i>Using the Image Comparator</i>

To open a Comparator from the LogViewer:

- ▶ In the **Log Event** column of a log, double-click a verification point.

Playing Back GUI Scripts

As the following figure shows, double-clicking an Object Properties verification point in the LogViewer opens the Object Properties Comparator.



NOTE: You can also open a Comparator from Robot by double-clicking a verification point in the Asset (left) pane of a Script window. However, when you open a Comparator this way, you can view only the baseline file. To compare the baseline and actual files, you must open the Comparator through the LogViewer.

▶▶▶ C H A P T E R 10

Reviewing Logs with the LogViewer

This chapter introduces the Rational LogViewer, and explains how to use the LogViewer to view logs and interpret their contents. This chapter includes the following topics:

- ▶ Overview
- ▶ Starting the LogViewer
- ▶ The LogViewer main window
- ▶ Opening a log file
- ▶ Deleting a log file
- ▶ Viewing log event properties
- ▶ Modifying the log window
- ▶ Locating failed log events
- ▶ Evaluating verification point failures in a Comparator
- ▶ Filtering the log event column
- ▶ Working with reports
- ▶ Entering and modifying defects

Overview

Use the Rational LogViewer to view the logs created after you run scripts or schedules.

A typical test process involves using Rational Robot to record scripts that contain verification points. After the script is played back, Robot writes the results to a log. Certain verification points also have **baseline data files** that are saved. If the verification point fails during playback, **actual data files** are also saved. You can then compare the actual to the baseline file to examine the failures.

In addition to using the LogViewer to view the playback results of verification points, you can use it to view procedural failures, aborts, and any additional playback information. You then use the appropriate Comparators to view actual data or image files, and view and edit the baseline files as needed.

A testing cycle can have many individual tests for specific areas of an application. Reviewing the results of tests in the LogViewer reveals whether each passed or failed. Analyzing the results in a Comparator helps determine why a test may have failed. Review and analysis help determine where you are in your software development effort and whether a failure is a defect or a design change.

Usage Scenarios

The following are the main usage scenarios for the LogViewer:

- ▶ After a script or schedule runs, the LogViewer opens automatically and displays the newly generated log.
- ▶ Open the LogViewer independently, and then open any log that you want to view.
- ▶ From an open log, double-click a log event generated from a verification point. The appropriate Comparator opens so you can view the results of the verification point. The Comparator shows the original results (the baseline) and the current playback (the actual) results side-by-side so you can compare them and analyze their differences.
- ▶ From an open log, double-click a log event generated from playing back the script under Purify, Quantify, or PureCoverage. The script opens in Robot, and the file opens in the diagnostic tool. (For more information, see *Setting Diagnostic Tools Options* on page 9-11.)
- ▶ From an open log, select any script-based log event, and click **View** → **Script** to open the script in Robot (for GUI scripts) or TestManager (for manual scripts).

- ▶ From an open log, generate a report of the log events using the **File** → **Quick Report** command.
- ▶ From an open log, generate a defect from a failed log event using the **Defect** → **Generate** command. The LogViewer automatically fills in some of the information in the defect form, which is part of Rational ClearQuest.

Starting the LogViewer

You can start the LogViewer in the following ways:

- ▶ Automatically from Robot
- ▶ Automatically from LoadTest
- ▶ From TestManager
- ▶ From any Rational Test product
- ▶ From the Windows desktop

NOTE: You can also start the LogViewer from a selected script in a Rational TestFactory application map. For more information, see the *Using Rational TestFactory* manual.

Starting the LogViewer Automatically from Robot

To set up Robot so it automatically opens the LogViewer after playback:

1. In Robot, click **Tools** → **GUI Playback Options**.
2. Click the **Log** tab.
3. Select **Output Playback Results to Log** and **View Log After Playback**.
4. Click **OK**.
5. Play back a script in Robot.

When the script has finished playing back, Robot starts the LogViewer and opens the log.

Starting the LogViewer Automatically from LoadTest

To set up LoadTest so it automatically opens the LogViewer after running a schedule:

1. In LoadTest, click **Tools** → **Options**.
2. Click the **Reports** tab.

Reviewing Logs with the LogViewer

3. Under **Automatic LogViewer**, click **Always**.
4. Click **OK**.
5. Run a schedule.

When the schedule has finished running, LoadTest starts the LogViewer and opens the log file.

Starting the LogViewer from TestManager

To open a log from TestManager:

1. Click **View** → **Asset Browser**.
2. In the Asset Browser, expand the Builds tree until the log name appears.
3. Double-click the log to open it.

Starting the LogViewer from a Rational Test Product

To start the LogViewer from a Rational Test product:

- ▶ Click **Tools** → **Rational Test** → **Rational LogViewer** in any Rational Test product, or click the Rational LogViewer toolbar button.

You can then open a log file by clicking **File** → **Open**.

NOTE: You can only open log files that are part of the current project. To change your project, click **File** → **Change Project**.

Starting the LogViewer from the Desktop

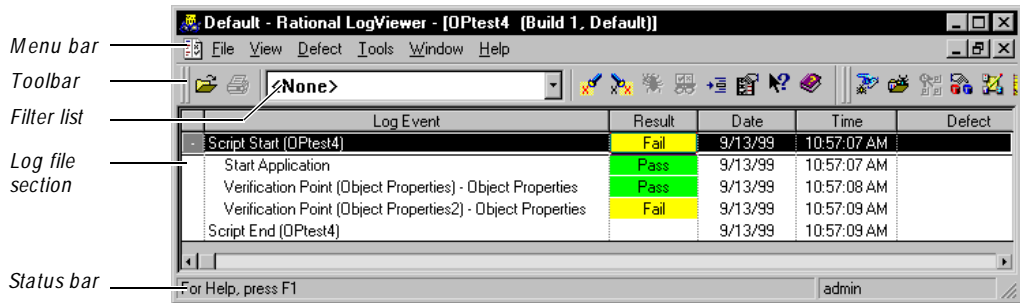
To start the LogViewer from the Windows desktop:

1. Click **Start** → **Programs** → *Rational product name* → **Rational Test** → **Rational LogViewer**.
2. Type your user ID and password. If you do not know these, see your administrator.
3. Select a repository and project name.
4. Click **OK**.

You can then open a log file by clicking **File** → **Open**.

The LogViewer Main Window

The main window of the LogViewer contains the log file section, as well as the toolbar, the menus, and the status bar.



The log file section displays the log events that are generated when scripts or schedules are run. By default, the following columns appear:

- ▶ **Log Event** – Lists all log events, such as the script start and end, verification points, manual steps, and unexpected active windows. Right-click to display a shortcut menu. To define filters to narrow down the event types that appear, click **Tools** → **Manage Filters**. You can apply a filter to the currently active log using the Filter list on the toolbar.
- ▶ **Result** – Indicates the results of the events. If a verification point fails, double-click the failure to open the appropriate Comparator. The Comparator shows the baseline and actual files so you can evaluate any failures found and determine whether they are intentional changes or defects. To define filters to narrow down the result types that appear, click **Tools** → **Manage Filters**.
- ▶ **Date** – Lists the date of the log event.
- ▶ **Time** – Lists the start time of the log event.
- ▶ **Defect** – Lists the defect number associated with this log event (if there is one).
- ▶ **Computer** – Lists the computer on which the script was run.

You can customize the order of these columns by clicking **View** → **Column Order**.

You can filter the currently active log by selecting a filter from the list on the toolbar. For information about log filters, see *Filtering the Log Event Column* on page 10-11.

Opening a Log File

To open a log file:

1. In the LogViewer, click **File** → **Open**.
2. Select a log file to open.

You can only open log files that are part of the current project. To change your project, click **File** → **Change Project**.

3. Optionally, select **Display Builds by State** to show the hierarchy of builds and logs by states.

You can create build states using TestManager. Four build states are already available to you, and you can also create custom build states. If you do not change the build state, the default of < **None**> is assigned to each build. For information about build states, see *Working with Build States* on page 3-9.

4. Click **OK** to open the log.

You can open more than one log file in the LogViewer. If you have more than one log file open, the log that is currently active is the one that is acted upon when you use most menu commands.

Deleting a Log File

You can delete log files from your project. Once you delete a log file, you cannot get the log back.

To delete a log file:

1. Click **File** → **Delete**.
2. Select the log or log folder to delete.

When you delete a log folder, all logs in the folder are deleted.

3. Optionally, select **Display Builds by State** to show the hierarchy of builds and logs by states.

You can create build states using TestManager. Four build states are already available to you, and you can also create custom build states. If you do not change the build state, the default of < **None**> is assigned to each build. For information about build states, see *Working with Build States* on page 3-9.

4. Click **OK**, and click **Yes** to confirm the deletion.

Viewing Log Event Properties

Every log event in the log file has properties, such as the date and time the event was recorded, the type of event, the name of the script, and the configuration of the computer that the script was recorded on.

To view the log event properties:

1. Select a log event in the active log.
2. Click **View** → **Log Event Properties**.

The properties of the log event appear in the **General** tab. You cannot edit these properties.

Modifying the Log Window

You can modify the log window by:

- ▶ Collapsing and expanding log events
- ▶ Changing column widths
- ▶ Changing the column order

Collapsing and Expanding Log Events

The playback of a script results in a log of events. These events are stored hierarchically in the **Log Event** column of the LogViewer.

To expand or collapse the view of log events in a log, do one of the following:

- ▶ To expand or collapse a single log event, click the plus or minus sign at the beginning of the event, or select the event and click **View** → **Expand Log Event** or **Collapse Log Event**.
- ▶ To expand or collapse all log events, click **View** → **Expand All Log Events** or **Collapse All Log Events**.

Changing Column Widths

To change the column width settings in the log file section of a log, position the pointer on the vertical border between the column title cells and drag the border. To hide a column completely, drag its right border to the left until the column disappears.

To keep any new column width settings that you make during a LogViewer session, click **View** → **Save Column Widths**. To restore the columns to their default widths, click **View** → **Restore Column Widths**.

Changing the Column Order

To change the order of the columns in the log file section of a log:

1. Click **View** → **Column Order**.

The **Column Display Order** box lists the columns from top to bottom, in the order in which they appear in the LogViewer from left to right.

2. Select the name of the column to move.
3. Click **Move Up** or **Move Down** to move the column in that direction.
4. Click **OK**.

To restore the columns to their default order, click the **Default Order** button.

Locating Failed Log Events

In the log file section of the LogViewer window, failed events are indicated in red in the **Result** column.

To navigate to failed events in a log:

1. To locate the first failed log event, click **View** → **First Failure**.
2. To locate the next failed event, click **View** → **Next Failure**. Continue to use this command to navigate through all of the failures.

These navigation commands are especially convenient when a log file is large and contains hundreds of log events.

After you locate a verification point failure, you can open a Comparator to analyze the failure by double-clicking the event. In addition, if the event is associated with a script, click **View** → **Script** to see the line of script that contains the failure in Robot.

Evaluating Verification Point Failures in a Comparator

When the application-under-test does not perform on playback exactly as specified by the recording, Robot generates a failed entry in the LogViewer.

Failure indications in log files do not necessarily mean that the application-under-test has failed. You need to evaluate each verification point failure with the appropriate Comparator to determine whether it is an actual defect, a playback environment difference, or an intentional design change made to a new build of the application-under-test.

Viewing a Verification Point in the Comparators

In the log file section of the LogViewer window, failed events are indicated in red in the **Result** column. If the event is a failed verification point, you can analyze the failure using one of the Comparators.

To view a verification point in a Comparator:

1. Open a log file.
2. Do one of the following:
 - Double-click a verification point in the **Log Event** column.
 - Select a verification point and click **View** → **Verification Point**.
 - Right-click a verification point and click **View Result**.

The appropriate Comparator opens based on the type of verification point, as shown in the following table. You can then analyze the results to determine whether the failure was caused by a defect or an intentional change in the application.

Comparator	Verification points
Text Comparator	Alphanumeric
Grid Comparator	Object Data Menu Clipboard
Image Comparator	Window Image Region Image
Object Properties Comparator	Object Properties

Viewing a Script

You can select any log event that is associated with a script and view it in Robot (for GUI scripts) or TestManager (for manual scripts).

To view a script:

1. Open a log file.
2. Select a log event that is associated with a script.
3. Click **View** → **Script**, or right-click the event and click **View Script**.

Playback/Environmental Differences

Differences between the recording environment and the playback environment can generate failure indications that do not represent an actual defect in the software. This can happen if there are applications or open windows in the recorded Windows environment that are not in the playback Windows environment, or vice versa.

For example, if you have the Calculator open in the recorded environment but not open in the playback environment, Robot can generate a failure that has nothing to do with the software that you are actually testing.

You should analyze these failure indications with the appropriate Comparator to determine whether the window that Robot could not find is an application window that should have opened during the script playback or an unrelated window.

Intentional Changes to an Application Build

Revisions to the application-under-test can generate failure indications in scripts and verification points developed using a previous build as the baseline. This is especially true if the user interface has changed.

For example, the Window Image verification point compares a pixel-for-pixel bitmap from the recorded baseline image file to the current version of the application-under-test. If the user interface changes, the Window Image verification point will fail. When intentional application changes result in failures, you can easily update the baseline file to correspond to the new interface using the Image Comparator. Intentional changes in other areas can also be updated using the other Comparators.

For information about updating the baseline, see the Comparator chapters.

Filtering the Log Event Column

You can use filters to narrow down the log event list in the log file section of the LogViewer window. Filters can make it easier to view large log files.

You can:

- ▶ Apply a filter to the log.
- ▶ Create or edit log filters.
- ▶ Copy, rename, or delete a log filter.

These tasks are described in the following sections.

Applying a Log Filter

Use the following steps to apply already existing filters to the log. If you do not have any filters set up, use the **Tools** → **Manage Filters** command to create them. For information, see the following section, *Creating or Editing a Log Filter*.

To apply a filter:

1. Open a log file.
2. Do one of the following:
 - Select a filter from the Filter drop-down list on the toolbar.
 - Click **View** → **Filters**. Select the filter and click **OK**.

That filter is applied to the log.

Creating or Editing a Log Filter

You can use filters to narrow down the log event list in the log file section of the LogViewer window. Filters can make it easier to view large log files.

To create or edit a log filter:

1. Open a log file.
2. Click **Tools** → **Manage Filters**.
3. To create a new filter, click **New**.

To edit a filter, select a filter in the list, and then click **Edit**.

4. In the **General** tab of the Log Filter Properties dialog box, type the name of the filter in the **Name** box. This field is required.
5. Optionally, type or edit a description of the filter in the **Description** box.
6. Specify which results should appear in the log in the **Show Results** option:
 - All** – Displays all results in the log. This is the default setting.
 - Fail and Warning** – Displays only failures and warnings in the log.
7. The **Show Events** list shows the different types of events that can be shown in the log. Select the types that you want shown, and clear the types that you want to filter.
8. If you selected the **Verification Point** check box in the General tab, click the **Verification Points** tab. The **Selected** list contains the verification point types that will be shown in the log. Move all verification points that you want to see to the **Selected** list, and move all the ones you want filtered to the **Available** list.
9. Click **OK** in the Log Filter Properties dialog box.

The filter appears in the Manage Log Filter dialog box.
10. Click **Close**.

NOTE: Use this procedure to create and manage filters. If you just want to apply a log filter to the current log, click **View** → **Filters** or use the Filters list on the toolbar.

Copying, Renaming, and Deleting a Log Filter

The copy feature is useful when you want to create multiple filters that are similar to one another. After you create the first filter, you can make a copy of it. Then you can edit the copied filter to make the necessary modifications. You can also rename and delete a log filter.

To copy, rename, or delete a filter:

1. Open a log file.
2. Click **Tools** → **Manage Filters**.
3. In the Manage Log Filters dialog box, select the filter.
4. Click **Copy**, **Rename**, or **Delete**. Type a new name and click **OK**, or click **Yes** to confirm the deletion.

Working with Reports

The LogViewer has a set of predefined reports that you can use to analyze test results. You can work with reports as follows:

- ▶ Set a default report layout.
- ▶ Generate, print, or save a report.

These tasks are described in the following sections.

Setting a Default Report Layout

When you generate a quick report of the active log, the report is created using the default report layout.

To set the default layout:

1. Open a log file.
2. Click **Tools** → **Options**.
3. Select the default report layout from the **Default Report Layout** list.
4. Optionally, select a default log filter from the **Default Filter** list. For information about using filters, see *Filtering the Log Event Column* on page 10-11.
5. Click **OK**.

NOTE: The Report Layout list shows only report layouts that have already been created. To create a report layout, use the Report Layout Editor. For information, see the TestManager Help.

Generating, Printing, and Saving a Quick Report

You can generate a Quick Report of the active log. It contains a summary or detailed report of the log that you can view, print, or save to a file.

To generate a report of the active log:

1. Click **File** → **Quick Report**.
2. If there is no default layout set, the Quick Report dialog box appears. Select a report layout and click **OK**.

The report is generated using the default layout. The **Report** toolbar appears in the report window. Use this toolbar to navigate through the report or to print, save, or close it.

3. Optionally, click **File** → **Print Setup**. Set the print options and click **OK**. (For information about printer options, see the Print Manager section of your Windows documentation.)
4. Optionally, click the **Print** toolbar button to print the report. Specify the page information and click **OK**.
5. Optionally, click the **Write to File** toolbar button to save the report to a file. Specify the drive, folder, and file name where you want to store the report text and click **OK**.
6. When you have finished viewing the report, click the **Close** toolbar button.

NOTE: You can set the default report layout using the **Tools** → **Option** command.

Entering and Modifying Defects

A **defect** can be anything from a request for a new feature to an actual bug found in the application-under-test. Defect tracking is an important part of the software testing effort. Rational Suite TestStudio, Rational TeamTest, and Rational Suite PerformanceStudio all incorporate the change-request management technology of Rational ClearQuest to track defects.

You can use the LogViewer to enter defects for any verification points that fail during playback of a recorded script. When you enter a defect from the LogViewer, the LogViewer displays the TestStudio defect form and fills in several fields for you with information from the playback log. (When you enter defects this way, the LogViewer does not start ClearQuest; it opens the defect form, which is part of ClearQuest.) You can also enter defects manually using ClearQuest, but none of the fields will be automatically filled in for you.

Once you have entered defects, you can use ClearQuest to review the data and decide upon further action.

During the course of developing your application, you can update the state of each defect to keep the information current with the development-test-repair cycle. You can then use ClearQuest's reporting options to retrieve current information about the defects being tracked and the overall progress of development. You can also send defect information to other members of your development team using ClearQuest's e-mail features. For information about working with ClearQuest, see the Rational ClearQuest Help

NOTE: To use ClearQuest to store defects, an administrator must first set up the ClearQuest schema repository, and then create or attach a ClearQuest user database as part of a Rational repository. For information, see the *Using the Rational Administrator* manual.

About ClearQuest and Defect Tracking

ClearQuest is a change-request management system designed for the dynamic and interactive nature of software development. With ClearQuest, you can manage all of the change-request needs of software development — for example, enhancement requests, defect reports, and documentation modifications.

For your convenience, a specially designed schema for defect tracking is included with your software. In ClearQuest, the term **schema** refers to all attributes associated with a change-request database. This includes field definitions, field behaviors, the state transition tables, actions, and forms. For more information about ClearQuest schemas, see the Rational ClearQuest Help.

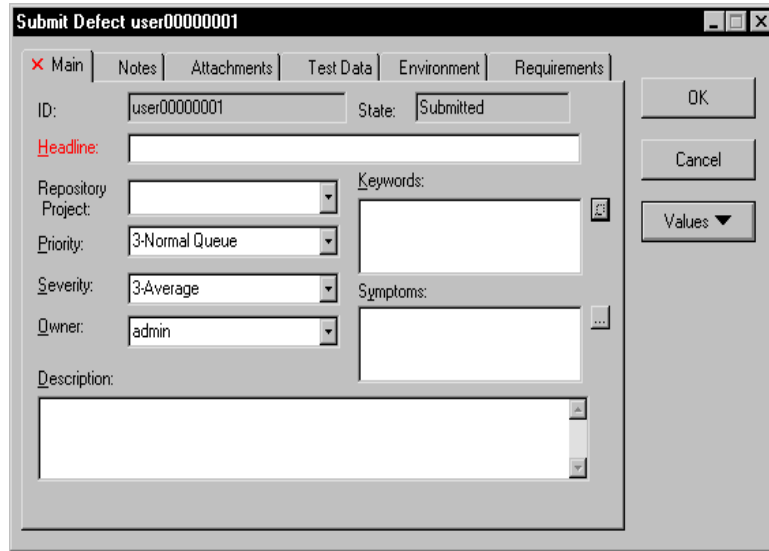
About the Rational TestStudio Schema

The **TestStudio schema** includes two TestStudio defect forms, one for entering new defects, and one for modifying and tracking defect information.

NOTE: To use the TestStudio schema, you must select it when you create or attach a ClearQuest user database as part of a Rational repository. For information, see the *Using the Rational Administrator* manual.

About the TestStudio Defect Form

You can use the TestStudio defect form to track as many or as few details about a defect as you want.

The image shows a screenshot of the 'Submit Defect user00000001' dialog box. The dialog has a title bar with the text 'Submit Defect user00000001' and standard window controls. Below the title bar are several tabs: 'Main' (selected), 'Notes', 'Attachments', 'Test Data', 'Environment', and 'Requirements'. The 'Main' tab contains the following fields: 'ID:' with the value 'user00000001', 'State:' with the value 'Submitted', 'Headline:' (a required field), 'Repository Project:' (a dropdown menu), 'Priority:' with the value '3-Normal Queue', 'Severity:' with the value '3-Average', 'Owner:' with the value 'admin', 'Keywords:' (a text area), 'Symptoms:' (a text area), and 'Description:' (a large text area). On the right side of the dialog, there are three buttons: 'OK', 'Cancel', and 'Values' (with a dropdown arrow).

The defect form contains the following tabs that let you track defects and annotate the defect tracking process:

- ▶ **Main** – Lets you specify basic details about a defect such as a brief description, the repository project, repair priority of a defect, severity, owner, keywords, and symptoms. The **Headline** and **Severity** fields are required.
- ▶ **Notes** – Lets you type additional information about a defect. If you enter a defect from the LogViewer, the software automatically enters information about the defect in this field.
- ▶ **Attachments** – Lets you associate one or more files with a particular defect. You can add, delete, copy, or open an attachment. Attachments are stored in the ClearQuest user database, along with other data contained in the record of a defect.
- ▶ **Test Data** – Lets you specify the build, log folder, log, script, verification point, and test requirement of a particular defect. For information about creating test requirements, see *Defining Test Requirements* on page 2-3.

If you enter a defect using ClearQuest, the **Test Data** information is empty. If you enter a defect from the LogViewer, it fills in the **Test Data** tab with information from the Robot playback log. To manually select test data information, you must select a valid project from the **Repository Project** box in the **Main** tab of the defect form.

- ▶ **Environment** – Lets you specify information about who found the defect, the type of computer on which the defect was found, and the testing environment. You can also customize three fields to include information pertinent to your project.
- ▶ **Requirements** – Lets you associate one or more requirements, created with TestManager or RequisitePro, with a particular defect. For information about creating test requirements, see *Defining Test Requirements* on page 2-3.
- ▶ **History** – Displays the complete history of the current defect: the date of the action, who took the action, the action taken, the old state of the defect, and the new state of the defect. The **History** tab appears after you submit a defect.

When you create a new defect, ClearQuest automatically enters the defect in the Submitted state. Use ClearQuest to change the state of an existing defect. For information, see the ClearQuest Help.

- ▶ **Resolution** – Lets you specify information related to the defect being fixed. You can enter the type of resolution, the software build in which the defect was repaired, and an in-depth explanation of the resolution. The **Resolution** tab appears after you submit a defect.

NOTE: To display information about each item in the defect form, right-click the item and click **Help**.

Starting ClearQuest

You can start ClearQuest from another Rational Test product or component, or from the Windows desktop.

NOTE: To enter a defect directly from the LogViewer, see the next section, *Entering Defects*.

To start ClearQuest:

1. Do one of the following:



- From a Rational Test product or component, click **Tools** → **Rational ClearQuest**.
 - From the Windows desktop, click **Start** → **Programs** → *Rational product name* → **Rational ClearQuest**.
2. If the ClearQuest Login dialog box appears, type your ClearQuest user ID and password. Select the database in which you want to enter, find, or modify a defect.
 3. Click **OK**.

Entering Defects

You can enter defects from the LogViewer or ClearQuest. If you use the LogViewer, it fills in many of the fields in the defect form. If you use ClearQuest, you must enter the fields manually.

To enter a defect:

1. To enter a defect from the LogViewer, do one of the following:
 - Select the failed event in the **Log Event** column, and click **Defect** → **Generate**.
 - Click **Defect** → **Find and Generate**. Click **Find Next** until the failed event for which you want to generate a defect is selected. Click **Generate Defect**.

NOTE: The LogViewer attempts to connect to ClearQuest using your Rational Test user ID and password. If they do not exist in ClearQuest, the LogViewer attempts to create a user with the same ID and password. However, if the LogViewer still cannot connect to the ClearQuest database, the Login dialog box appears. In this case, type your ClearQuest user ID and password. Select the database in which you want to enter the defect.

Skip to step 3.

2. To enter a defect from ClearQuest, do the following:
 - a. Start ClearQuest as described in *Starting ClearQuest* on page 10-18.
 - b. In ClearQuest, click **Actions** → **New Defect**, or click **New Defect** from the toolbar.
3. In the **Main** tab, type a brief description in the **Headline** box, and select the **Severity** for this defect. These two fields are required.

NOTE: A red **x** on a tab indicates that you must enter required information in that tab. A red label next to a text box means that you must enter required information in that box.

ClearQuest automatically assigns an ID number and the submitted state to each new defect. The next defect that you create receives the next available defect ID number.

4. Optionally, type or select the repository project, priority, owner, description, keywords, and symptoms in the **Main** tab.
5. Optionally, click the **Notes** tab. Type any additional information about a defect, such as how to reproduce it, in the **New Note** box.
6. Optionally, click the **Attachments** tab, and do the following:
 - a. Click **Add** and select one or more files to associate with the defect. Click **Open**.
 - b. Optionally, type a comment about the attachments in the **Attachment comment** dialog box. To apply this description to all selected attachments, click **Apply to all**. To apply the description to a single attachment, click **OK**.

To edit a description, right-click the attachment, and then click **Edit Description**. To view details of a description, right-click the attachment, and then click **View** → **Details**.

To change the way attachments appear, right-click in the Attachments box, and then click **View** → **List view**, **Small icons**, or **Large icons**.

To disassociate an attachment from a defect, select a file, and then click **Delete**.

To save a copy of a file, select a file, and then click **Save as**.

To display a file in its native editor, click **Open**. To open the appropriate editor, the file name must include an extension that Windows recognizes.

7. Optionally, click the **Test Data** tab. Select the build in which the defect was found, as well as the log folder, log, script, and verification point for the defect.

To add test requirement information:

- a. Click **Select** to display a list of requirements from those created using TestManager or RequisitePro. For information about creating test requirements, see *Defining Test Requirements* on page 2-3.

Be sure to select the correct repository project in the **Main** tab of the defect form so that the correct requirements appear.

- b. Select a requirement from those listed, and then click **OK**.

Click **Clear** to delete a selected requirement.

Click **Properties** to display the properties of a requirement.

8. Optionally, click the **Environment** tab. Type or select information describing the environment in which the defect was found.
9. Optionally, click the **Requirements** tab, and do any of the following:
 - Click **Add to List** to associate a requirement with this defect.
 - Click **Remove** to delete a selected requirement.
 - Click **Properties** to display the properties of a requirement.
10. Click **OK**.

If you enter a defect from the LogViewer, the number of the new defect appears in the **Defect** column of the LogViewer.

NOTE: You can also enter defects from SiteCheck, after you play back a Web Site Scan or Web Site Compare verification point. From the LogViewer, double-click the failed Web verification point. In SiteCheck, click **Tools** → **Enter a Defect**.

Finding Defects

To find a defect using ClearQuest:

1. Start ClearQuest, as described in *Starting ClearQuest* on page 10-18.
2. Do one of the following to find a defect:
 - Create a query to find the defect. For information about creating a query, see the ClearQuest Help.
 - If you know the ID number of the defect that you want to find, do the following:
 - a. Click **Edit** → **Find Record**.
 - b. Select **Defect** from the **Entity** box.
 - c. Type the ID number of the defect that you want to find, and then click **OK**.

Modifying Defects

To modify a defect using ClearQuest:

1. Start ClearQuest, as described in *Starting ClearQuest* on page 10-18.
2. Open the defect, as described in *Finding Defects* on page 10-21.
3. Click the **Actions** button. Click **Modify**.
4. Click any of the tabs, and then modify the information about the defect.

The information that you can modify depends on the state of the defect.
5. Optionally, click the **History** tab to view the history of the defect. Any changes that you make will not appear in this tab until you click **OK**.
6. Click **OK** when you are finished.

Reviewing Logs with the LogViewer

Using the Object Properties Comparator

This chapter explains how to use the Object Properties Comparator. It includes the following topics:

- ▶ Overview
- ▶ Starting the Object Properties Comparator
- ▶ The main window
- ▶ The Objects hierarchy and Properties list
- ▶ Locating and comparing differences
- ▶ Viewing verification point properties
- ▶ Adding and removing properties
- ▶ Editing the baseline file
- ▶ Saving the baseline file

Overview

Use the Object Properties Comparator to view and compare the properties captured when you use the Object Properties verification point in a Rational Robot script.

When you record a script that includes an Object Properties verification point, Robot creates a baseline data file that contains captured objects and their properties.

When you play back a script, Robot compares the properties in the baseline data file with the properties in the application-under-test. If the comparison fails, Robot saves the data that caused the failure to an actual data file. The events in the script (for example, call script, start schedule, verification point, user abort, unexpected active window, end script, and so on) appear in the Rational LogViewer. From the LogViewer, you can start the Comparator by double-clicking an Object Properties verification point.

In summary, you can use the Object Properties Comparator to:

- ▶ Review, compare, and analyze the differences between the baseline data file and the actual data file.
- ▶ View or edit the baseline data file for an Object Properties verification point.

Starting the Object Properties Comparator

There are two ways to start the Object Properties Comparator:

- ▶ From Robot
- ▶ From the LogViewer

Starting the Comparator from Robot

To start the Comparator from Robot:

1. Start Robot and open a script.
For Robot to open this Comparator, the script must contain an Object Properties verification point.
2. Do one of the following in the Asset (left) pane:
 - Double-click an Object Properties verification point.
 - Right-click an Object Properties verification point and click **View Baseline**.

The Object Properties Comparator opens and that verification point appears.

You can also open the Object Properties Comparator from Robot by clicking **File** → **Properties**. In the **Related Assets** tab, select an Object Properties verification point and click the **View Baseline** button.

NOTE: When you open the Comparator through Robot, you can only view the baseline file. If you have a failed verification point and want to compare the baseline and actual files, you must start the Comparator through the LogViewer.

Starting the Comparator from the LogViewer

To start the Comparator from the LogViewer:

1. Start the LogViewer and open a log file.
For the LogViewer to open this Comparator, the log must contain an Object Properties verification point.

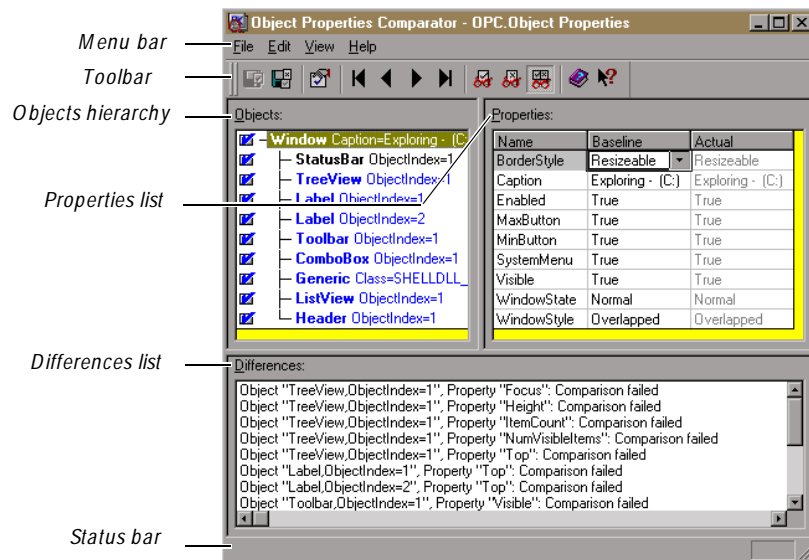
2. Do one of the following in the Log Event column:
 - Double-click an Object Properties verification point.
 - Select an Object Properties verification point and click **View** → **Verification Point**.
 - Right-click an Object Properties verification point and click **View Result**.

The Object Properties Comparator opens and that verification point appears.

If the verification point failed, the Comparator opens with both the baseline and actual files displayed.

The Main Window

The main window of the Object Properties Comparator contains the Objects hierarchy and the Properties list, the Differences list, and the toolbar, menus, and status bar.



The **Objects hierarchy** contains the list of all objects that Robot records in the Object Properties verification point. The **Properties list** contains the list of properties of those objects. When you select an object on the left, its properties appear on the right. You can control the display of both the Objects and Properties sections of the window by using the **View** commands.

The **Differences list** shows the objects that have differences between the baseline and the actual files. If you click an object in the list, that object is highlighted in the Objects hierarchy and Properties list. If you are viewing a file with no failures, this section does not appear. To show or hide this section, click **View** → **Show Difference List**.

The Objects Hierarchy and the Properties List

When the Object Properties Comparator is opened, the Objects hierarchy and Properties list appear as follows:

- ▶ The Objects hierarchy appears in the left pane of the window. It displays the list of all the objects recorded by Robot using the Object Properties verification point and saved in the baseline file.
- ▶ The Properties list appears in the right pane of the window. It displays the list of properties of the selected object, and the properties' values in the baseline file and the actual file (if there are differences).

If the verification point passed, the Comparator displays the Objects hierarchy and the Properties list with only the Baseline column.

If the verification point failed, the Comparator displays the Objects hierarchy and the Properties list with both the Baseline and Actual columns, so you can compare them.

NOTE: If the verification point contains just one object, the Objects hierarchy does not appear. To display it, click **View** → **Objects** or **View** → **Objects and Properties**.

Changing the Window Focus and Section Widths

To change the focus between the Objects hierarchy and the Properties list, do one of the following:

- ▶ Click the mouse in the section.
- ▶ Press TAB.
- ▶ Press ALT+ O to set the focus to the Objects hierarchy.
- ▶ Press ALT+ P to set the focus to the Properties list.

To change the widths of the Objects hierarchy and Properties list:

1. Position the pointer on the border separating the sections.
2. Drag the border to the right or to the left.

You can also follow the same steps to change the width of the columns within the Properties list.

Working Within the Objects Hierarchy

To display the Objects hierarchy, click **View** → **Objects** or **View** → **Objects and Properties**.

The object list is hierarchical. You can expand or collapse the view of objects by selecting a top-level object and using the **View** → **Expand** and **View** → **Collapse** commands.

When you select an object, the properties for that object are displayed in the Properties list.

Each object is listed by its object type and is bold. After the object name there may be information such as the object class or index, which can be used to identify the object. If the object is red, it has properties with different values in the baseline and the actual files. If the object is blue, it exists in the baseline file but not in the actual file.

You can do any of the following to work within the Objects hierarchy. The Objects hierarchy must have window focus.

- ▶ Press HOME, END, PAGEUP, PAGEDOWN, UP ARROW, and DOWN ARROW to move between objects.
- ▶ Click the check box that precedes each object to select or deselect it for testing. All objects preceded by a check mark are tested.
- ▶ Select an object preceded by a check mark to display its properties in the Properties list.
- ▶ Select an object and press INSERT to display a dialog box for adding and removing properties from the Properties list for that object.
- ▶ Double-click a parent object to expand or collapse its children.
- ▶ Press plus (+) to expand the highlighted object one level, or press minus (-) to collapse the highlighted object. Press asterisk (*) to expand all objects.
- ▶ Right-click an object in the hierarchy to display the Objects shortcut menu.
- ▶ Double-click an object that is labeled **Unknown** to define the object. For information, see *Defining Unknown Objects During Recording* on page 4-21.

Working Within the Properties List

To display the Properties list, click **View** → **Properties** or **View** → **Objects and Properties**.

The Name column shows the name of the property. The Baseline and Actual columns display the values for the properties. Values in the Baseline column represent the properties from the original recording of the Object Properties verification point. Values in the Actual column represent the state of the properties in the latest played back version. By default, if there are differences between the baseline and actual, both columns are displayed.

Use the **View** commands to control which columns appear in the Properties list.

If a property is red, it has different values in the baseline and the actual files. If a property is blue, it exists in the baseline file but not in the actual file. If a value cell is blank, the property has an empty value.

You can do any of the following to work within the Properties list. The Properties list must have window focus.

- ▶ Type the first letter of a property's name to move to that property or to the first property beginning with that letter.
- ▶ Press HOME, END, PAGEUP, PAGEDOWN, UP ARROW, and DOWN ARROW to highlight a property.
- ▶ Press INSERT to display a dialog box for adding and removing properties from the Properties list.
- ▶ Select a property and press DELETE to remove it from the list.
- ▶ Double-click the value cell of a property to edit the value.
- ▶ Position the pointer on the vertical border between column title cells. Drag the pointer to the right or left to change the column widths.
- ▶ Point to a property and click the right mouse button to display the Properties shortcut menu.

Locating and Comparing Differences

The Object Properties Comparator begins its comparison with the first object in the Objects hierarchy and its properties in the Properties list.

Objects that contain differences between the baseline and actual lists are red. Objects that exist in the baseline list but are missing from the actual list are blue.

To locate the first difference between the baseline data and the actual data, click **View** → **First Difference**. When the difference is located, the failure is highlighted. The Differences list indicates the failure number and provides information about the failure.

To navigate between differences, use the **View** commands.

You can also select a description in the Differences list to highlight that failure in the Properties list.

Viewing Verification Point Properties

To view verification point properties:

- ▶ Click **File** → **Verification Point Properties**.

The Verification Point Properties dialog box shows the verification point type, the name of the baseline file, and the name of the actual file.

Adding and Removing Properties

When you first create an Object Properties verification point, you can specify the properties to test by adding and removing them from the Properties list. You can also add and remove properties from the list when you view the data file in the Object Properties Comparator. This lets you refine a test even after it has been created and played back.

For example, if the Properties list for a verification point contains a Height property that you decide you do not want to test, you can remove the property in the Comparator. You can also apply the properties in the list to all objects of the same type for this verification point, and define a list of default properties for each type of object.

Adding Properties to the Properties List

To add properties to the Properties list:

1. Select an object in the Objects hierarchy.
2. Click **Edit** → **Edit Property List**, or right-click a property in the Properties list and click **Edit List**.
3. Select the properties to add from the **Available** list and click > or >>. Added properties appear in the **Selected** list.

4. Select the following as needed:

Apply to all like objects – Applies the selected properties to all objects with the same classification as the selected object.

Save as default – Saves the selected properties as defaults for all objects with the same classification as the selected object for use in future tests.

5. Click **OK**.

Removing Properties from the Properties List

Removing a property removes it from the Properties list but does not delete it from the verification point's baseline file. Removing a property means that it will no longer be tested in future playbacks. Once removed, properties can be added back later.

To remove a single property from the Properties list:

1. Select the property in the Properties list.
2. Do one of the following:
 - Click **Edit** → **Remove Property**.
 - Right-click the property and click **Remove Property**.
 - Press **DELETE**.

To remove multiple properties from the Properties list:

1. Select an object in the Objects hierarchy.
2. Click **Edit** → **Edit Property List**, or right-click the property and click **Edit List**.
3. Select the properties to remove from the **Selected** list and click < or << .
4. Select the following as needed:
 - Apply to all like objects** – Applies the selected properties to all objects with the same classification as the selected object.
 - Save as default** – Saves the selected properties as defaults for all objects with the same classification as the selected object for use in future tests.
5. Click **OK**.

If you remove a property, you can add it back to the Properties list at a later time by using the **Edit** → **Edit Property List** command.

Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the baseline file to keep it up-to-date with the developing application.

When editing the baseline file, you can:

- ▶ Edit a value in the Properties list.
- ▶ Cut, copy, and paste a value.
- ▶ Copy values from the actual to the baseline file.
- ▶ Change a verification method.
- ▶ Change an identification method.
- ▶ Replace the baseline file.

These tasks are described in the following sections.

NOTE: You cannot edit the actual data file.

Editing a Value in the Properties List

To edit a value in the Properties list:

1. Select an object in the Objects hierarchy.
2. Click a property name in the Name column of the Properties list.
3. Double-click the Value cell.

What happens when you double-click depends on what the value cell contains, as described in the following table:

If the value cell contains	The following happens when you double-click the cell
A string	A blinking cursor appears. Edit the value. ...or... A Property dialog box appears. Edit the value. (The string can contain multiple lines of text.)
A float or integer	A blinking cursor appears. Edit the value. Only numeric characters are supported. If a spin button appears, you can click the arrows to change the value.
A down-arrow	A list of available choices appears. Select a value. ...or... A blinking cursor appears. Type a value in the cell, or click the down-arrow and select a value from the list.
A color	The Color dialog box appears. Select a color from the basic colors or the color palette, or type values in the Red, Green, and Blue edit boxes.
(list)... (array)...	A Property dialog box appears, displaying all of the values in the property. Click Edit to edit any value. Click Select to highlight the items to test in the list or array.
Bitmap, OLE Object, Unknown	These properties cannot be edited.

Cutting, Copying, and Pasting a Value

You can cut and copy values from the baseline file and paste values into it. Since you cannot edit the actual file, you cannot cut a value from it or paste a value into it.

To cut, copy, and paste a value:

1. Select an object in the Objects hierarchy.
2. Select the value to cut or copy in the Properties list.
3. Click **Edit** → **Copy Property** or **Edit** → **Cut Property**.

These commands, unlike standard Windows commands, do not place values on the Windows Clipboard. Instead, they place values on an internal clipboard.

4. To paste the value, click the cell in the baseline file where you want to paste the value. (The pasted value will replace the value in the cell.)
5. Click **Edit** → **Paste Property**.

NOTE: The two values must be of similar types. For example, you can paste a string over a string or an integer over an integer.

Copying Values from the Actual to the Baseline File

To copy one of an object's property values from the actual file to the baseline file:

1. Select an object in the Objects hierarchy.
2. Select a value in red from the actual column in the Properties list.
3. Click **Edit** → **Copy Property to Baseline** or press F9.

To copy all of an object's values from the actual file to the baseline file:

1. Select an object in the Objects hierarchy.
2. Click **Edit** → **Copy All Properties to Baseline**.

Changing a Verification Method

A verification method for a property specifies how Robot compares the property captured during recording with the property captured during playback.

To change a verification method for a property:

1. Click **View** → **Show Verification Method**, or right-click in the Properties list and click **Verification Method**.
2. Select a property in the Properties list.

The current verification method of that property appears in the Verification Method box. If **Not Applicable** appears in the box, you cannot change the way that property is verified. It will always be an exact match.

The box stays open until you close it, so you can keep selecting different properties to see their verification methods.

3. Select a different verification method for a property. Note that:
 - For a (list) or (array), the selected verification method applies to all of the items.
 - If you select **Numeric Range**, type the **From** and **To** values.
 - If you select **User Defined**, type the **Library** and **Function**.

Changing an Identification Method

You can change the identification method for properties that have a list or array value. The identification method defines how the items in the list or array are identified during playback. For example, the item could be identified by Location or Content.

To view or change an identification method:

1. In the Properties list, double-click a list or array value to open the Property dialog box.
2. In the **Identification Method** list, select an identification method.
The selected method applies to all of the items.
3. Click **OK**.

Replacing the Baseline File

You may want to overwrite the baseline file with the actual file when revisions to your software application require that you update your baseline verification point data files.

Each time you run your scripts against the revised software and the verification points fail, an actual file is saved. You should compare the baseline file to the actual file to make sure that the failure was caused by an intentional change and not by a defect in the new build.

If the failure was caused by an intentional change, you can convert the actual data into the new baseline data. This updates your script with the new application state.

To replace the baseline file:

1. Click **File** → **Replace Baseline With Actual**.
2. When prompted for a confirmation, click **Yes** to replace the baseline data or click **No** to leave it unchanged.

Saving the Baseline File

To save changes made to the baseline file:

- ▶ Click **File** → **Save Baseline**.

This command is enabled only if you have made changes to the baseline file.

▶▶▶ C H A P T E R 12

Using the Text Comparator

This chapter explains how to use the Text Comparator. It includes the following topics:

- ▶ Overview
- ▶ Starting the Text Comparator
- ▶ The main window
- ▶ The text window
- ▶ Locating and comparing differences
- ▶ Viewing verification point properties
- ▶ Editing the baseline file
- ▶ Saving the baseline file

Overview

Use the Text Comparator to view and compare alphanumeric data captured when you use the Alphanumeric verification point in a Rational Robot script.

When you record a script that includes the Alphanumeric verification point, Robot creates a baseline data file that contains the data you captured. When you play back the script, Robot compares the data in the baseline file with the data in the application-under-test. If the comparison fails, Robot saves the data that caused the failure to an actual data file. The events in the script (for example, call script, start schedule, verification point, user abort, unexpected active window, end script, and so on) appear in the Rational LogViewer. From the LogViewer, you can start the Text Comparator by double-clicking an Alphanumeric verification point.

In summary, you can use the Text Comparator to:

- ▶ Review, compare, and analyze the differences between the baseline data file and the actual data file.
- ▶ View or edit the baseline data file for an Alphanumeric verification point.

Starting the Text Comparator

There are two ways to start the Text Comparator:

- ▶ From Robot
- ▶ From the LogViewer

Starting the Comparator from Robot

To start the Comparator from Robot:

1. Start Robot and open a script.

For Robot to open this Comparator, the script must contain an Alphanumeric verification point.

2. Do one of the following in the Asset (left) pane:
 - Double-click an Alphanumeric verification point.
 - Right-click an Alphanumeric verification point and click **View Baseline**.

The Text Comparator opens and that verification point appears.

You can also open the Text Comparator from Robot by clicking **File** → **Properties**. In the **Related Assets** tab, select an Alphanumeric verification point and click the **View Baseline** button.

NOTE: When you open the Comparator through Robot, you can only view the baseline file. If you have a failed verification point and want to compare the baseline and actual files, you must start the Comparator through the LogViewer.

Starting the Comparator from the LogViewer

To start the Comparator from the LogViewer:

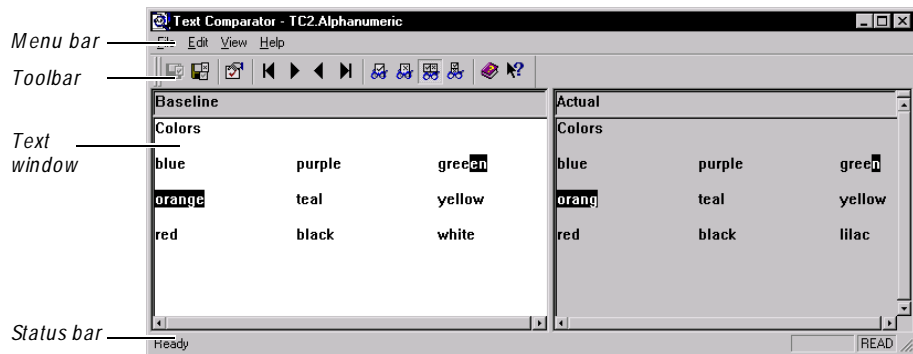
1. Start the LogViewer and open a log file.
For the LogViewer to open this Comparator, the log must contain an Alphanumeric verification point.
2. Do one of the following in the Log Event column:
 - Double-click an Alphanumeric verification point.
 - Select an Alphanumeric verification point and click **View** → **Verification Point**.
 - Right-click an Alphanumeric verification point and click **View Result**.

The Text Comparator opens and that verification point appears.

If the verification point failed, the Comparator opens with both the baseline and actual files displayed.

The Main Window

The main window of the Text Comparator contains the text window and the toolbar, menus, and status bar.



The Text Window

The text window has two panes: Baseline and Actual. The Baseline pane shows the data file that serves as a baseline file for a comparison. The Actual pane shows data from the current playback in Robot. You can control the display of the panes by using the **View** commands.

The text window uses a typical text editor format. In general, you use the same rules and methods of typing, selecting, and deleting that you would use in a standard text editor (such as Notepad).

The Baseline pane has a white background and the Actual pane has a gray background. Data that failed the comparison between the baseline file and the actual file appears in reverse color when you use one of the locating commands to highlight it.

Scrolling the Text Window

The vertical scroll bar is automatically synchronized. It is on the far right side of the window, to the right of the Actual pane. When you use it to scroll the actual data, the baseline data scrolls at the same time.

There are also independent horizontal scroll bars at the bottom of each pane.

You can also use the cursor to scroll in both the Baseline and Actual panes. Click in one of the panes to place the cursor, and then use the arrow keys to move it.

Changing the Widths of the Text Panes

To change the widths of the Baseline and Actual panes:

- ▶ Position the pointer on the vertical border between the two panes. Drag the border to the right or to the left.

Using Word Wrap

To wrap long text strings onto multiple lines within the text window, click **View** → **Word Wrap**.

When this command is checked, the data wraps to fit into the text window area.

When this command is not checked, the data displays on one line that you must scroll horizontally to read.

Locating and Comparing Differences

To locate the first difference between the baseline data and the actual data, click **View → First Difference**.

To navigate between differences, use the **View** commands.

The comparison starts in the upper left corner of the pane. The Comparator then scans for differences by going across each row of text in order, as it would in a text editor.

When a difference is found using the **View** commands, the difference between the baseline file and the actual file appears in reverse color.

The Alphanumeric verification point stores the specified verification method as part of the script command. For data files created by the Alphanumeric verification point, the Comparator assumes a case-sensitive comparison, regardless of how it was recorded. For numeric data, the Comparator assumes Numeric Equivalence as the verification method.

Viewing Verification Point Properties

To view verification point properties:

- ▶ Click **File → Verification Point Properties**.

The Verification Point Properties dialog box shows the verification point type, the name of the baseline file, and the name of the actual file.

Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the baseline file to keep it up-to-date with the developing application.

When editing the baseline file, you can:

- ▶ Edit the data.
- ▶ Cut, copy, and paste data.
- ▶ Copy data from the actual to the baseline file.
- ▶ Replace the baseline file.

These tasks are described in the following sections.

NOTE: You cannot edit the actual data file.

Editing Data in the Baseline File

To edit the data in the baseline file:

1. Click the location where you want to type.
2. Type the new data.

In general, you use the same rules and methods of typing, selecting, and deleting that you would use in a standard text editor (such as N otepad).

Cutting, Copying, and Pasting Data

You can cut or copy data from the baseline file and paste date into it. However, since you cannot edit the actual file, you cannot cut data from it or paste date into it.

To cut, copy, and paste data from the baseline file:

1. Select the data to cut or copy.

In general, you use the same rules and methods of typing, selecting, and deleting that you would use in a standard text editor (such as N otepad).

2. Click **Edit** → **Copy** or **Edit** → **Cut**.
3. To paste data, click in the same or a different baseline file. (The pasted data will be inserted where the cursor is located.) You can also select data that the pasted data will replace.
4. Click **Edit** → **Paste**.

Copying Data from the Actual to the Baseline File

You can use the **Copy to Baseline** command to copy data from the actual file into the baseline file. This is the equivalent of using the **Copy** and **Paste** commands.

To copy data from the actual file to the baseline file:

1. In the baseline file, do one of the following:
 - Click the location where you want to paste the data. The data will be pasted where the cursor is located.
 - Select some data. The pasted data will replace the selection.
2. In the actual file, select the data to copy.
3. Click **Edit** → **Copy to Baseline**.

The command copies the selected data from the actual file to the insertion point in the baseline file or replaces the selected data in the baseline file.

Replacing the Baseline File

You may want to overwrite the baseline file with the actual file when revisions to your software application require you to update your baseline verification point data files.

Each time you run your scripts against the revised software and the verification points fail, an actual file is saved. You should compare the baseline file to the actual file to make sure that the failure was caused by an intentional change and not by a defect in the new build.

If the failure was caused by an intentional change, you can convert the actual data into the new baseline data. This updates your script with the new application state.

To replace the baseline file:

1. Click **File** → **Replace Baseline With Actual**.
2. When prompted for a confirmation, click **Yes** to replace the baseline data or click **No** to leave it unchanged.

Saving the Baseline File

To save changes made to the baseline file:

- ▶ Click **File** → **Save Baseline**.

This command is enabled only if you have made changes to the baseline file.

Using the Text Comparator

Using the Grid Comparator

This chapter explains how to use the Grid Comparator. It includes the following topics:

- ▶ Overview
- ▶ Starting the Grid Comparator
- ▶ The main window
- ▶ Setting display options
- ▶ Locating and comparing differences
- ▶ Viewing verification point properties
- ▶ Using keys to compare data files
- ▶ Editing the baseline file
- ▶ Saving the baseline file

Overview

Use the Grid Comparator to view and compare data captured when you use the following verification points in a Rational Robot script:

- ▶ Object Data
- ▶ Menu
- ▶ Clipboard

When you record a script with one of these verification points, Robot creates a baseline data file containing the data you captured. When you play back the script, Robot compares the data in the baseline file with the data in the application-under-test. If the comparison fails, Robot saves the data that caused the failure to an actual data file. The events in the script (for example, call script, start schedule, verification point, user abort, unexpected active window, end script, and so on) appear in the Rational LogViewer. From the LogViewer, you can start the Grid Comparator by double-clicking an Object Data, Menu, or Clipboard verification point.

In summary, you can use the Grid Comparator to:

- ▶ Review, compare, and analyze the differences between the baseline data file and the actual data file.
- ▶ View or edit the baseline data file for a verification point.

Starting the Grid Comparator

There are two ways to start the Grid Comparator:

- ▶ From Robot
- ▶ From the LogViewer

Starting the Comparator from Robot

To start the Comparator from Robot:

1. Start Robot and open a script.
For Robot to open this Comparator, the script must contain an Object Data, Menu, or Clipboard verification point.
2. Do one of the following in the Asset (left) pane:
 - Double-click an Object Data, Menu, or Clipboard verification point.
 - Right-click an Object Data, Menu, or Clipboard verification point and click **View Baseline**.

The Grid Comparator opens and that verification point appears.

You can also open the Grid Comparator from Robot by clicking **File** → **Properties**. In the **Related Assets** tab, select an Object Data, Menu, or Clipboard verification point and click the **View Baseline** button.

NOTE: When you open the Comparator through Robot, you can only view the baseline file. If you have a failed verification point and want to compare the baseline and actual files, you must start the Comparator through the LogViewer.

Starting the Comparator from the LogViewer

To start the Comparator from the LogViewer:

1. Start the LogViewer and open a log file.

For the LogViewer to open this Comparator, the log must contain an Object Data, Menu, or Clipboard verification point.

2. Do one of the following in the Log Event column:

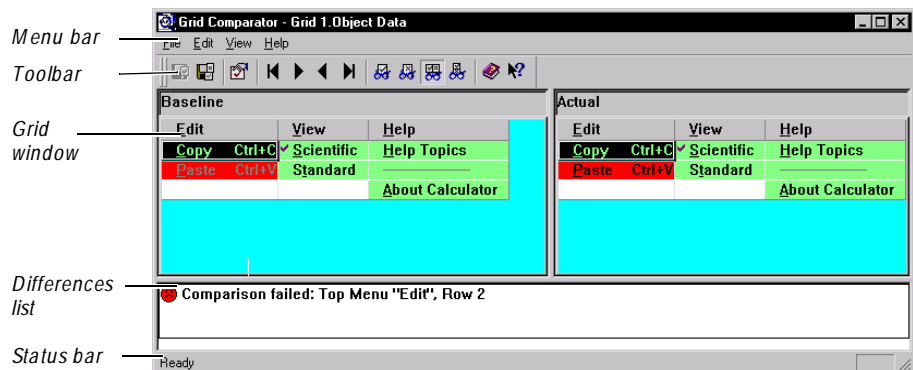
- Double-click an Object Data, Menu, or Clipboard verification point.
- Select an Object Data, Menu, or Clipboard verification point and click **View → Verification Point**.
- Right-click an Object Data, Menu, or Clipboard verification point and click **View Result**.

The Grid Comparator opens and that verification point appears.

If the verification point failed, the Comparator opens with both the baseline and actual files displayed.

The Main Window

The main window of the Grid Comparator contains the grid window, the Differences list, and the toolbar, menus, and status bar. The grid window contains the grids of data recorded by Robot in an Object Data, Menu, or Clipboard verification point. The Differences list displays descriptions of any items that failed during playback.



The Grid Window

The grid window has two panes: Baseline and Actual. The Baseline pane shows the data file that serves as a baseline file for a comparison. The Actual pane shows data from the current playback in Robot. You can control the display of the baseline and actual files using the **View** commands.






The grids in the panes show data in row and column format. Cells with a green background contain data that passed the comparison between the baseline file and the actual file. Cells with a red background failed the comparison.

You can set display options to control the grid window. For more information, see *Setting Display Options* on page 13-5.

Differences List

The Differences list displays the actual items that failed during playback. It shows the reasons why a verification point failed, and it displays icons to graphically illustrate the failure type. If you click an item in the list, that item is highlighted in the grid. If you are viewing a file with no differences, this section does not appear.

The following icons may appear in the Differences list:

Icon	Meaning
	No differences found
	Comparison failed
	Item not found
	Different sizes
	Key not found

To work in the Differences list:

- ▶ Use the vertical scroll bar to scroll through the list of descriptions.
- ▶ Select a description in the Differences list to highlight the failure in the baseline and actual files.

Setting Display Options

You can set the following display options in the Grid Comparator:

- ▶ Change the column widths.
- ▶ Transpose the grid data.
- ▶ Synchronize the scroll bars.
- ▶ Synchronize the cursors.

These tasks are described in the following sections.

Changing the Column Widths

To change the column widths:

1. Position the pointer on the vertical border between the column title cells in the grid.
2. Drag the border to the right or to the left.
3. To hide a column completely, drag its right border to the left until the column disappears.

To restore the columns to their default widths, press F8.

Transposing the Grid Data

You can view the baseline and actual grid data in the standard column format or with the rows and columns switched.

To switch the view of data in the grid, click **View** → **Transpose View**.

This command can be used for Object Data and Clipboard verification points. It is disabled for Menu verification points.

Synchronizing the Scroll Bars

You can either synchronize the scroll bars in the two panes, or you can use them independently. To link the scroll bars of the Baseline and Actual panes, click **View** → **Synchronize Scrolling**.

When this command is checked, the data in both the panes scroll at the same time when you use the scroll bars for either pane.

When this command is not checked, only the data in the pane corresponding to the scroll bar scrolls.

Synchronizing the Cursors

You can synchronize the cursors in the two panes, or you can use them independently. To synchronize the cursors in the Baseline and Actual panes, click **View → Synchronize Cursors**.

When this command is checked, the cursors in both panes highlight corresponding cells even if the cells are in different locations.

If you highlight more than one cell in a column or row in either pane, only the first cell in the other pane is highlighted.

If you select a description in the Differences list, the corresponding cells are highlighted in both panes, regardless of synchronizing cursors.

When this command is not checked, only the cell in the selected pane is highlighted.

Locating and Comparing Differences

To locate the first difference between the baseline data and the actual data, click **View → First Difference**.

To navigate between differences, use the **View** commands.

You can also select a description in the Differences list to highlight that failure in the Baseline and Actual panes.

In the grid panes, the comparison starts with the first data cell in the grid (the cell in the upper-left corner). The Comparator then scans for differences by going down the first column. At the end of the column the comparison goes to the top of the second column, and so on.

When a difference is located, the Comparator highlights the area of difference using reverse color and highlights the description in the Differences list. (You can also select a description in the Differences list to highlight that failure in the baseline and actual files.)

Verification points that have entire rows or columns selected compare the data in each cell as well as the number of cells in the row or column. If the number of cells is different, the Comparator highlights the row or column and italicizes the header number or text. It also displays a red line around the header cell.

If the data displayed in the grid is larger than the window, you can use the scroll bars to view other areas of the data, or you can resize the window.

NOTE: If a difference is highlighted in the baseline file and the description in the Differences list is *Item cannot be found*, it means that there is no difference to highlight in the actual file, since the item is missing there.

Viewing Verification Point Properties

To view verification point properties:

- ▶ Click **File** → **Verification Point Properties**.

The Verification Point Properties dialog box shows the following information:

Verification Point Type – Lists the type of verification point being displayed: Object Data, Clipboard, or Menu.

Baseline – The name of the baseline file.

Actual – The name of the displayed actual file.

Verification Method – The verification method specified when the verification point was recorded.

Test Menu States – Indicates whether menu states were included when the verification point was recorded.

Test Menu Keys – Indicates whether keyboard shortcuts and accelerators were included when the verification point was recorded.

Identification Method – The identification method specified when the verification point was recorded.

Using Keys to Compare Data Files

You can select the Key/Value identification method when you create Object Data or Clipboard verification points in Robot.

For verification points that have the Rows by Key/Value identification method, you can use the Grid Comparator to add or change keys in the baseline file. As in a relational database, keys can be used to uniquely identify a row for comparison.

You can add or change keys to determine what the important comparisons are in a verification point and to possibly change a failed verification point into one that passes.

If the value of the data in a key column changes, Robot will not be able to locate the record, and the verification point will fail. You may then want to change the keys in the Comparator to gain more insight into why the verification point failed.

If you have not specified keys that ensure uniqueness, the test can fail because Robot may compare the selected record to a record that contains similar values but is not the record that you want to test. You can experiment by changing the keys in the Comparator to improve the predictability of the verification point.

If the database schema changes, you can change the keys in the Comparator to identify new and unique columns.

To use keys to compare data files:

1. Click the name of a column in the baseline file.
2. Click the right mouse button, or press CTRL+ K to add or remove a key.

The data in the baseline and actual files should be automatically recompared. At this point you can evaluate the new key placement.

If a key column in the baseline file has different data from the actual file, the Differences list displays **Row not found: Row x** and includes the value from the baseline key column.

If there are no key columns and the row data in the baseline and actual files do not match exactly, the Differences list displays **Row not found: Row where x** and includes each column name and value from the baseline file.

Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the baseline file to keep it up-to-date with the developing application.

When editing the baseline file, you can:

- ▶ Edit the data.
- ▶ Edit a menu item.
- ▶ Cut, copy, and paste data.
- ▶ Copy data from the actual to the baseline file.
- ▶ Save the baseline file.

These tasks are described in the following sections.

NOTE: You cannot edit the actual data file.

Editing Data in the Baseline Grid

To edit the data in the baseline file:

1. Double-click a data cell, or select a cell and press ENTER.

The cell is highlighted in reverse color, and the blinking cursor appears at the end of the last character in the cell.

2. Type the new data.

The change is made when you click outside of that cell or press ENTER.

If you edited a cell that had an identical value in the baseline and actual files, the change results in a difference in the two files. The cells in the two files become red, and that value is listed in the Differences list.

Editing a Menu Item

You can edit a menu item in the baseline file of an Object Data or Menu verification point.

To edit a menu item:

1. Double-click a menu item cell in the grid.
2. In the Edit Menu Item dialog box, set the **Menu Item** option.
 - If the menu item is a command, click **Text** and type the name of the menu command. During playback, Robot tests that the menu item is the same as the text that appears in the box.

To designate one of the letters in the menu command name to be an accelerator key, type an ampersand character (&) before that letter. For example, for the **File** → **New** command, the grid text for *New* would be *Ne&w* if *w* is the accelerator key for the command.

- Click **Custom** if it is a custom item. During playback, Robot tests a custom menu item that it cannot capture.
 - Click **Separator** to make the item a separator. During playback, Robot tests that the menu item is a horizontal line.
3. Set the **Menu State** option by clicking one of the three menu states: **Enabled**, **Disabled**, or **Grayed**.

During playback, Robots tests that the menu item is in that state.

4. Optionally, set the **Checkmark** option.

This option is used for toggle-type menu commands. It determines whether the menu command is checked in the application. Select the option if you want the command to have a check mark. Clear the option if you want the menu to be unchecked. The actual bitmap used for the check mark may vary between applications.

5. Click **OK**.

You can double-click a cell in the actual file to view the attributes of a menu item, but you cannot change the attributes.

Cutting, Copying, and Pasting Data

You can cut or copy data from the baseline file and paste data into it. However, since you cannot edit the actual file, you cannot cut data from it or paste data into it.

Note that the **Cut** command does not cut the entire grid row. It cuts only the contents of the value cells.

To cut, copy, and paste data:

1. Select the data to cut or copy.

You can select one or more grid cells to be copied or cut. To select multiple cells, click cells while pressing the **SHIFT** key.

2. Click **Edit** → **Copy** or **Edit** → **Cut**.

3. To paste data, click the grid cell in the same or a different baseline file. (The pasted data replaces the data in the cell.)

You can paste individual data cells or entire columns, including the column header.

4. Click **Edit** → **Paste**.

NOTE: The currently selected cell in the baseline file is used as the top-left corner of the area being pasted into. A range of data is pasted into the cells starting at the upper-left corner and filling in the other cells as needed.

Copying Data from the Actual to the Baseline File

You can use the **Copy to Baseline** command to copy data from an actual file into a baseline file. This is the equivalent of using the **Copy** and **Paste** commands.

To copy data from the actual file to the baseline file:

1. Select the data that you want to copy from the actual file.
2. Click **Edit** → **Copy to Baseline**.

This command copies all values that are different in the highlighted selection of the actual file, replacing the corresponding cells in the baseline.

To select multiple cells, click cells while pressing the **SHIFT** key.

If **Row not found** appears in the Differences list, **Copy to Baseline** does not paste the data into the baseline file.

Replacing the Baseline File

You may want to overwrite the baseline file with the actual file when revisions to your software application require that you to update your baseline verification point data files.

Each time you run your scripts against the revised software and the verification points fail, an actual file is saved. You should compare the baseline file to the actual file to make sure that the failure was caused by an intentional change and not by a defect in the new build.

If the failure was caused by an intentional change, you can convert the actual data into the new baseline data. This updates your script with the new application state.

To replace the baseline file:

1. Click **File** → **Replace Baseline With Actual**.
2. When prompted for a confirmation, click **Yes** to replace the baseline data or click **No** to leave it unchanged.

Saving the Baseline File

To save changes made to the baseline file:

- ▶ Click **File** → **Save Baseline**.

This command is enabled only if you have made changes to the baseline file.

Using the Grid Comparator

Using the Image Comparator

This chapter explains how to use the Image Comparator. It includes the following topics:

- ▶ Overview
- ▶ Starting the Image Comparator
- ▶ The main window
- ▶ Locating and comparing differences
- ▶ Changing how differences are determined
- ▶ Changing the color of masks, OCR regions, or differences
- ▶ Moving and zooming an image
- ▶ Viewing image properties
- ▶ Working with masks
- ▶ Working with OCR regions
- ▶ Replacing and saving the baseline file
- ▶ Viewing an unexpected active window

Overview

Use the Image Comparator to open and view bitmap images captured when you use the following verification points in a Rational Robot script:

- ▶ Region Image
- ▶ Window Image

When you record a script with one of these verification points, Robot creates a baseline image file containing the image you captured. When you play back the script, Robot compares the image in the baseline file with the image in the application-under-test. If the comparison fails, Robot saves the image that caused the failure to an actual image file. The events in the script (for example, call script, start schedule, verification point, user abort, unexpected active window, end script, and so on) appear in the Rational LogViewer. From the LogViewer, you can start the Image Comparator by double-clicking a Region Image or Window Image verification point.

You can use the Image Comparator to:

- ▶ Review and analyze the differences between the baseline image file and the actual image file.
- ▶ Edit the Region Image or Window Image verification points by creating masks on the image.
- ▶ Create OCR regions to read the text within a region.
- ▶ View images of unexpected active windows that cause a failure during a script's playback.

Starting the Image Comparator

There are two ways to start the Image Comparator:

- ▶ From Robot
- ▶ From the LogViewer

Starting the Comparator from Robot

To start the Comparator from Robot:

1. Start Robot and open a script.

For Robot to open this Comparator, the script must contain a Region Image or Window Image verification point.

2. Do one of the following in the Asset (left) pane:
 - Double-click a Region Image or Window Image verification point.
 - Right-click a Region Image or Window Image verification point and click **View Baseline**.

The Image Comparator opens and that verification point appears.

You can also open the Image Comparator from Robot by clicking **File** → **Properties**. In the **Related Assets** tab, select a Region Image or Window Image verification point and click the **View Baseline** button.

NOTE: When you open the Comparator through Robot, you can only view the baseline file. If you have a failed verification point and want to compare the baseline and actual files, you must start the Comparator through the LogViewer.

Starting the Comparator from the LogViewer

To start the Comparator from the LogViewer:

1. Start the LogViewer and open a log file.

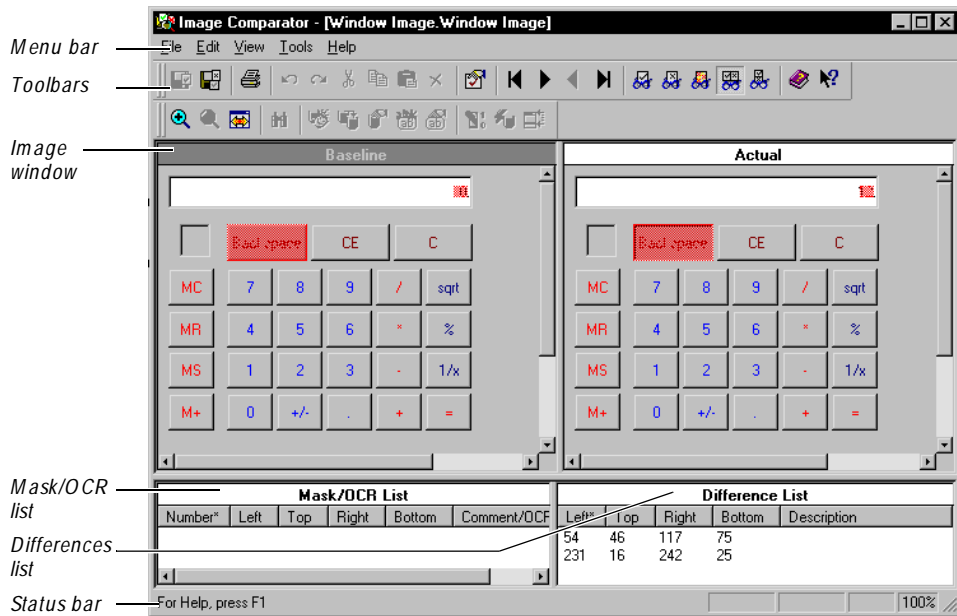
For the LogViewer to open this Comparator, the log must contain a Region Image or Window Image verification point.
2. Do one of the following in the Log Event column:
 - Double-click an Image verification point.
 - Select an Image verification point and click **View** → **Verification Point**.
 - Right-click an Image verification point and click **View Result**.

The Image Comparator opens and that verification point appears.

If the verification point failed, the Comparator opens with both the baseline and actual files displayed. If it does not, click **View** → **Both Horizontal** or **View** → **Both Vertical**.

The Main Window

The main window of the Image Comparator contains the image window, the Mask/OCR List, the Differences List, and the toolbars, menus, and status bar.



The Image Window

The image window has two panes: Baseline and Actual. The Baseline pane shows the image file that serves as an expected file for a comparison. The Actual pane shows the image from the current playback in Robot. You can control the display of both panes by using the **View** commands.

The parts of the image that passed the comparison between the baseline file and the actual file appear exactly as they were recorded. The parts of the image that failed the comparison (that is, the differences) are shown as red regions.

You can move the image within a pane and zoom the image. For information, see *Moving and Zooming An Image* on page 14-8.

Differences List

The Differences List displays a list of the items that failed during playback. The Left, Right, Top, and Bottom columns represent the measurement of the sides of the difference area, in numbers of pixels. The number in the Left column is the number of pixels from the left margin to the left edge of the difference region. The number in the Right column is the number of pixels from the left margin to the right edge of the difference region. In the same manner, the Top and Bottom columns define the number of pixels to the top and bottom edges of the difference region, from the top margin.

To work in the Differences List:

- ▶ Use the vertical scroll bar to scroll through the list of descriptions.
- ▶ Select a description in the Differences list to highlight the failure in the baseline and actual files.
- ▶ Double-click an item in the list to cause the image to be positioned so that the region is centered in the view. It will momentarily flash, and then become selected.
- ▶ The Difference list is sortable by column. The currently sorted column is indicated with an asterisk. To sort by a different column, click the column header. The list is sorted in ascending order of the selected column.

Mask/OCR List

Masks are used to hide the underlying masked area from comparison when scripts are played back. Any areas of the image that contain a mask will not be compared when you play back a script containing an Image verification point.

Robot uses OCR regions to read the text within a designated region and compare it in subsequent playbacks of the script.

The Mask/OCR List in the lower left pane of the main window lists any masks and OCR regions being used in the verification point. When you select a mask or OCR region in the list, it is highlighted in the baseline and actual files. This list works in the same way that the Differences List works, as described in the previous section. This section is empty if you do not have any masks or OCR regions defined for the verification point.

The Left, Right, Top, and Bottom columns represent the measurement of the sides of the mask or OCR region in number of pixels. This measurement works in the same way as it works in the Difference List. The Comment column for masks contains optional comments, which you can add by selecting a mask and clicking **Edit** → **Mask Properties**. The OCR Text column for OCR regions contains the text in the region that will be tested.

The Status Bar

The status bar at the bottom of the main window provides useful information as you work with the Comparator. To show or hide the status bar, choose **View → Status Bar**.

The message area in the leftmost part of the status bar displays menu command descriptions and operational messages, such as progress updates while the Comparator is scanning the image for differences.

On the right side, there are four small panes for specific information:

ReadOnly – Indicates a read-only state. This happens if the current baseline is not displayed since the current baseline is the only file that you can edit.

Load CBL – Indicates that the current baseline is not being displayed. If you want to make edits, click **File → Load Current Baseline** to display the current baseline.

BLINK – Indicates that the Blink feature is turned on.

< zoom percentage > – Indicates the zoom percentage of the window. If you have the original or normal view, the zoom percentage is 100%. If you have zoomed to some percentage of the normal view, that percentage is shown. If you have fit the image to the window, *FITTED* appears.

Locating and Comparing Differences

To display differences in the baseline and actual images, click **View → Show Differences**.

To locate the first difference, click **View → First Difference**. When a difference is located, the Comparator flashes it briefly, centers the difference in the panes, and then selects it in both panes.

To navigate between differences, use the **View** commands.

You can also select a difference in the Differences List to highlight that failure in the baseline and actual images.

Changing How Differences are Determined

Each difference region represents a logical set of differing pixels — a cluster of differing pixels close together. Depending on your preference setting, the Comparator determines whether this region is close enough to the last one to be classified as either the same or a different difference region. Every time the Comparator defines a new region around a differing pixel, it determines whether the region is close enough to any other previously defined region. If so, the Comparator combines the two rectangular regions. Otherwise, the region becomes a new difference region.

To change how differences are determined:

1. Click **Tools** → **Options**.

Use this setting to specify how close is close enough when a new differing pixel has been found.

2. Change the setting under **Difference Regions**. Move the sliding bar to choose whether you want more or fewer difference regions to be created.

The default setting is the second of the four possible choices. When you move the bar, the picture next to the slide is a representation of that choice.

Changing the Color of Masks, OCR Regions, or Differences

To change the color of masks, OCR regions, or differences in the image window:

1. Click **Tools** → **Options**.

2. Change the setting under **Colors**.

Masks – Select the highlight color for masks in the image. The masks are displayed as a block of this color in the baseline and actual files. The default color is a light green. Click **Change** to select a different color.

Differences – Select the highlight color for differences in the image. The difference regions are displayed as a block of this color in the baseline and actual files. The default color is a light red. Click **Change** to select a different color.

OCR regions – Select the highlight color for OCR regions in the image. The regions are displayed as a block of this color in the baseline and actual files. The default color is a light blue. Click **Change** to select a different color.

Moving and Zooming An Image

There are several ways to move the image within the Baseline and Actual panes:

- ▶ Use the horizontal and vertical scroll bars. Scrolling is synchronized if you are viewing both files.
- ▶ Use the moving hand pointer. If you hold down the left mouse button anywhere in the image that is not a mask or a difference region, the mouse pointer turns into a hand. You can then use it to move the image around in the window.

You can use the zooming commands to move around the image. You can zoom in, zoom out, zoom by percentage, fit the image exactly to the window, or return to the normal image size.

- ▶ To zoom in on the image, click **View** → **Zoom** → **Zoom In**.

This zooms in on the image by a factor of 2. If you have a mask, OCR region, or difference region selected when you use the Zoom command, the zooming is centered on that region. If you do not have a region selected, the zoom is centered on the entire image. You can use the command repeatedly to keep zooming into the image.

- ▶ To zoom out from the image, click **View** → **Zoom** → **Zoom Out**.
- ▶ To zoom the normal display of the image by a percentage, click **View** → **Zoom** → **Zoom Special** and the percentage.
- ▶ To restore the image to its original size, click **View** → **Zoom** → **Normal Size**.
- ▶ To fit the image to the full size of the pane, click **View** → **Zoom** → **Fit To Window**.

Zoom factors always retain the image's aspect ratio to ensure that text and images appear without distortion. **Fit To Window** represents the largest zoom factor that can display the entire image in the window while maintaining the image's aspect ratio.

Viewing Image Properties

To view the properties of an image:

- ▶ Click **File** → **Properties**.

The Image Properties dialog box shows information about the image, including its scale, color, size, and the creation date of the file.

Working with Masks

You can create masks in the Image Comparator with the **Edit** → **New Mask** command. Masks are used to hide the underlying masked area from comparison when scripts are played back. Any areas of the image that contain a mask are not compared when you play back a script that contains an Image verification point.

Use masks to ensure that certain regions are not tested. For instance, if your application has a date field, you might want to mask it so that it will not produce a failure every time the script is played back. You can also apply masks to hide differences that you determine were caused by intentional changes to the application, so that they do not cause failures in future tests.

Since you can only edit the baseline file, you cannot perform the following procedures in the actual file. However, when you select a mask in the baseline file, the mask is also selected in the actual file. You cannot modify the mask in the actual file — it is shown there as well for convenience only.

You can do the following tasks with masks:

- ▶ Display masks.
- ▶ Create masks.
- ▶ Move and resize masks.
- ▶ Cut, copy, and paste masks.
- ▶ Duplicate masks.
- ▶ Delete masks.
- ▶ Automatically mask differences.

These tasks are described in the following sections.

Displaying Masks

To display masks in the image, click **View** → **Show Masks**.

When the command is checked, the masks are displayed in the baseline and actual images.

If you do not have any masks on the current verification point, the **Show Masks** command is disabled.

Creating Masks

To create a mask directly from the Image Comparator:

1. Open an image file through the LogViewer. (Double-click a passed or failed Region Image or Window Image verification point in the log.)
2. If *Load CBL* appears in the status bar of the Image Comparator, click **File** → **Load Current Baseline**.
3. Click **Edit** → **New Mask**.

Your mouse pointer turns into a drawing tool when positioned over the image. (If you activate the drawing tool accidentally, press the ESC key.)

4. Depress the left mouse button and move the tool to draw the mask. You can draw a rectangular mask of any size over any portion of the image. When you release the mouse button, the mask is drawn and the mouse pointer becomes a normal pointer again.
5. If the mask is not visible, click **View** → **Show Masks**.
6. Move or resize the pasted mask if necessary. See the next section, *Moving and Resizing Masks*.

If you want to automatically mask a difference in the image without drawing the mask, see *Automatically Masking a Difference* on page 14-12.

NOTE: You can also add masks from Robot while recording a Window Image or Region Image verification point. In the Capture Window Image or Region Image dialog box, click the **Edit** button. That opens the Image Comparator, where you can click **Edit** → **New Mask** to add a mask.

Moving and Resizing Masks

Once a mask is created, you can move it or resize it.

Moving Masks

To move a mask, do one of the following:

- ▶ Select the mask and use the left mouse button to drag it to a new location on the image.
- ▶ Select the mask and use the arrow keys to move it.
- ▶ Select the mask and click **Edit** → **Mask Properties**, or double-click the mask. In the Mask Properties dialog box, change the **Position** setting, and then click **OK**.

You cannot move a mask outside the limits of the image.

Resizing Masks

To resize a mask, do one of the following:

- ▶ Select the mask and drag one of the handles to resize it.
- ▶ Select the mask and click **Edit** → **Mask Properties**, or double-click the mask. In the Mask Properties dialog box, change the **Size** setting, and then click **OK**.

Cutting, Copying, and Pasting Masks

You can cut or copy a mask from the baseline file and paste a mask into it. However, since you cannot edit the actual file, you cannot cut a mask from it or paste a mask into it.

A mask can only be pasted into the baseline image during the current session.

To cut, copy, and paste a mask:

1. If *Load CBL* appears in the status bar, click **File** → **Load Current Baseline**.
2. Click a mask to select it.
3. Click **Edit** → **Copy** or **Edit** → **Cut**.
4. Place the pointer over the area in the baseline image where you want to paste the mask.
5. Do one of the following:
 - Press **CTRL+ V**. The pasted mask is centered on the spot where the pointer is located.
 - Click **Edit** → **Paste**. The pasted mask is centered in the image window.
6. Move or resize the pasted mask if necessary. See *Moving and Resizing Masks* on page 14-10.

Duplicating Masks

The **Duplicate** command copies and pastes a mask in one operation, and then it places a copy of the mask on the Clipboard.

To duplicate a mask:

1. Select the mask.
2. Click **Edit** → **Duplicate**.

This copies the selected mask and pastes it into the image. The copied mask is pasted slightly below and to the right of the original mask.

3. Move or resize the pasted mask if necessary. See *Moving and Resizing Masks* on page 14-10.

Since the mask is still on the Clipboard, you can optionally paste additional copies of the mask using the **Edit** → **Paste** command.

This procedure is useful if you need to use many masks of the same size. For instance, you may need a mask the size of an edit box in an application that has many edit boxes of the same size, or you may need one the exact size of a toolbar button to mask some of the buttons in your application.

You can edit only the baseline file, not the actual file.

Deleting Masks

To delete a mask:

1. Select the mask.
2. Click **Edit** → **Delete**.

You can edit only the baseline file, not the actual file.

Automatically Masking a Difference

You can automatically turn a difference region into a mask region. This feature is useful if a difference found in playback is an intentional change in the application and not a defect, or if you want to mask a region that will always cause a difference, such as a date in a file. When you do this, the region will not fail in subsequent playbacks.

To automatically mask a difference:

1. Select the difference to mask.
2. Click **Edit** → **Auto Mask**.

The selected difference is automatically masked. The mask is the exact dimensions of the difference.

Working with OCR Regions

Robot uses Optical Character Recognition (OCR) regions to read the text within a designated region and compare it in subsequent playbacks of the script.

You can use OCR regions to verify proper operation of an application that dynamically paints text in window areas or where the actual text is difficult to obtain. OCR regions are also valuable in situations where a text string's font or weight may change unexpectedly but go undetected using traditional verification methods. To achieve the correct verification, you can define OCR regions on existing or newly-captured Image verification points.

You can do the following tasks with OCR regions:

- ▶ Create an OCR region.
- ▶ Move and resize OCR regions.
- ▶ Cut, copy, and paste OCR regions.
- ▶ Duplicate OCR regions.
- ▶ Delete OCR regions.

These tasks are discussed in the following sections.

NOTE: For more detailed information about OCR regions, including tips for using them, see the Image Comparator Help.

Creating an OCR Region

To create an OCR region directly from the Image Comparator:

1. Open an image file through the LogViewer. (Double-click a passed or failed Region Image or Window Image verification point in the log.)
2. If *Load CBL* appears in the status bar of the Image Comparator, click **File** → **Load Current Baseline**.
3. Click **Edit** → **New OCR Region**. The pointer becomes a drawing tool, which you use to outline the OCR region. Drag the pointer to create the region, and then release the mouse button.

The OCR Properties dialog box appears for the newly created region.

4. If the region contains white text on a dark background, select **Light text**. If the region has text on a gray or dark background, select **Gray background**. If the language in which the underlying text is written is not English, select a different language in the **Language** box. Click **OK**.

The Image Comparator creates an OCR on the region by reading the text within. The Comparator lists the region in the Mask/OCR List.

Note that for the first instance of OCR, it may take a few seconds to initialize the OCR engine.

5. Create as many OCR regions as you want using these steps.

NOTE: You can also add OCR regions from Robot while recording a Window Image or Region Image verification point. In the Capture Window Image or Region Image dialog box, click the **Edit** button. That opens the Image Comparator, where you can click **Edit** → **New OCR Region**.

Moving and Resizing OCR Regions

When you move or resize an OCR region, the region is automatically updated — the Image Comparator rereads the text within the region in its new location or size. The new information is then displayed in the **Mask/OCR List**.

If your OCR region is picking up extraneous characters, you can fix the region by moving or resizing it, and you can immediately verify whether it contains the appropriate text by looking at the text in the **Comment/OCR Text** column.

NOTE: For tips on the best operation of OCR, see *OCR Tips* in the Image Comparator Help.

Moving OCR Regions

To move an OCR region, do one of the following:

- ▶ Select the region and use the left mouse button to drag it to a new location on the image.
- ▶ Select the region and use the arrow keys to move it.
- ▶ Select the region and click **Edit** → **OCR Properties**, or double-click a region in the image window. In the OCR Region dialog box, change the **Position** setting, and then click **OK**.

You cannot move an OCR region outside the limits of the image.

Resizing OCR Regions

To resize an OCR region, do one of the following:

- ▶ Select the region and drag one of the handles to resize it.
- ▶ Select the region and click **Edit** → **OCR Properties**, or double-click the region. In the OCR Region dialog box, change the **Size** setting, and then click **OK**.

You cannot resize an OCR region outside the limits of the image.

Cutting, Copying, and Pasting an OCR Region

You can cut or copy an OCR region from the baseline file and paste a region into it. However, since you cannot edit the actual file, you cannot cut an OCR region from it or paste a region into it.

A copied OCR region can only be pasted into the baseline image during the current session.

To cut, copy, and paste an OCR region:

1. Click an OCR region.
2. Click **Edit** → **Copy** or **Edit** → **Cut**.
3. Place the pointer over the area in the baseline image where you want to paste the region.
4. Do one of the following:
 - Press **CTRL+ V**. The pasted region is centered on the spot where the pointer is located.
 - Click **Edit** → **Paste**. The pasted region is centered in the image window.
5. Move or resize the region if necessary.

When you paste an OCR region, OCR is automatically performed on the new region. The Image Comparator reads the text within the region, and the new information is then displayed in the **Mask/OCR List**.

Duplicating OCR Regions

The **Duplicate** command copies and pastes an OCR region in one operation, and then it places a copy of the region on the Clipboard.

To duplicate an OCR region:

1. Select the OCR region.
2. Click **Edit** → **Duplicate**.

This copies the selected region and pastes it into the image. The copied region is pasted slightly below and to the right of the original region.

When you duplicate an OCR region, OCR is automatically performed on the new region. The Image Comparator reads the text within the region, and the new information is then displayed in the Mask/OCR List.

3. Move or resize the pasted region if necessary.

Since the region is still on the Clipboard, you can optionally paste additional copies of the region with the **Edit** → **Paste** command.

This procedure is useful if you need to use many OCR regions of the same size, or with the same option settings. For example, you might want to recognize all the text on an application's toolbar icons, which are all black text on a gray background and are in Norwegian. To verify this text, you would create an OCR region the size of a toolbar icon and with the **Gray background** and **Norwegian** language options selected. Then, you would use the **Duplicate** command multiple times and reposition each new region.

Deleting OCR Regions

To delete an OCR region:

1. Select the region.
2. Click **Edit** → **Delete**.

You can edit only the baseline file, not the actual file.

Replacing the Baseline File

You may want to overwrite the baseline file with the actual file when revisions to your software application require you to update your baseline verification point data files.

Each time you run your scripts against the revised software and the verification points fail, an actual file is saved. You should compare the baseline file to the actual file to make sure that the failure was caused by an intentional change and not by a defect in the new build.

If the failure was caused by an intentional change, you can convert the actual image into the new baseline image. This updates your script to the new application state.

To replace the baseline:

1. Click **File** → **Replace Baseline With Actual**.
2. When prompted for a confirmation, click **Yes** to replace the baseline data or click **No** to leave it unchanged.

Saving the Baseline File

To save changes made to the baseline file:

- ▶ Click **File** → **Save Baseline**.

This command is enabled only if you have made changes to the baseline file.

Viewing Unexpected Active Window

Robot is designed to respond to unexpected active windows (UAW) during script playbacks. An unexpected active window is any unscripted window appearing during script playback that interrupts the playback sequence and prevents the expected window from being made active. An example of a UAW is an error message generated by the application-under-test, or an e-mail notification message window.

You can view the unexpected window in the Image Comparator only if you have set up the option in Robot. In the GUI Playback dialog box in Robot, click the **Unexpected Active Window** tab. Make sure that the **Detect unexpected active windows** and the **Capture screen image** options are both selected. (For more information, see *Setting Unexpected Active Window Options* on page 9-10.)

To open a UAW to view in the Comparator:

1. Start the LogViewer and open a log file containing a UAW.
2. Do one of the following in the Log Event column:
 - Double-click an unexpected active window event.
 - Select an unexpected active window event and click **View** → **UAW**.

The Image Comparator opens and that UAW appears.

▶ ▶ ▶ Part V

Running Queries and Reports

Querying the Rational Repository

This chapter explains how to use queries to manage the scripts, schedules, and sessions in your projects. The chapter includes the following topics:

- ▶ Overview
- ▶ Running queries
- ▶ Creating new queries
- ▶ Editing existing queries
- ▶ Setting query options
- ▶ Configuring the Query window

Overview

A **query** is a request for information stored in the repository.

Rational Test provides a powerful set of query tools that help you manage the vast amount of information accumulated during the software testing effort. All of this information is stored in the repository and is accessed with the query tools in TestManager, Robot, and LoadTest. You can use the default queries provided with TestManager, or you can create queries of your own.

For information about querying the Rational ClearQuest defect database, see the ClearQuest Help.

Running Queries

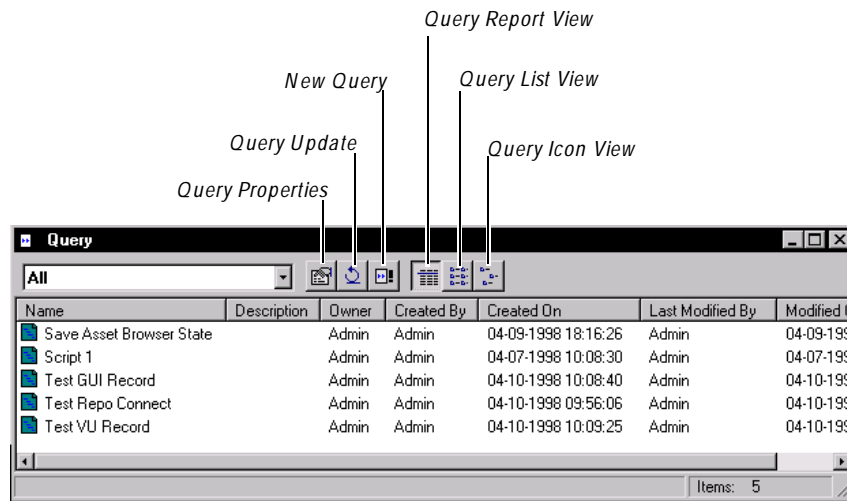
TestManager includes several default queries that you can use. There are queries for retrieving all scripts from the repository, or for retrieving all GUI scripts or all VU scripts. There are similar default queries for schedules and sessions.

To run a query, do one of the following:

- ▶ In the Asset Browser, right-click a query (script, schedule, or session) and then click **Run**.
- ▶ Click **Query** → **Run**, select a query type from the list, select a query, and then click **OK**.

The Query Window

Query results are displayed in the Query window, as shown in the following figure. This window contains a toolbar for selecting commands and a grid for displaying the results of your query.



You can use the toolbar buttons to display a query’s properties, to retrieve updated data from the repository, to create a new query, and to change how the query results are displayed.

For more information about the toolbar buttons, see the TestManager Help. For information about refreshing the Query window with the most up-to-date information from the repository, see *Setting Query Options* on page 15-9.

You can access the Query window (minus some of the toolbar buttons) from other parts of Rational Test, such as the Robot Record and Playback dialog boxes and the LoadTest Open Schedule dialog box.

NOTE: If no items (scripts, schedules, or sessions) are listed in the Query window, click the **Query Update** button on the toolbar. To make items appear automatically, click **Automatic** in the Query View tab of the TestManager Options dialog box. For more information about query options, see *Setting Query Options* on page 15-9.

Deleting Scripts, Schedules, and Sessions

NOTE: Schedules and sessions are available only with Rational Suite PerformanceStudio.

During your testing effort, you may find scripts, schedules, and sessions that you no longer need. You can delete them from the Query window.

To delete a script, schedule, or session:

1. Run a query, as described in *Running Queries* on page 15-2.
2. From the Query window, right-click the item you want to delete and click **Delete Script**, **Delete Schedule**, or **Delete Session**.
3. When TestManager asks if you want to delete the item, click **Yes**.
4. If you are deleting a session, you are asked whether or not you want to delete all of the scripts contained within the session.
 - Click **Yes** to delete all of the scripts.
 - Click **No** to delete the session without deleting any of the scripts contained within the session.

You can also delete scripts and schedules from the Requirements Hierarchy in TestManager and from Robot. For information about deleting scripts from within the Requirements Hierarchy, see *Deleting Scripts* on page 2-18. For information about deleting schedules from within the Requirements Hierarchy, see *Deleting Schedules* on page 2-21. For information about deleting scripts using Robot, see *Deleting Scripts* on page 7-15.

Creating New Queries

The major tasks involved in creating a new query are:

1. Open the Query Properties dialog box and type a query name.
2. Choose the fields or columns that you want the query to display.
3. Click the **Sort** tab and specify the sort order.
4. Click the **Filters** tab and define one or more filters to narrow down the amount of data to display.

These tasks are described in more detail in the following sections.

Opening the Query Properties Dialog Box

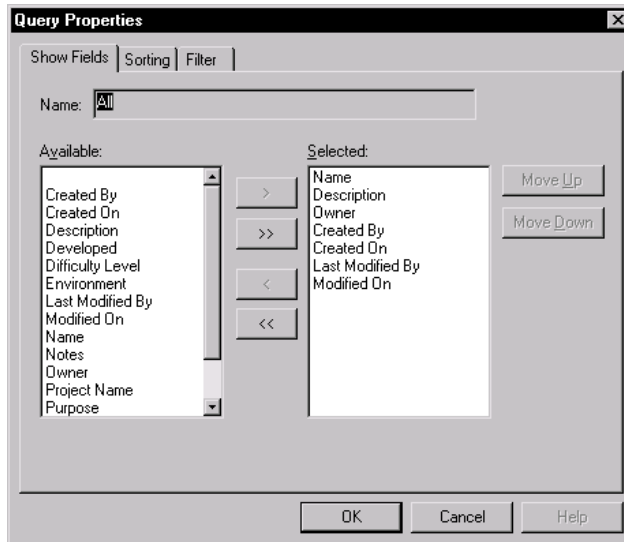
You can open the Query Properties dialog box using menu commands or the Asset Browser.

To open the Query Properties dialog box using menu commands:

- ▶ Click **Query** → **New** and click a query command.

To open the Query Properties dialog box using the Asset Browser:

1. Click **View** → **Asset Browser** to display the Asset Browser.
2. Right-click a queries folder, and then click **New Query**.



Choosing Fields to Display

Each query that you create contains several fields or attributes that are predefined in the repository. When you create a query, you can choose to display some or all of these fields. Each field appears as a column in the Query window. Some of the query fields include the query name, description, purpose, owner, environment, and type.

To choose the fields to display:

1. Open the Query Properties dialog box. For details, see *Opening the Query Properties Dialog Box* on page 15-4.
2. Select one or more files in the **Available** list, and then click > or > > .
3. In the **Selected** list, click **Move Up** or **Move Down** to set the display order.
4. Click **OK** to finish defining the query, or click the **Sorting** tab or the **Filters** tab to add additional query properties.

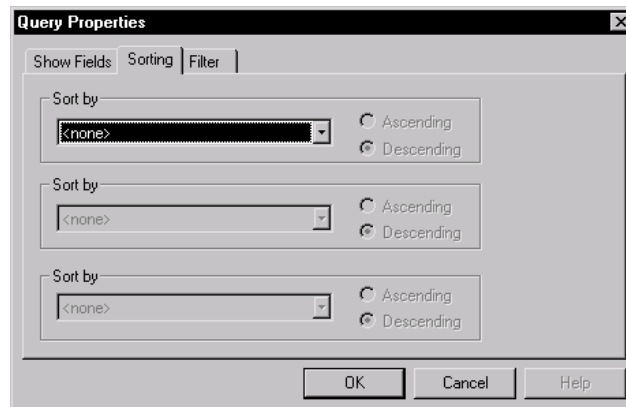
Specifying the Sort Order

By specifying a sort order, you can control the order in which query items are displayed. TestManager lets you define three sort levels.

You sort the query by the field specified in the first sort box, then by the field specified in the second and third sort boxes. You can sort in either ascending or descending order.

To sort the items in query:

1. Open the Query Properties dialog box. For details, see *Opening the Query Properties Dialog Box* on page 15-4.
2. Click the **Sorting** tab.



3. Select a field from the **Sort by** list and click either **Ascending** or **Descending**.
4. To add a second-level sort, select a field from the second **Sort by** list and click either **Ascending** or **Descending**.
5. To add a third-level sort, select a field from the third **Sort by** list and click either **Ascending** or **Descending**.
6. Click the **Filters** tab to add additional query properties, or click **OK** to finish defining the query.

You can also sort by clicking any column header of the Query window.

Adding a Filter Statement

A **filter** is the part of the query that specifies the information to be retrieved from the repository. Filter statements help you make queries more specific by narrowing down the information you are searching for. You can build simple filter statements or combine simple filter statements into more complex ones. For detailed information about the options that are available in filter statements, see the TestManager Help.

A Filter consists of two parts:

- ▶ Build filter statement
- ▶ Select where list

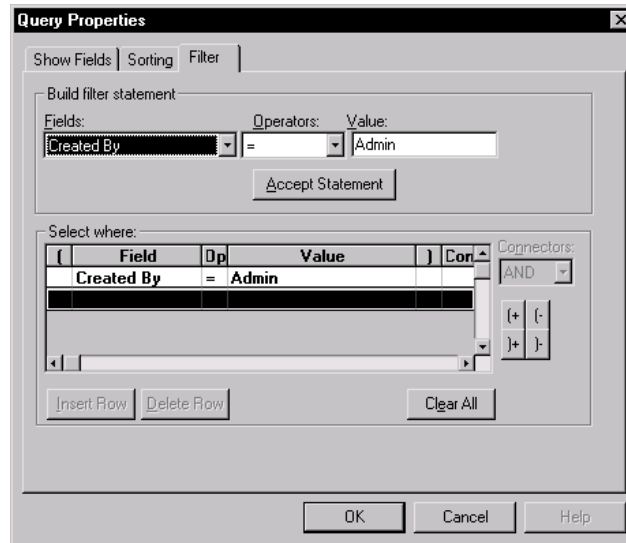
You use the **Build filter statement** to specify the filtering criteria for the query. A sample filter statement for a script could be something like *Created By = Mark*, which means, retrieve all of the scripts created by Mark. You build the filter statement by selecting a field and operator and typing a value. After you define the filter statement, you add it to the **Select where** list.

Each line in the **Select where** list shows one filter statement. You can combine filter statements into complex filters by using the **Connectors** options and the **Parentheses** buttons.

Adding a Simple Filter Statement

To add a simple filter statement:

1. Open the Query Properties dialog box. For details, see *Opening the Query Properties Dialog Box* on page 15-4.
2. Click the **Filter** tab.



3. Select a field from the **Fields** list. If you select one of the Date fields (such as Created on), TestManager inserts Date into the Value field and displays another field into which you can type or select the date.
4. Select an operator from the **Operators** list. The list of available operators varies, depending on the which field you choose. For information about which Operators are available, see the TestManager Help.
5. Specify a filter **Value**. For alphanumeric fields, you simply type the value. For Date fields, you can either type a date, use the spin buttons, or double-click the Date field to display a calendar.
6. Click **Accept Statement** to add the filter statement to the **Select where** list.
7. Repeat steps 3 – 6 to build additional filter statements (if necessary).
8. Click **OK**.

Building Complex Filter Statements

You can create complex filters based on multiple fields and data values in the repository. You do this by building and accepting several filter statements and using connectors to link the individual statements. The two available connectors are **AND** and **OR**. To combine two or more statements, use parentheses.

To create a complex filter:

1. Build at least two filter statements using the procedure in *Adding a Simple Filter Statement* on page 15-6.
2. Highlight the first filter statement in the **Select where** list, and then click the button next to the **Connectors** list.

The **Connectors** list shows the options for linking the filter statements.

3. Select either **AND** or **OR** from the **Connectors** list, or type your choice. The text appears on the line with the filter statement.
4. You can use the parentheses buttons to combine more than two statements.

The following is an example of a complex filtering statement that uses connectors and parentheses:

```
Owner = Eric AND  
(Environment = Windows NT 4.0 OR  
Environment = Windows 2000)
```

This statement retrieves all scripts that are owned by Eric and expected to run on either Windows NT 4.0 or Windows 2000.

Editing Existing Queries

With TestManager, you can edit the properties of a query. You can also copy, rename, and delete queries.

To edit, copy, rename, or delete a query:

1. Click **Tools** → **Manage Queries**.
2. Type or select a query type (script, schedule, or session).
3. Select a query from the list.
4. Click **Edit** to edit the query properties or click **Rename**, **Copy**, or **Delete** to perform one of these actions.
5. Click **OK** twice.

Viewing Query Properties

In addition to using the previous procedure, you can also view a query's properties by doing one of the following:

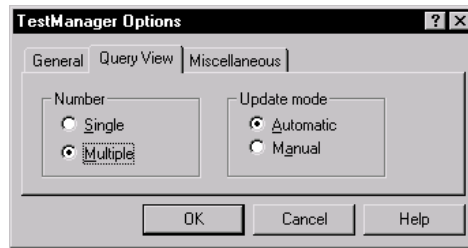
- ▶ Right-click a query in the Asset Browser, and then click **Properties**.
- ▶ Run a query, and then click the Query Properties button.

Setting Query Options

TestManager provides two query options that you can set: Number and Update Mode.

To set query options:

1. Click **Tools** → **Options**.
2. Click the **Query View** tab.



3. In the **Number** area:
 - Click **Single** to indicate that you want only one Query window for each query type (script, schedule, or session) to be open at a time. Running successive queries for a particular type replaces the data in the original window.
 - Click **Multiple** to allow more than one open Query window for each query type.
4. In the **Update mode** area:
 - Click **Manual** to update the Query window only when you click the **Query Update** button.
 - Click **Automatic** to update the Query window automatically each time you run a query or change the way you want the query displayed.

NOTE: If your repository is relatively small, **Automatic** ensures that the Query window always contains the latest data. However, if your repository contains a large amount of data or if you are in a networked environment, automatic updating can affect the performance of your queries.

Configuring the Query Window

After you choose the fields you want to display in the Query window, you can adjust the column widths. Place the pointer over the column separator line in the title, and then click and drag the pointer to set the width.

At any time, you can also:

- ▶ Replace or insert a column.
- ▶ Delete a column.

Replacing or Inserting a Column

To replace one column with another, or to insert a column:

1. Run a query.
2. Right-click the column header that you want to replace, or the header where you want to insert a column.
3. Click **Set Field** or **Insert Field** and select a field from the list.

Deleting a Column

To delete a column:

1. Run a query.
2. Right-click the column header that you want to delete.
3. Click **Delete Field**.

Running TestManager Reports

This chapter explains how to create and run TestManager reports to help you manage your testing efforts. This chapter includes the following topics:

- ▶ Types of reports
- ▶ Selecting which reports to use
- ▶ Working with listing reports
- ▶ Working with coverage reports
- ▶ Working with the Test Results Progress report
- ▶ Copying, renaming, and deleting reports

Types of Reports

TestManager provides three types of reports to help you in your testing efforts:

- ▶ Listing reports
- ▶ Coverage reports
- ▶ Progress reports

In addition, there are reports available in Rational ClearQuest and the LogViewer. ClearQuest reports, as well as report formats, queries, and charts, are available to help you manage your defect database. These reports and other items are created for you automatically when you create a repository that contains an associated ClearQuest database.

For information about using these defect reports, see the ClearQuest Help. For information about creating a repository, see the *Using the Rational Administrator* manual. For information about LogViewer reports, see *Working with Reports* on page 10-13.

Listing Reports

Listing reports display lists of the different items stored in a project. TestManager includes listing reports for scripts, schedules, sessions, builds, users, computers, and test documents. For example, you can create a Script Listing report that:

- ▶ Lists the properties of all of the scripts in your project.
- ▶ Summarizes all of the scripts in your project.
- ▶ Lists or summarizes only the GUI scripts or virtual user scripts.

Coverage Reports

Coverage reports help you track the progress of your test planning, test development, and test execution efforts by showing you whether or not you are validating your test requirements. TestManager includes three types of coverage reports:

Planning Coverage reports facilitate test planning by showing you the percentage and number of test requirements that have been assigned scripts. You can use these scripts to validate your test requirements. Planning Coverage reports can be especially useful during the test planning phase of a project. For information about the planning phase, see Chapter 2, *Planning Your Tests*.

Development Coverage reports facilitate test development by showing you what percentage and number of test requirements have scripts that are ready to be run. These reports show you not only whether you have planned a script for each requirement, but also whether you have actually recorded or edited the script. TestManager uses the **Developed** check box in the Plan Script and Script Properties dialog boxes to determine whether or not a script has been developed. TestManager considers a script to be developed when it has been recorded or edited. For more information about the test development phase, see Chapter 4, *Recording GUI Scripts*.

Execution Coverage reports provide crucial information about the quality of a specific build and the progress of your ability to test that build. These reports tell you the percentage and number of tests that have passed or failed for a specific build. Execution Coverage reports show you not only whether you have planned and developed a script for each requirement, but also whether you have actually run the script and obtained test results. For more information about the test execution phase, see Chapter 9, *Playing Back GUI Scripts*.

Progress Reports

TestManager includes one progress report — the Test Results Progress report. This report lets you track the success rate of your scripts over multiple builds of an application-under-test.

Selecting Which Reports to Use

This section provides some basic guidelines for deciding which report to use to find the information you need from the Rational repository.

When you want to retrieve	Use this report type
A list of some or all of the scripts in your project.	Script Listing
A list of some or all of the schedules in your project.	Schedule Listing
A list of some or all of the sessions in your project.	Session Listing
A list of the builds in your project.	Build Listing
A list of the users in your project. (Users are defined in the Rational Administrator.)	User Listing
A list of the computers in your project. (Computers are defined in the Rational Administrator.)	Computer Listing
A list of the test documents in your project.	Test Document Listing
The percentage of requirements that include planned scripts.	Planning Coverage
The percentage of requirements that have scripts ready for testing.	Development Coverage
The percentage of scripts that have passed or failed for a particular build.	Execution Coverage
The success rate of a set of scripts over a particular set of builds.	Test Results Progress

NOTE: Schedule Listing and Session Listing reports are available only with Rational Suite PerformanceStudio.

Working with Listing Reports

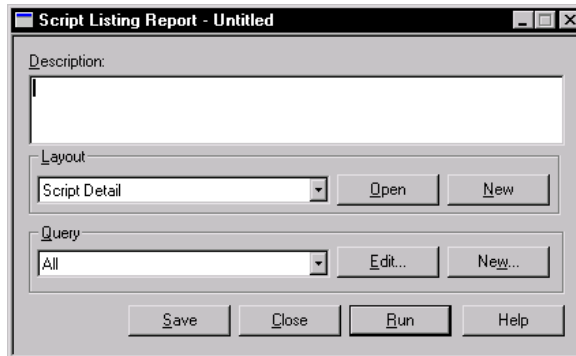
This section includes the following topics:

- ▶ Creating listing reports
- ▶ Running listing reports
- ▶ Opening listing reports

Creating Listing Reports

To create a listing report (in this case, a Script Listing report):

1. Do one of the following:
 - Click **Reports** → **New** → **Script Listing**.
 - Right-click one of the listing report folders in the Asset Browser and click **New Report**.



2. Type a description for the report.

3. Select a report layout from the list. Default layouts include:

Detail – Lists all properties. For example, if you are creating a Script Listing report, a detailed layout displays the script name, description, owner, environment, purpose, and so on.

Summary – Lists just the name, type, and description.

NOTE: You can also create your own report layout with the Report Layout Editor. Click **Open** to open the selected layout, and then edit it as needed. Click **New** to open a blank page that you can use to design a layout from scratch. For more information about the Report Layout Editor, see the TestManager Help.

4. If you are creating a Script, Schedule, or Session Listing report, select a query from the list.

Use the query to narrow down the number of items to be displayed in the report. Optionally, click **Edit** to modify the selected query, or click **New** to create a new query. For more information about queries, see *Creating New Queries* on page 15-4.
5. Click **Save**, type a name (40 characters maximum) for the report, and click **OK**.
6. Click **Close**.

Running Listing Reports

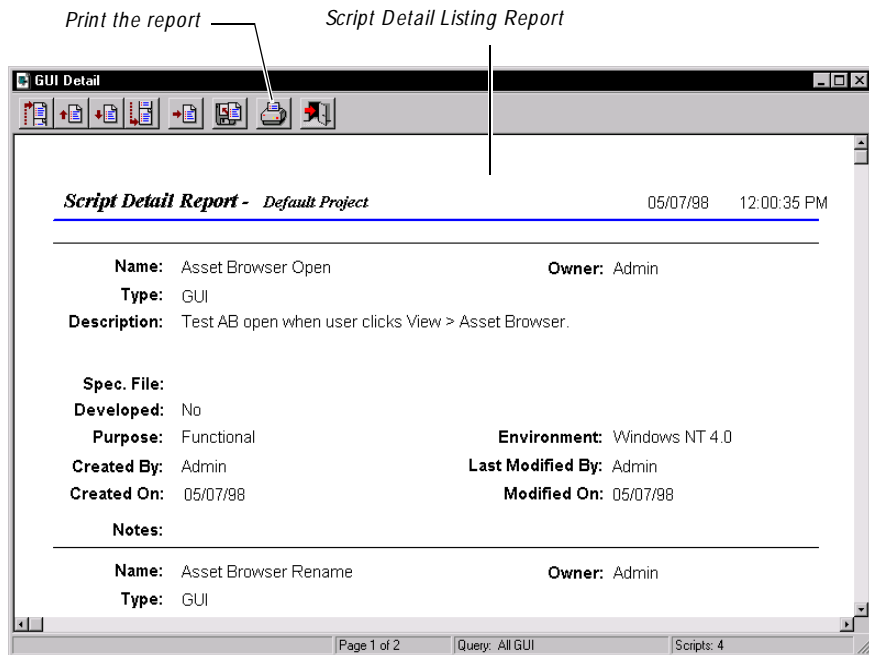
After you create a listing report, you can run it to see the results.

To run a listing report, do one of the following:

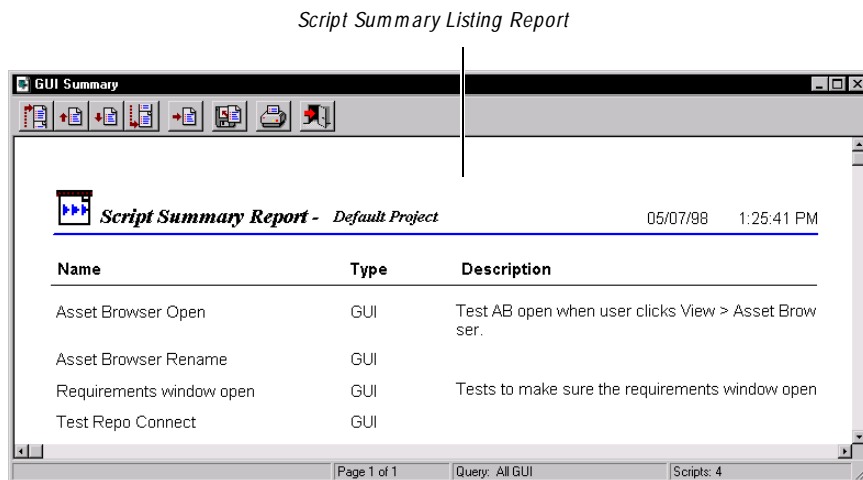
- ▶ Click **Reports** → **Run**, select a report from the list, and click **OK**.
- ▶ Right-click a listing report in the Asset Browser and click **Run**.

Running TestManager Reports

The following figure show a Script Detail Listing report:



The following figure show a Script Summary Listing report:



Opening Listing Reports

At any point after creating a report, you can open it and, if necessary, make changes to the report definition.

To open a listing report, do one of the following:

- ▶ Click **Reports** → **Open**, select a report from the list, and click **OK**.
- ▶ Right-click a listing report in the Asset Browser and click **Open**.

Working with Coverage Reports

This section includes the following topics:

- ▶ Creating coverage reports
- ▶ Running Planning and Development Coverage reports
- ▶ Running Execution Coverage reports
- ▶ Opening coverage reports

Creating Coverage Reports

With TestManager, you can create three types of coverage reports:

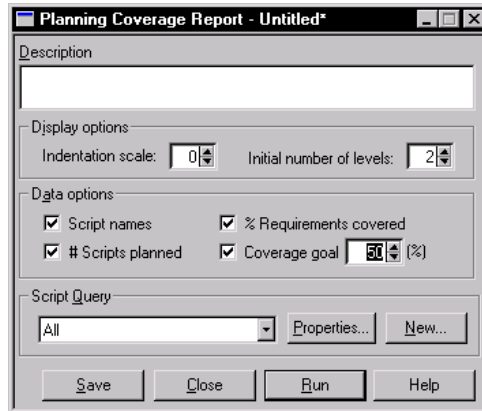
- ▶ **Planning Coverage** reports show how many requirements you have planned scripts for.
- ▶ **Development Coverage** reports show how many requirements you have actually developed scripts for.
- ▶ **Execution Coverage** reports show how many requirements you have actually developed and run scripts for.

The following sections explain how to create each type of report.

Creating Planning Coverage Reports

To create a Planning Coverage report:

1. Do one of the following:
 - Click **Reports** → **New** → **Planning Coverage**.
 - Right-click the **Planning Coverage** folder in the Asset Browser and click **New Report**.



2. Type a description for the report.
3. Set the **Display options** and the **Data options**.
4. Select a script query from the list. Use the query to narrow down the number of items to be displayed in the report.

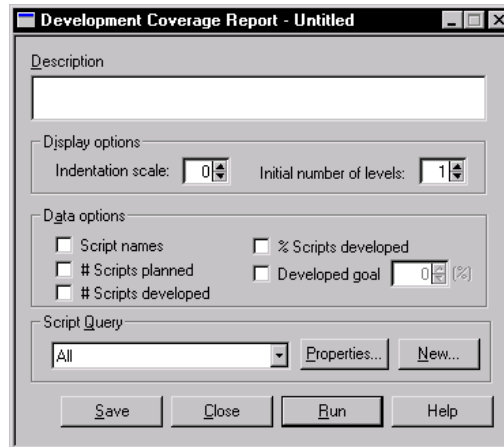
Optionally, click **Properties** to modify the query, or click **New** to create a new query. (For information about queries, see *Creating New Queries* on page 15-4.)

5. Click **Save**, type a name for the report, and click **OK**.
6. Click **Close** to close the Planning Coverage Report specification window.

Creating Development Coverage Reports

To create a Development Coverage report:

1. Do one of the following:
 - Click **Reports** → **New** → **Development Coverage**.
 - Right-click the Development Coverage folder in the Asset Browser and click **New Report**.



2. Type a description for the report.
3. Set the **Display options** and the **Data options**.
4. Select a script query from the list. Use the query to narrow down the number of items to be displayed in the report.

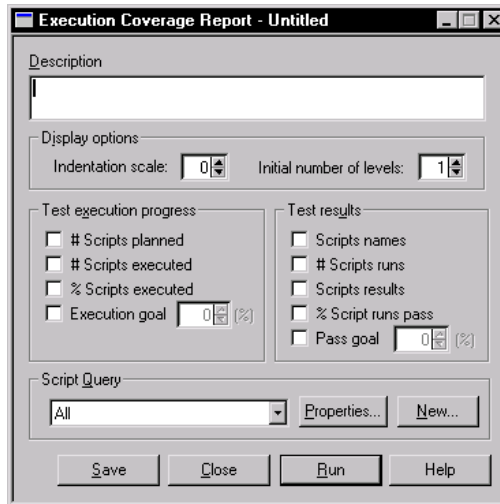
Optionally, click **Properties** to modify the query, or click **New** to create a new query. (For information about queries, see *Creating New Queries* on page 15-4.)

5. Click **Save**, type a name for the report, and click **OK**.
6. Click **Close** to close the Development Coverage Report specification window.

Creating Execution Coverage Reports

To create an Execution Coverage report:

1. Do one of the following:
 - Click **Reports** → **New** → **Execution Coverage**.
 - Right-click the Execution Coverage folder in the Asset Browser and click **New Report**.



2. Type a description for the report.
3. Set the **Display options**, the **Test execution progress** options, and the **Test results** options.
4. Select a script query from the list. Use the query to narrow down the number of items to be displayed in the report.

Optionally, click **Properties** to modify the query, or click **New** to create a new query. (For information about queries, see *Creating New Queries* on page 15-4.)

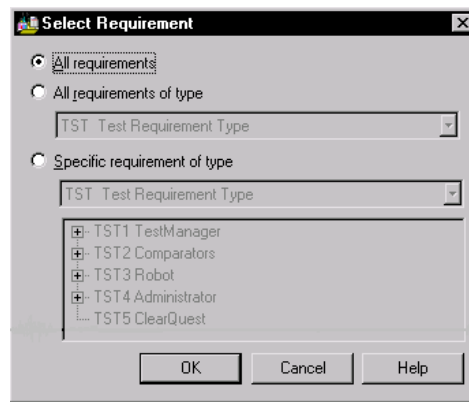
5. Click **Save**, type a name for the report, and click **OK**.
6. Click **Close** to close the Execution Coverage Report specification window.

Running Planning and Development Coverage Reports

After you create a Planning or Development Coverage report, you can run it to see the results.

To run a Planning Coverage or Development Coverage report:

1. Do one of the following:
 - Click **Reports** → **Run**, select a Planning Coverage or Development Coverage report from the list, and click **OK**.
 - Right-click a Planning Coverage or Development Coverage report in the Asset Browser and click **Run**.



2. Specify the requirements you want to include.

All requirements – Includes all requirements, regardless of type, in the Requirements Hierarchy.

All requirements of type – Includes all requirements of the selected requirement type.

Specific requirement of type – Includes a single branch of the Requirements Hierarchy for a specific requirement type. In this case, a branch is a single parent requirement and its children.

3. Click **OK** to run the report.

Running TestManager Reports

The following figures show a Planning Coverage report and a Development Coverage Report:

Copy to Clipboard Save report to disk Close report window

Print the report

Requirements	Script Name	# Scripts Planned	% Reqs Covered	At (50%) Coverage Goal?
- TST1 TestManager		2	40	NO
- TST1.1 MDI Windows		0	0	NO
TST1.1.1 Main		0	0	NO
TST1.1.2 Query window		0	0	NO
TST1.1.5 Report window		0	0	NO
- TST1.2 Asset Browser		1	100	YES
	Asset Browser Open			
- TST1.3 Requirements Hierarchy		1	100	YES
	Requirements window			
- TST2 Comparators		0	0	NO
TST2.1 Grid		0	0	NO
TST2.2 Text		0	0	NO
- TST3 Robot		0	0	NO
TST3.1 Recording		0	0	NO
TST3.1.1 SQABasic		0	0	NO
- TST4 Administrator		1	100	YES
- TST4.1 Connect to Repository		1	100	YES
	Test Repo Connect			
TST5 ClearQuest		0	0	NO

Query: All GUI

Requirements	Script Name	Is Script Recorded	# Scripts Planned	# Scripts Recorded	% Scripts Recorded	At (50%) Scripted Goal?
- TST1 TestManager			2	0	0	NO
- TST1.1 MDI Windows			0	0	0	
TST1.1.1 Main			0	0	0	
TST1.1.2 Query window			0	0	0	
TST1.1.5 Report window			0	0	0	
- TST1.2 Asset Browser			1	0	0	NO
	Asset Browser Open	NO				
- TST1.3 Requirements Hierarchy			1	0	0	NO
	Requirements window open	NO				
- TST2 Comparators			0	0	0	
TST2.1 Grid			0	0	0	
TST2.2 Text			0	0	0	
- TST3 Robot			0	0	0	
TST3.1 Recording			0	0	0	
TST3.1.1 SQABasic			0	0	0	
- TST4 Administrator			1	0	0	NO
- TST4.1 Connect to Repository			1	0	0	NO
	Test Repo Connect	NO				
TST5 ClearQuest			0	0	0	

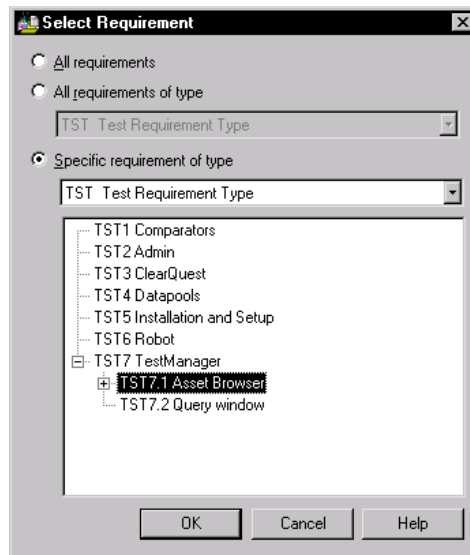
Query: All GUI

Running Execution Coverage Reports

After you create an Execution Coverage report, you can run it to see the results. With Execution Coverage reports, you specify the requirements to report on, as well as the set of logs to examine. (For information about logs, see *Generating Log Files* on page 3-2.)

To run an Execution Coverage report:

1. Do one of the following:
 - Click **Reports** → **Run**, select an Execution Coverage report from the list, and click **OK**.
 - Right-click an Execution Coverage report in the Asset Browser and click **Run**.



2. Set the requirements you want to include.

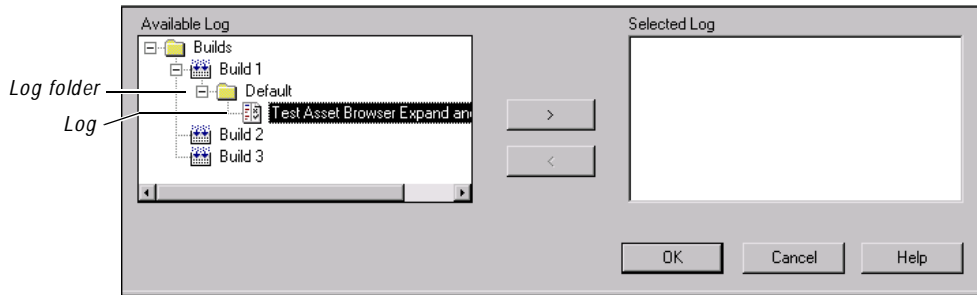
All requirements – Includes all requirements, regardless of type, in the Requirements Hierarchy.

All requirements of type – Includes all requirements of the selected requirement type.

Specific requirement of type – Includes a single branch of the Requirements Hierarchy for a specific requirement type. In this case, a branch is a single parent requirement and its children.

Running TestManager Reports

3. Select a log from a build. Click > to move the log to the **Selected Log** list.
Repeat as necessary.



4. Click **OK** to run the report.

The following figure shows an Execution Coverage report for a requirement called *TST7.1 Asset Browser*.

The screenshot shows a window titled 'Test Execution Coverage Report - Results against 50% pass goal'. The window contains a table with the following data:

Requirements	# Scripts Planned	# Scripts Executed	% Scripts Executed	At (50%) Executed Goal?	Script Name	# Script Runs	# Script Runs Passed	# Script Runs Failed	% Procs Runs Pass	At (50%) Pass Goal?
- TST7.1 Asset Browser	3	0	0	NO		0	0	0	0	
+ TST7.1.1 Expand/collapse	1	0	0	NO		0	0	0	0	
+ TST7.1.2 Close	1	0	0	NO		0	0	0	0	
+ TST7.1.3 Open	1	0	0	NO		0	0	0	0	

At the bottom of the window, there is a status bar that reads 'Query: All GUI'.

Opening Coverage Reports

At any point after creating a report, you can open it and, if necessary, make changes to the report definition.

To open a coverage report, do one of the following:

- ▶ Click **Reports** → **Open**, select a report from the list, and click **OK**.
- ▶ Right-click a coverage report in the Asset Browser and click **Open**.

Working with the Test Results Progress Report

This section includes the following topics:

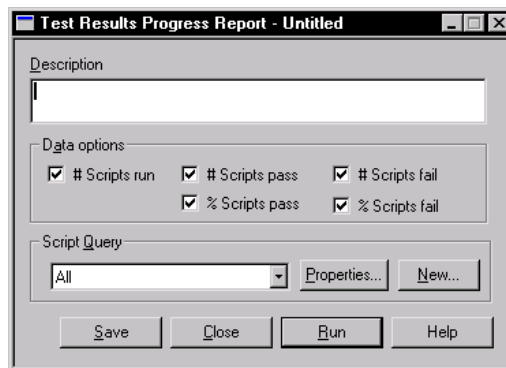
- ▶ Creating a Test Results Progress report
- ▶ Running a Test Results Progress report
- ▶ Opening a Test Results Progress report

Creating a Test Results Progress Report

With the Test Results Progress report, you can track the success rate of your scripts from build to build. When you create the report, you specify the scripts that you want included. Later, when you run the report, you specify the builds you want to run the report against.

To create a Test Results Progress report:

1. Do one of the following:
 - Click **Reports** → **New** → **Test Results Progress**.
 - Right-click the Test Results Progress folder in the Asset Browser and click **New Report**.



2. Type a description for the report.

Running TestManager Reports

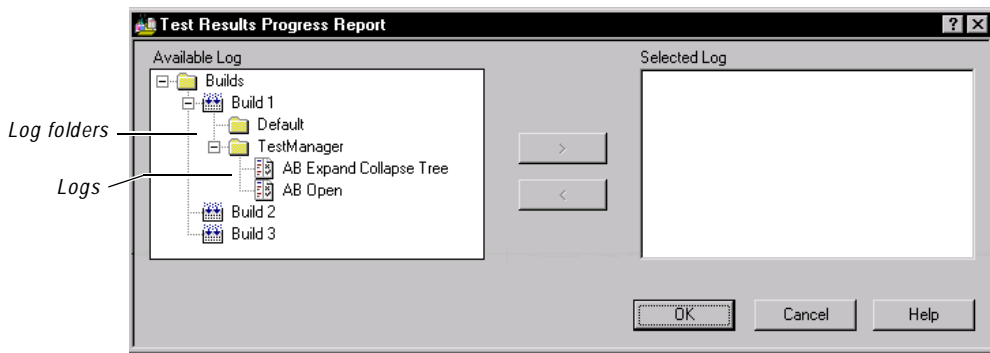
3. Set the **Data options**.
4. Select a script query from the list. Use the query to narrow down the number of items to be displayed in the report.
Optionally, click **Properties** to modify the query, or click **New** to create a new query. (For information about queries, see *Creating New Queries* on page 15-4.)
5. Click **Save**, type a name for the report, and click **OK**.
6. Click **Close** to close the Test Results Progress Report specification window.

Running a Test Results Progress Report

After you create a Test Results Progress report, you can run it to see the results. With a Test Results Progress report, you need to indicate the set of logs to examine. (For information about logs, see *Generating Log Files* on page 3-2.)

To run a Test Results Progress report:

1. Do one of the following:
 - Click **Reports** → **Run**, select a Test Results Progress report from the list, and click **OK**.
 - Right-click a Test Results Progress report in the Asset Browser and click **Run**.



2. Select a log from the list of available logs. Click > to move the log to the **Selected Log** list. Repeat as necessary.
3. Click **OK** to run the report.

The following figure shows a Test Results Progress report:

Build	Total Runs	Pass	% Passed
Build 1	2	2	100

Opening a Test Results Progress Report

At any point after creating a report, you can open it and, if necessary, make changes to the report definition.

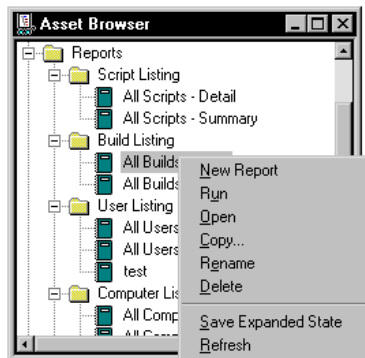
To open a Test Results Progress report, do one of the following:

- ▶ Click **Reports** → **Open**, select a report from the list, and click **OK**.
- ▶ Right-click a Test Results Progress report in the Asset Browser and click **Open**.

Copying, Renaming, and Deleting Reports

To copy, rename, or delete a report from the Asset Browser:

1. Right-click a report in the Asset Browser.



2. Do one of the following:
 - Click **Copy**, type a name for the new report, and click **OK**.
 - Click **Delete** and then click **Yes** to delete the report.
 - Click **Rename**, type the new name for the report, and press **ENTER**.

Running TestManager Reports

▶ ▶ ▶ Part VI

Testing IDE Applications

Testing Visual Basic Applications

This chapter explains how to test 32-bit Visual Basic applications with Rational Robot. It includes the following topics:

- ▶ About Robot support for Visual Basic applications
- ▶ Try it! with Visual Basic
- ▶ Verifying that the Visual Basic extension is loaded

About Robot Support for Visual Basic Applications

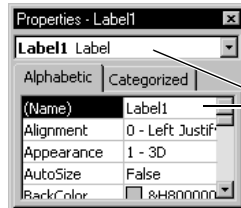
Rational Robot provides comprehensive support for testing 32-bit applications built with Visual Basic version 4.0 and later. Robot supports the testing of applications that you migrate from one Visual Basic version to another, and allows for the reuse of scripts between Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

With its Object Testing technology, Robot examines data and properties that are not visible to the user. This means that Robot can do the following:

- ▶ Recognize all Visual Basic objects, including objects that have windows associated with them (such as EditBoxes) and objects that are “painted” on the containing form (such as Labels).
- ▶ Determine the names of objects in your program (as given in the Visual Basic source code), and use those names for object recognition.
- ▶ Capture properties of Visual Basic objects, using the Object Properties verification point.
- ▶ Capture the data underlying a Visual Basic data control, using the Object Data verification point.

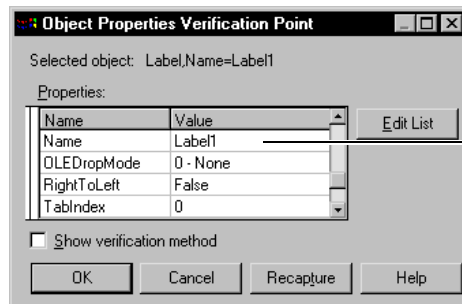
Testing Visual Basic Applications

As an example, suppose you have a label in a Visual Basic form. If you click the label during Robot recording, the label's name appears in the Robot script. If you create an Object Properties verification point on the label, the label's name is captured. The name by which Robot identifies the object is the same as its Visual Basic name, as shown in the Visual Basic Properties window.



Label Click, "Name=Label1"

... is the same name that appears in the Robot script when you click the object ...

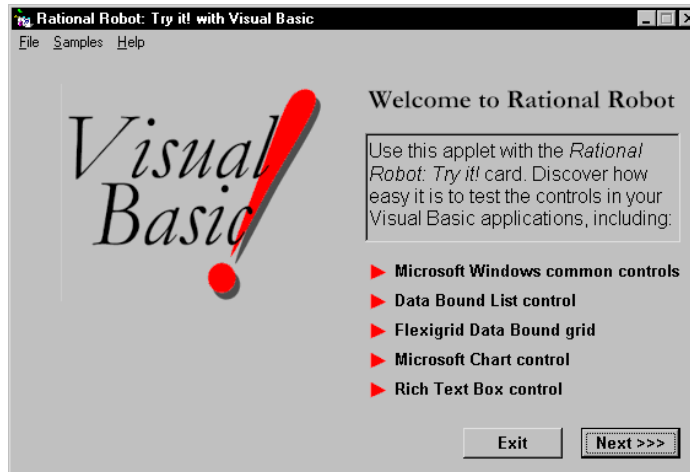


NOTE: To test Visual Basic 4.0 applications, you need to add the Rational ActiveX Test Control to your Visual Basic forms. For information, see *Visual Basic support, making Visual Basic 4.0 applications testable* in the Robot Help Index.

Try it! with Visual Basic

Robot comes with the *Try it! with Visual Basic* card and sample applet.

The *Try it!* card provides quick instructions for recording tests on objects in the Visual Basic applet. The applet contains objects that are specific to the Visual Basic development environment. For example, you can test the properties and data of an ActiveX control and a hidden data control.



Verifying that the Visual Basic Extension Is Loaded

To test Visual Basic applications, you should first verify that the Robot Visual Basic extension is loaded in Robot.

To verify that the extension is loaded:

1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **Visual Basic** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Testing Oracle Forms Applications

This chapter describes how to test 32-bit Oracle Forms applications with Rational Robot. It includes the following topics:

- ▶ About Robot support for Oracle Forms applications
- ▶ Try it! with Oracle Forms
- ▶ Making Oracle Forms applications testable
- ▶ Recording actions and testing objects
- ▶ Testing an object's properties
- ▶ Testing an object's data

About Robot Support for Oracle Forms Applications

Rational Robot provides comprehensive support for testing 32-bit applications built with Oracle Forms 4.5 and 5.0. Robot supports the testing of applications that you migrate from one Oracle Forms version to another, and allows for the reuse of scripts between Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

With its Object Testing technology, Robot examines data and properties that are not visible to the user. Robot uses Object-Oriented Recording to recognize an Oracle Forms object by its internal name.

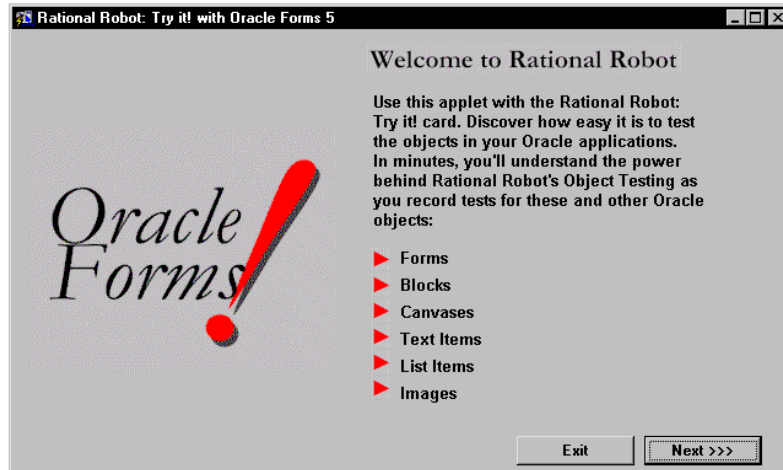
You can use Robot to test Oracle Forms objects, including:

- ▶ Windows
- ▶ Forms
- ▶ Canvas-views
- ▶ Base-table blocks (single- and multi-record)
- ▶ Items, including OLE containers

Try it! with Oracle Forms

Robot comes with the *Try it! with Oracle Forms* card and sample applet.

The *Try it!* card provides quick instructions for recording tests on objects in the Oracle Forms applet. The applet contains objects that are specific to the Oracle Forms development environment. For example, you can test the properties and data of a base-table block (a block associated with a database table).



NOTE: Before you can install and run the Oracle Forms sample applet, the following must be installed: Oracle Forms 5.0 or 4.5 and the Oracle Open Client Adapter for ODBC.

Making Oracle Forms Applications Testable

Before you can test your Oracle Forms applications, you must:

- ▶ Install the Rational Test Oracle Forms Enabler.
- ▶ Run the Enabler on your application.
- ▶ Verify that the Robot Oracle Forms extension is loaded.

Installing the Rational Test Oracle Forms Enabler

You can install the Enabler from the Rational Software Setup wizard. For instructions, see one of the following manuals: *Installing Rational Suite* or *Installing Rational TeamTest and Rational Robot*.

You then need to run the Enabler, which instruments the code in your .fmb file to make your application testable by Robot.

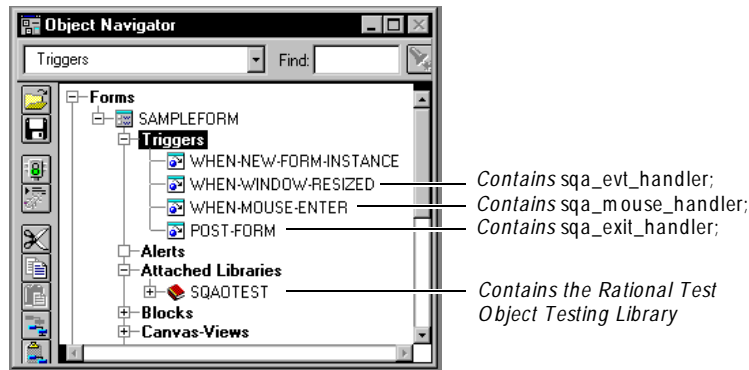
Running the Enabler on Your Application

After you install the Enabler, you need to enable every form in your application. The following sections explain:

- ▶ What happens when you run the Enabler.
- ▶ How to run the Enabler.
- ▶ What to distribute with your application.

What Happens When You Run the Enabler?

The Enabler adds the Rational Test Object Testing Library and three triggers to one or all .fmb files in a directory, as shown in the following figure and table.



As shown in the previous figure, the triggers contain the following code:

Trigger	Code
WHEN - WINDOW - RESIZED	sqa_evt_handler;
WHEN - MOUSE - ENTER	sqa_mouse_handler;
POST - FORM	sqa_exit_handler;

The Enabler handles the triggers and code as follows:

If the .fmb file	The Enabler
Does not contain the trigger	Adds a new trigger containing the code.
Contains the trigger	Prepends the code to your existing trigger.
Contains a reference to the trigger	Does not change the trigger in either the selected (referencing) .fmb file or in the referenced .fmb file. Displays a message indicating that you should run the Enabler on the referenced .fmb file.

NOTE: If objects in your application contain the `WHEN-MOUSE-ENTER` trigger, the Enabler prepends `sqa_mouse_handler;` to each trigger. This is necessary for Robot to correctly record mouse actions against these objects. If you need to prevent this modification, you can clear the **Modify local WHEN-MOUSE-ENTER triggers** option in the Enabler. (See the next section, *Running the Enabler*.) In this case, the Enabler displays warning messages in the Status window when it detects any of these local triggers.

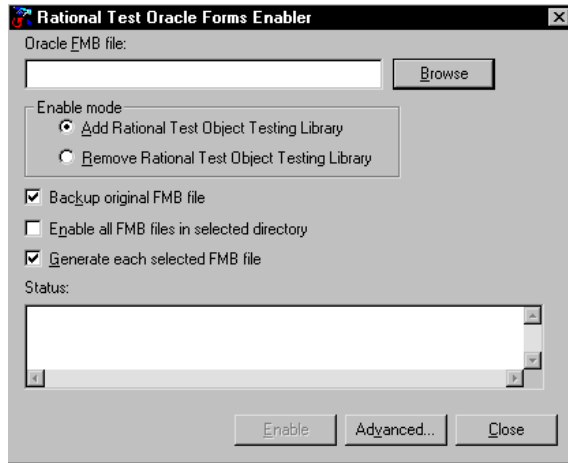
You can leave the triggers and the Object Testing Library in your application when you distribute it. For more information, see *Distributing Your Application on page 18-8*.

Running the Enabler

To run the Enabler, which adds the Object Testing Library and triggers to your application:

1. Start the **Rational Test Oracle Forms Enabler** from the folder in which it was installed (the default folder is Developer 2000).

The following dialog box appears:



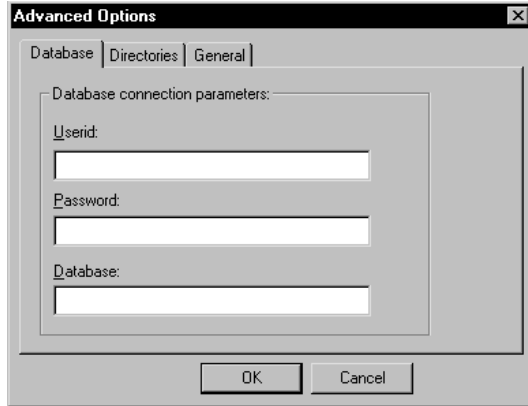
2. Click **Browse**. Select the .fmb file that you want to make testable and click **OK**.
3. Click **Add Rational Test Object Testing Library**.
4. Set the following options as needed:

Backup original FMB file – Creates a backup file before the file is enabled.

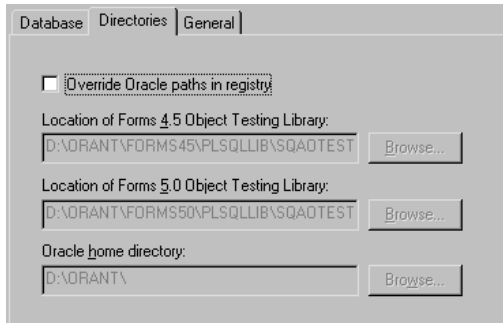
Enable all FMB files in selected directory – Enables every .fmb file in the directory. If this check box is not selected, only the .fmb file in the **Oracle FMB file** box is enabled.

Generate each selected FMB file – Generates each .fmb file after enabling it. If this check box is not selected, you will need to generate each .fmb file from the Oracle Forms 5.0 Builder or Oracle Forms 4.5 Designer after the Enabler runs.

5. Click **Advanced** to open the following dialog box:

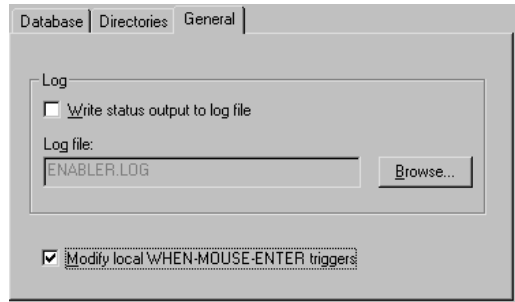


6. If you selected the **Generate each selected FMB file** option, type your database connection parameters in the **Database** tab.
7. Click the **Directories** tab.



8. If you need to change the default locations of the Object Testing Library and Oracle home directory, select **Override Oracle paths in registry**. Click each **Browse** button and select the new location.

9. Click the **General** tab.



10. To send the output in the **Status** box of the Enabler to a log file that you can view or print, select **Write status output to log file**.

In the **Log file** box, use the default log file name, type a new name, or click **Browse** and select a file.

The log file is stored in the same directory as the .fmb file unless you specify another path. If the file already exists, the text is appended to the file.

11. If objects in your application contain the `WHEN-MOUSE-ENTER` trigger, the Enabler prepends `sga_mouse_handler;` to each trigger. This is necessary for Robot to correctly record mouse actions against these objects. If you need to prevent this modification, clear **Modify local WHEN-MOUSE-ENTER triggers**. (For information about the Enabler and triggers, see *What Happens When You Run the Enabler?* on page 18-3.)

If this check box is cleared, the Enabler displays warning messages in the **Status** box when it detects any of these local triggers.

12. Click **OK**.

13. Click **Enable**. As the file is enabled, information appears in the **Status** box.

14. If you did not select the **Generate** option in step 4, regenerate your application once before using Robot by doing one of the following:

- In Oracle Forms 5.0, open the Forms Builder. Load each enabled .fmb file, and click **File** → **Administration** → **Compile File**.
- In Oracle Forms 4.5, open the Forms Designer. Load each enabled .fmb file, and click **File** → **Administration** → **Generate**.

You are now ready to use Robot with your Oracle Forms application.

Distributing Your Application

The triggers and the Object Testing Library are not visible and are non-intrusive, and there are no license restrictions on them. Therefore, you can leave them in the application when you distribute it. In this case, you must distribute the Rational Test Object Testing Library with your application.

You may freely distribute the Object Testing Library for Oracle Forms and the Enabler. The Rational Test Enablers directory on the CD-ROM contains a Setup wizard for these and other items. You may copy this directory to a network drive or to a diskette for distribution to your developers.

NOTE: If you choose to remove the Object Testing Library and triggers from your application, follow the procedure in *Running the Enabler on page 18-4*. In step 3, click **Remove Rational Test Object Testing Library**.

Verifying that the Oracle Forms Extension Is Loaded

To test Oracle Forms applications, you should first verify that the Robot Oracle Forms extension is loaded in Robot.

To verify that the extension is loaded:

1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **Oracle Forms** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Recording Actions and Testing Objects

An Oracle Forms application is made up of visual and nonvisual objects.

- ▶ **Visual objects** are GUI objects that you can see in the application. Examples are check boxes and push buttons.
- ▶ **Nonvisual objects** are non-GUI objects that you cannot see in the application. Examples are blocks and forms.

You can record actions against visual objects, and you can test both visual and nonvisual objects.

Recording Actions

When you record actions against a visual Oracle object, Robot recognizes the object by its internal name as follows:

- ▶ **Window** – Recognized by the *window* internal name assigned by the developer.
- ▶ **Item** – Recognized by the *block.item* name assigned by the developer.

For example, if you click a button within a window, the script appears as follows:

```
Window SetContext, "Name=SAMPLE_WINDOW_TWO;ChildWindow", ""
PushButton Click, "Name=DATA_CONTROLS.GO_CUSTOMER"
```

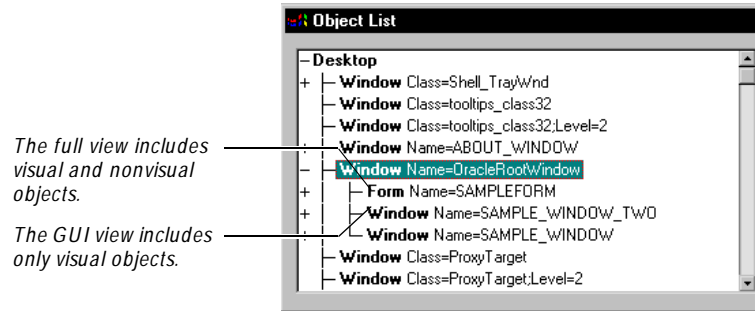
Testing Objects

When you test a visual or nonvisual Oracle object, Robot can provide two views into the Oracle application:

- ▶ **Full View** – Includes all objects (visual and nonvisual) in the application. (This is similar to the Ownership View in the Oracle Forms Navigator.) In this view, items are children of blocks, which are children of a form. This view also includes canvas-views and windows. When you select an object from the full view, the object is identified by its complete path in the script.
- ▶ **GUI View** – Includes only the visual (GUI) objects in the application. (This is similar to the Visual View in the Oracle Forms Navigator.) In this view, all objects are children of a window. When you select an object from the GUI view, the object is identified by its *block.item* name relative to the window in the script.

Testing Oracle Forms Applications

The following figure shows the full view and the GUI view (collapsed):



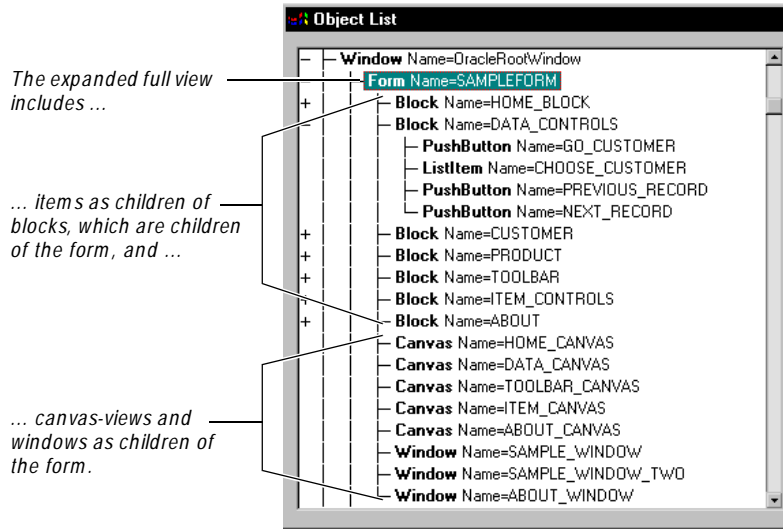
To see both Robot views of an Oracle application:

1. Start to create a verification point.
2. In the Select Object dialog box, click **Browse** to open the Object List.

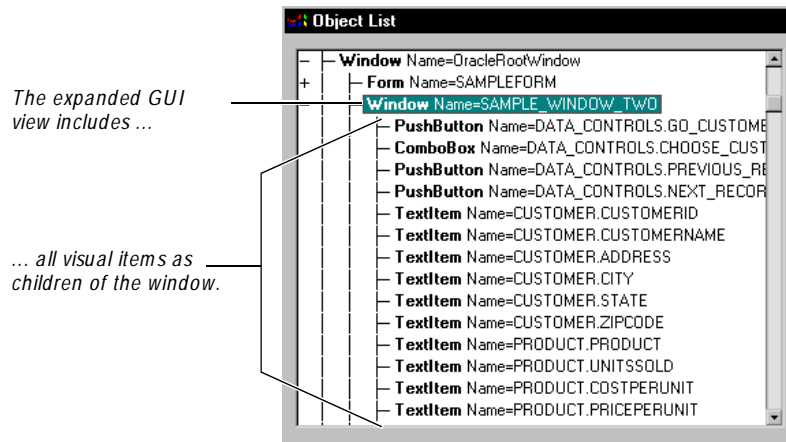
There are two types of branches under the **OracleRootWindow** branch:

- ▶ The **Form** branch gives you a full view of all objects (visual and nonvisual) in the application. (If you have multiple forms, Robot displays a full view for the active form only.)
- ▶ The **Window** branches give you a GUI view of only the visual objects in the application.

The following figure shows the expanded full view of the Form branch:



The following figure shows the expanded GUI view of the Window branch:



Testing an Object's Properties

You can use two methods to test the properties of an Oracle object:

- ▶ **Object Properties verification point** – Use to test properties while recording or editing a script.
- ▶ **Object Scripting commands** – Use to test properties programmatically while editing a script.

NOTE: Before you can test an Oracle Forms application, you need to run the Enabler. For instructions, see *Making Oracle Forms Applications Testable on page 18-2*.

Object Properties Verification Point

You can use the Object Properties verification point to test any property that you can access in PL/SQL (the Oracle programming language) for the following objects:

- ▶ Visual (GUI) objects:

Chart item	Push Button item
Check Box item	Radio Button item
Display item	Radio Group item
Image item	Text item
List item	Oracle VBX Control item
OLE Container item	Window

- ▶ Nonvisual (non-GUI) objects:

Block
Canvas-view
Form

NOTE: To test the properties of list of values (LOV) and record group objects, see *Object Scripting Commands on page 18-15* and *Testing LOVs and Record Groups on page 18-17*.

Testing Properties of Visual Objects

To test the properties of a visual (GUI) object:

1. Start creating a Object Properties verification point as usual. (For instructions, see *Object Properties Verification Point* in the Robot Help Index.)
2. When the Select Object dialog box appears, drag the Object Finder tool to the object to test and release the mouse button. If the Select Object dialog box appears again, click **OK**.

If you point to the title bar of a window (other than the Developer/2000 Forms Runtime window), Robot captures the properties of *all* of the visual objects in the window.

If you point to the title bar of the Developer/2000 Forms Runtime window, Robot captures the properties of *all* of the visual and nonvisual objects in the application. (For more information, see the next section, *Testing Properties of Nonvisual Objects*.)

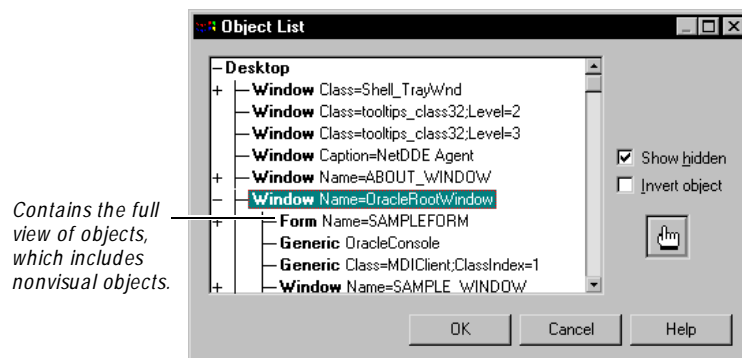
3. Complete the verification point as usual.

Testing Properties of Nonvisual Objects

To test the properties of a nonvisual object:

1. Start creating a Object Properties verification point as usual. (For instructions, see *Object Properties Verification Point* in the Robot Help Index.)
2. In the Select Object dialog box, click **Browse** to display the Object List. This is a list of all objects on the desktop.
3. Expand the **Window Name= OracleRootWindow** branch by double-clicking the plus sign.

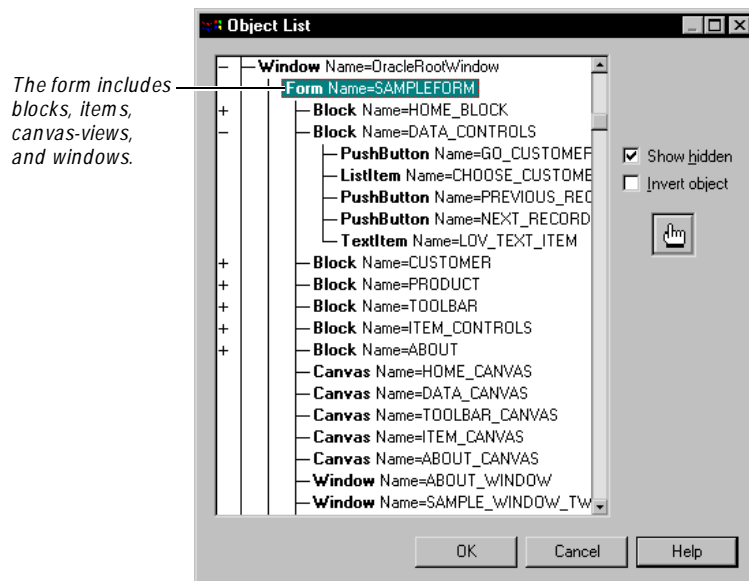
The nonvisual objects are contained in the **Form** branch of the **OracleRootWindow**.



NOTE: To capture the properties of *all* of the objects, you can drag the Object Finder tool to the Developer/2000 Forms Runtime window and release the mouse button (instead of clicking **Browse**). Using **Browse** is usually faster because you can select only the objects that you want to test from the Object List *before* the properties are captured.

- Expand the **Form** branch by double-clicking the plus sign.

All of the blocks, canvas-views, and windows appear as children of the form. Items appear as children of a block. (For information about working with the Object List, see *Selecting the Object to Test on page 6-10*.)



- Select the object to test and click **OK**. If the Select Object dialog box still appears, click **OK**.
- Complete the verification point as usual.

NOTE: Visual items within the full view are children of blocks. If you select a visual item from the full view, Robot tests its Oracle properties only. If you select a visual item from the GUI view, Robot tests both its Oracle properties and its standard properties. For a description of the two views, see *Testing Objects on page 18-9*.

Object Scripting Commands

You can manually add the Object Scripting commands to any script to access the properties of Oracle Forms objects. For example, you can use the `SQAGetProperty` command to retrieve the value of a specified property. (For information about the Object Scripting commands, see the *SQABasic Language Reference*.)

The Object Scripting commands are especially useful for accessing the properties of LOV and record group objects, which cannot easily be tested with the Object Properties verification point.

NOTE: For instructions about testing LOVs and record groups with verification points, see *Testing LOVs and Record Groups on page 18-17*.

To reference an LOV or record group object in an Object Scripting command, you need to know the name assigned to the object within your Oracle Forms application. Once you know the object name, you can access the following properties:

Object Type	Properties
List of values (LOV)	X_Pos Y_Pos Auto_Refresh Group_Name Width Height
Record group	Row_Count Selection_Count

The following example uses the `SQAGetProperty` command to assign the value of the `Group_Name` property of an LOV object to a variable called `Value`:

```
Sub Main
  Dim Result As Integer
  Dim Value As Variant
  Window SetTestContext, "Name=OracleRootWindow", ""
  Result = SQAGetProperty("Type=Form;Name=FORM_NAME;\;
    Type=LOV;Name=LOV_NAME", "Group_Name", Value)
  MsgBox Value
End Sub
```

Testing an Object's Data

You can use the Object Data verification point to test the data in the following Oracle objects:

- ▶ Base-table blocks and items
- ▶ LOV and record group objects

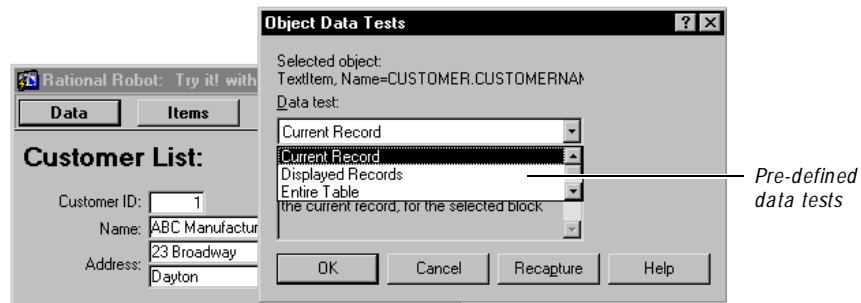
NOTE: Before you can test an Oracle Forms application, you need to run the Enabler. For instructions, see *Making Oracle Forms Applications Testable on page 18-2*.

For instructions about testing an object's data, see *Object Data Verification Point* in the Robot Help Index.

Testing Base-Table Blocks and Base-Table Items

Several pre-defined data tests are supplied with Robot to test any base-table block or base-table item. These tests include:

- ▶ **Current Record** – Captures the currently selected record.
- ▶ **Displayed Records** – Captures the currently displayed records.
- ▶ **Entire Table** – Captures the entire contents of the database table associated with the object.



You can use the Object Data Test Definition to define additional data tests. (For information, see Appendix B, *Working with Data Tests*.)

Testing LOVs and Record Groups

You can use the Object Data verification point to test the data in list of values (LOV) and record group objects in Oracle Forms. (You can also use the Object Scripting commands. For information, see *Object Scripting Commands on page 18-15*.)

To test an LOV or record group, you need to perform two steps:

1. Create an .sqa text file containing information about the LOV or record group.
2. Create the Object Data verification point.

NOTE: Once you create the .sqa file, you can also test the properties of LOVs and record groups using the Object Properties verification point.

Creating an .SQA Text File

Before you test an LOV or record group, you need to create an .sqa text file. To create this file:

1. Before creating the verification point, identify which LOV or record group objects to test, including:
 - The Form containing the LOV or record group.
 - The internal Oracle name of the LOV or record group.
 - The names of the data columns and each column's data type (char, number, or date).

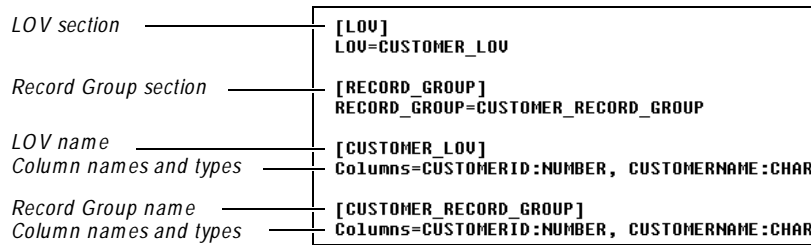
You may need to get this information from the developer of the form.

2. In the same directory as the form's executable (.fmx) file, create a text file with the same name as the .fmx file, but with an **.sqa** extension.

For example, if the form's executable file is Oraapp32.fmx, create a text file named Oraapp32.sqa.

3. In the .sqa text file, type:
 - An [LOV] section containing the names of all LOVs to test.
 - A [RECORD_GROUP] section containing the names of all record groups to test.
 - The name of each LOV, the name of each column, and the data type of each column.
 - The name of each record group, the name of each column, and the data type of each column.

The following figure shows an example.



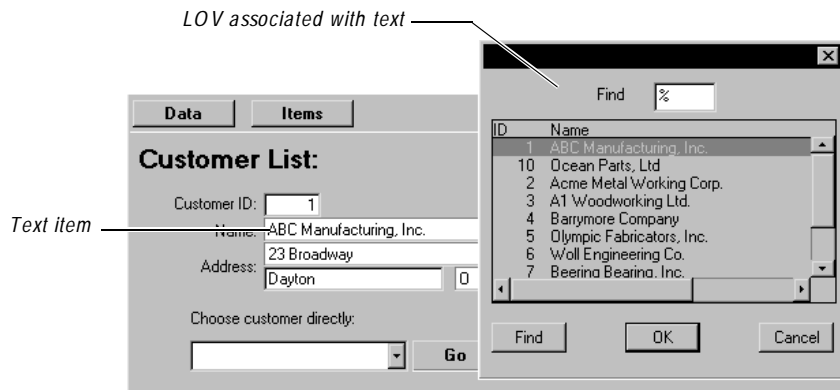
After you create the .sqa file, you can capture the data in:

- ▶ An LOV associated with a text item.
- ▶ Any LOV or record group.

Once you create the .sqa file, you can also test the properties of LOVs and record groups using the Object Properties verification point.

Capturing Data in an LOV Associated with a Text Item

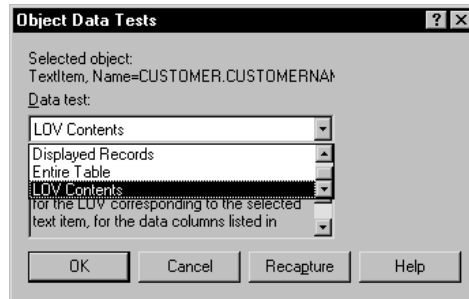
If an LOV is associated with a text item, you can point to the text item to capture the data in the LOV.



To capture the data:

1. Display the form containing the LOV.
2. Make sure the LOV is closed. You can capture the data in an LOV only when the LOV is closed.

3. Start creating an Object Data verification point.
4. When the Select Object dialog box appears, drag the Object Finder tool to the text item and release the mouse button. If the Select Object dialog box still appears, click **OK**. The Object Data Tests dialog box appears.
5. From the **Data test** list, select **LOV Contents**.



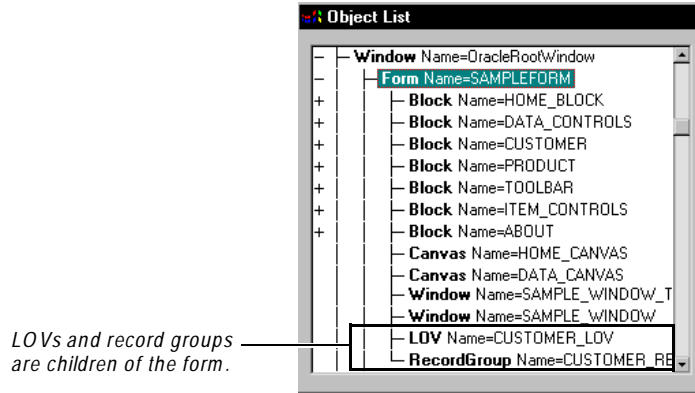
6. Click **OK** to open the Object Data Verification Point dialog box.
If you typed incorrect information in the .sql file, a message appears in the data grid in the dialog box.
7. Complete the verification point as usual.

Capturing Data in LOVs and Record Groups

To capture the data in any LOV or record group, you can select the object from the Object List.

1. Display the form containing the LOV or record group.
2. If the form contains an LOV, make sure that the LOV is closed. You can capture the data in an LOV or record group only when the LOV is closed.
3. Start creating an Object Data verification point.
4. In the Select Object dialog box, click **Browse** to open the Object List dialog box.
5. Expand the **Window Name= OracleRootWindow** branch by double-clicking the plus sign.

- Expand the **Form** branch by double-clicking the plus sign.



If the LOV or record group objects do not appear in the Object List, check that the .sqa file has been created in the same directory as the form's executable file and contains the correct information. For details, see *Creating an .SQA Text File* on page 18-17.

- Select the object to test and click **OK** to open the Object Data Verification Point dialog box.
If you typed incorrect information in the .sqa file, a message appears in the data grid in the dialog box.
- Complete the verification point as usual.

Testing HTML Applications

This chapter explains how to use Robot to test HTML applications. It includes the following topics:

- ▶ About Robot support for HTML applications
- ▶ Configuring Internet Explorer for testing
- ▶ Try It! with HTML
- ▶ Making HTML applications testable
- ▶ Testing data in HTML elements
- ▶ How Robot maps HTML elements
- ▶ Supported data tests for HTML testing
- ▶ Testing properties of HTML elements
- ▶ Playing back scripts in Netscape Navigator
- ▶ Recording tips
- ▶ Enhancing object recognition of HTML elements

About Robot Support for HTML Applications

Rational Robot provides comprehensive support for testing HTML applications that run on the World Wide Web. Robot lets you test both static and dynamically-generated pages accessed from both standard and secured HTTP servers, regardless of make or model. Robot examines the data and properties of each HTML element, letting you test the elements that appear on your Web pages, including table data, links, and form elements, such as buttons, check boxes, lists, and text.

With Robot you can record and play back scripts in Microsoft Internet Explorer versions 4.x and later and play back these same scripts in Netscape Navigator versions 4.x and later. Scripts can be played back on a variety of Windows platforms, including Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

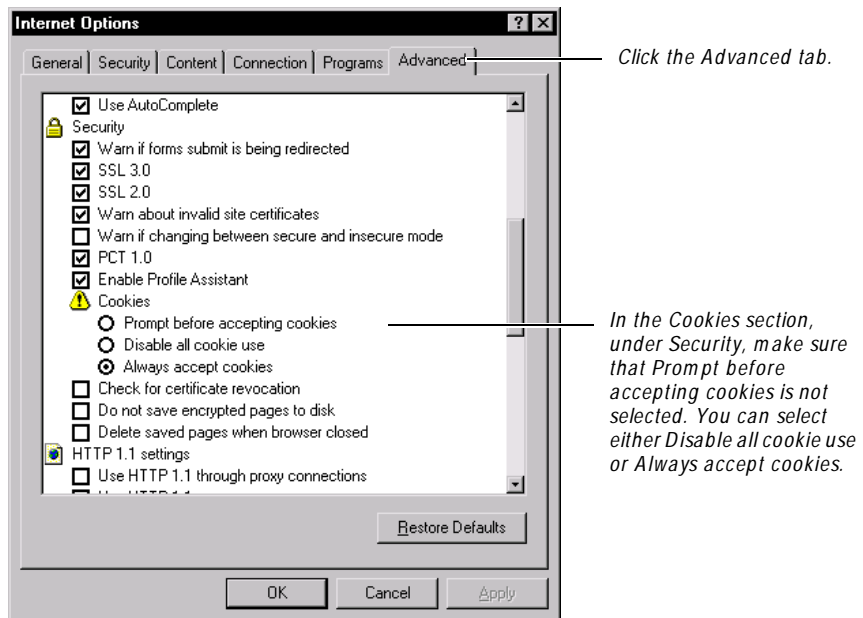
Configuring Internet Explorer for Testing

Before you record scripts, you should configure Internet Explorer so that scripts will play back in the same way as when you recorded them. For best results, you should configure Internet Explorer identically on both the computer that you record scripts on and the computer that you play back scripts on. In addition, you should disable the cookie prompt.

Disabling the Cookie Prompt

To disable the cookie prompt:

1. Start Internet Explorer.
2. Click **View** → **Internet Options**.



3. Click **OK**.

Try It! with HTML

Robot comes with the *Try it! with HTML* card and sample application.

The *Try it!* card provides quick instructions for recording tests on the HTML elements in the HTML application. The application contains elements that are specific to the HTML development environment. For example, you can test the properties and data of an HTML form.

Making HTML Applications Testable

To make HTML applications testable:

- ▶ Verify that the HTML extension is loaded in Robot.
- ▶ Enable HTML testing in Robot.

These steps are described in the following sections.

Verifying that the HTML Extension Is Loaded

To test HTML applications, you must first make sure that the HTML extension is loaded in Robot. To do this:

1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **HTML-MSIE** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Enabling HTML Testing in Robot

After loading the HTML extension, you must enable HTML testing so that Robot can recognize HTML elements. You can do this either by starting Internet Explorer through the Robot **Start Browser** command or by loading the Rational ActiveX Test control.

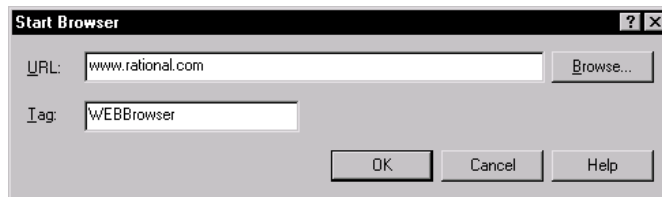
To enable HTML testing using the **Start Browser** command:



1. Start recording in Robot. To record, click the **Record GUI Script** button on the Robot toolbar. For details, see *Recording a New GUI Script* on page 4-16.
2. Type a script name or select a name from the list.
3. Click **OK** to display the GUI Record toolbar.



4. Click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
5. Click the **Start Browser** button on the GUI Insert toolbar.
6. Type the URL of the HTML application that you plan to test, or click **Browse** and select a local file.



7. Type the name of a tag to uniquely identify this instance of the browser. By using tags, you can test multiple instances of the browser.
8. Click **OK**.
Robot starts Internet Explorer and navigates to the specified URL.
9. Continue recording in Robot, as described in *Recording a New GUI Script* on page 4-16.

NOTE: If you start Internet Explorer outside of Robot without using the **Start Browser** command, you must open the `rbtstart.htm` page in your browser before loading the Web pages for testing. The `rbtstart.htm` page loads the Rational ActiveX Test Control, which is required for HTML testing in Robot. By default, this file is located in `C:\Program Files\Rational\Rational Test 7`.

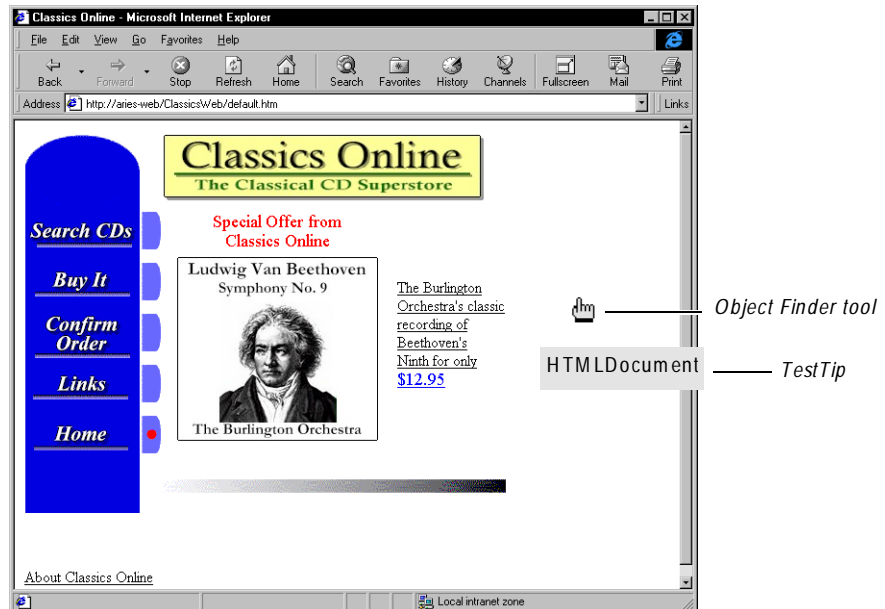
Testing Data in HTML Elements

Use the Object Data verification point to test the data in HTML elements. For example, you can use this verification point to test whether a purchase order has been processed or whether a Submit button returns the page that it is supposed to. For more specific information about verification points, see Chapter 6, *Creating Verification Points in GUI Scripts*.

To test an HTML element's data:

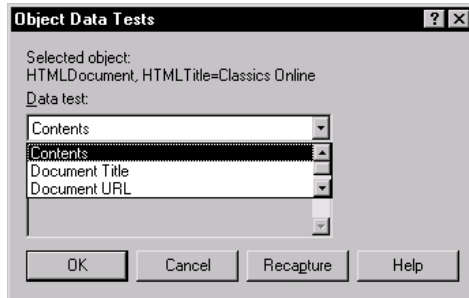
1. Start recording in Robot, as described in *Enabling HTML Testing in Robot* on page 19-3.
2. Navigate to the Web page that contains the elements to test. For example, navigate to the page that is returned after the user submits a page to be processed.
3. Click the **Object Data Verification Point** button on the GUI Insert toolbar.
4. Assign a name, wait state, and expected result for the verification point and then click **OK**, as described in *Tasks Associated with Creating a Verification Point* on page 6-6.
5. In the Select Object dialog box, drag the Object Finder tool over the page until the element that you want to test appears in the TestTip, as described in *Selecting and Identifying the Object to Test* on page 6-10.

For example, to test for the existence of a particular string of text on a page (any text within the < BODY> and < /BODY> tags), drag the Object Finder tool over the page until **HTMLDocument** appears in the TestTip and in the Selected Object field. You can see examples of these items in the following figure.



For a list of the HTML elements that you can test, see *How Robot Maps HTML Elements* on page 19-8.

6. Release the mouse button and click **OK**.
7. If the Object Data Test dialog box appears, select the data test to use and click **OK**.



There are five types of data tests that you can use on HTML elements. Not all tests are available for each type of element. For example, you might want to perform a Contents data test on an HTMLDocument. The Contents data test captures all of the visible text on the page, including text in forms fields, such as list boxes and combo boxes. For information about the types of data tests that are available for each element, see *Supported Data Tests for HTML Testing* on page 19-10.

8. Select the verification method that Robot should use to compare the baseline data captured while recording with the data captured during playback.

For example, you can use the **Find Sub String Case Sensitive** verification method to verify that the text captured during recording exactly matches a subset of the captured text during playback.

Suppose you want to verify that the text, *thank you for shopping with Classics Online*, is returned after a customer submits a purchase order. By selecting the **Find Sub String Case Sensitive** verification method, you can ensure that Robot will always test for the text, *thank you for shopping with Classics Online*, regardless of the text that surrounds it.

For more information about verification methods and the Object Data Verification Point dialog box, see *Working with the Data in Data Grids* on page 6-19.

9. Click **OK**.
10. When finished, click the **Stop Recording** button on the GUI Record toolbar.



Additional Examples

This section provides some additional examples of creating Object Data verification points for HTML elements.

To Test the Contents of a Drop-Down List Box

1. Add an Object Data verification point.
2. Select the ListBox with the Object Finder tool.
3. Select a Contents data test.
4. Select the **Case Sensitive** verification method to test for the entire contents of the list box. Select the **Find Sub String Case Sensitive** verification method to test for a subset of the list box items.

To Test for Text within a Table

1. Add an Object Data verification point.
2. Select the HTMLTable object with the Object Finder tool.
3. Select a Contents data test.
4. Select the **Case Sensitive** verification method to test for all of the text in the table. Select the **Find Sub String Case Sensitive** verification method to test for any text item with the table.

You can edit the text that the verification point captures. For information, see *Editing Data for a Clipboard or Object Data Verification Point* on page 6-21.

To Test the Destination of a Link

1. Add an Object Data verification point.
2. With the Object Finder tool, select HTMLLink to test a text-based link or HTMLImage to test an image link.
3. For the text-based link, select a Contents data test to capture the URL of the destination. For an image link, select an HTMLText data test to capture the URL of the destination.
4. Select the **Case Sensitive** verification method to test for the entire URL. Select the **Find Sub String Case Sensitive** verification method to test for part of the URL.

How Robot Maps HTML Elements

Robot maps HTML elements, such as INPUT, SELECT, BODY, TABLE, and others, to Robot object types, such as PushButton, ListBox, and HTMLDocument. The following table describes these mappings.

Robot object type	HTML element	Description
PushButton	<pre>< INPUT type= Submit> < INPUT type= Reset> < INPUT type= Button> < BUTTON></pre>	Used for elements in forms created with the < INPUT> tag where the type attributed is either Submit or Reset.
CheckBox	<pre>< INPUT type= Checkbox></pre>	Used for elements in forms created with the < INPUT> tag where the type attributed is Checkbox.
RadioButton	<pre>< INPUT type= Radio></pre>	Used for elements in forms created with the < INPUT> tag where the type attribute is Radio.
ComboBox	<pre>< SELECT size= 1> < OPTION> ... < /SELECT></pre>	Used for elements in forms created with the < SELECT> tag where the size attribute is equal to one.
ListBox	<pre>< SELECT size= > n> < OPTION> ... < /SELECT></pre>	Used for elements in forms created with the < SELECT> tag where the size attribute is greater than one.
EditBox	<pre>< INPUT type= Text> < INPUT type= TextArea></pre> <p>Text is for single line controls. TextArea is for multiline controls.</p>	Used for elements in forms created with the < INPUT> tag where the type attribute is equal to Text or TextArea.
HTMLLink	<pre>< A> ... < /A></pre>	Used for anchor elements.
HTMLImage	<pre>< IMG></pre>	Used for server- and client-side image maps or images on a page.

(Continued)

Robot object type	HTML element	Description
HTMLDocument	All text between < BODY> and < /BODY>	Used so that verification points can access all of the data on a page. Individual elements are identified by tag and name or prefix.
HTMLTable	All text between < TABLE> and < /TABLE>	Used to test tables. Verification points act on the entire table. When capturing object properties, each cell appears as a separate subelement.
HTMLActiveX	< OBJECT>	Used to record against clicks on ActiveX controls embedded in the page.
HTML	All other tags	Used for all other tags when the tag has an ID or a name. HTML can be used, for example, to identify and test a single paragraph on a page. In this case, you must manually insert an ID into the HTML source to tag the particular paragraph.

Supported Data Tests for HTML Testing

The following table describes the data tests that are available for each Robot object type supported in the HTML environment. For information about creating your own data tests, see Appendix B, *Working with Data Tests*.

Robot object type	Supported data test	Description of data test
PushButton CheckBox RadioButton EditBox	Contents	Captures and compares visible text.
	HTMLText	Captures and compares the HTML source.
ComboBox ListBox	Contents	Captures and compares the text of all items in the box.
	ItemData	Captures and compares the value attribute in the HTML source.
	HTMLText	Captures and compares the HTML source.
HTMLLink	Contents	Captures and compares the “href” of the anchor — that is, the URL of the destination.
	HTMLText	Captures and compares the HTML source.
HTMLImage	HTMLText	Captures and compares the HTML source of the image.
HTMLDocument	Contents	Captures and compares the visible text of the entire document, including text in all of the elements in HTML forms, such as list boxes and combo boxes.
	HTMLText	Captures and compares the HTML source of the entire document.
	Document URL	Captures the URL of the document.
	Document Title	Captures the title of the document.
HTMLTable	Contents	Captures and compares the visible text of the entire table.
	HTMLText	Captures and compares the HTML source of the table.
HTMLActiveX	None	

Testing Properties of HTML Elements

Another way to test your Web pages is to use the Object Properties verification point.

Properties describe an HTML element's characteristics, such as its appearance, state, behavior, and data. The Rational Object Testing technology inspects and verifies all properties of the HTML elements in your application, including hidden properties that cannot be tested manually.

For example, you can create an Object Properties verification point to capture and compare the modification date of a page or to determine whether a check box or a radio button is selected.

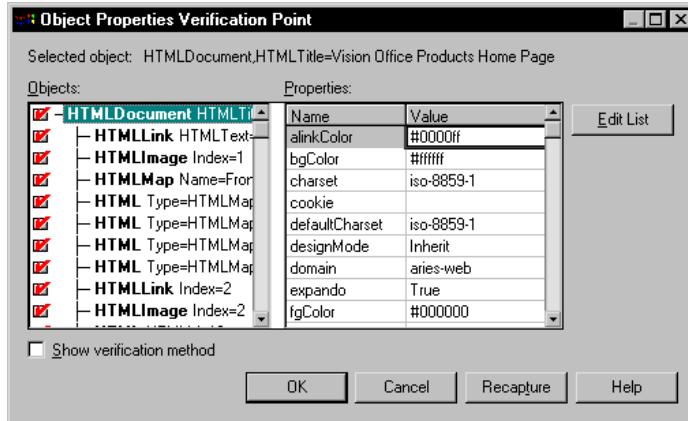
The Object Properties verification point provides you with information about more than 20 properties for each HTML element. Some properties provide you with the same information as a data test. For example, a radio button's Value property provides you with the same information as a Contents data test.

For more information about the Object Properties verification point, see the Robot Help.

To test an HTML element's properties:

1. Start recording in Robot, as described in *Enabling HTML Testing in Robot* on page 19-3.
2. Navigate to the Web page that contains the element to test.
3. Click the **Object Properties Verification Point** button on the GUI Insert toolbar.
4. Assign a name, wait state, and expected result for the verification point and then click **OK**, as described in *Tasks Associated with Creating a Verification Point* on page 6-6.

5. Select the element to test and then click **OK**. The element's properties are displayed as follows:



6. Click **OK** to insert the verification point.

Playing Back Scripts in Netscape Navigator

With Robot, you can record scripts in Internet Explorer and play them back in Netscape Navigator. Netscape playback requires at minimum a 200 MHz Pentium with at least 64 MB of RAM.

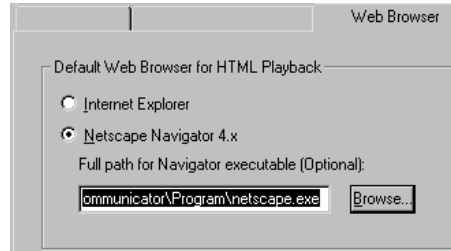
Configuring Robot for Netscape Playback

To configure Robot for Netscape playback:

1. In Robot, click **Tools** → **Extension Manager** and make sure that the **HTML-Navigator** extension is selected.

To improve performance, clear any extensions that are not used. Click **OK**. Restart Robot to load the extension.

2. In Robot, click **Tools** → **GUI Playback Options** and click the **Web Browser** tab.



3. Select Netscape Navigator 4.x.

If you have multiple versions of Navigator, you can specify the full path to the version of Navigator that you want to use for playback.

Alternatively, you can edit your script so that it will play back in Navigator. For example, to specify Navigator playback, type one of the following commands in your script:

```
SQASetDefaultBrowser "Navigator"
```

```
SQASetDefaultBrowser "Navigator=c:\program files\  
netscape\communicator\program\netscape.exe"
```

Use the first command if you have only one version of Navigator on your computer. Use the second command if you have multiple versions of Navigator on your computer, replacing the program path indicated with the actual path on your computer. Be sure to insert the command before the StartBrowser command is invoked in the script.

Differences Between Internet Explorer and Navigator

Both Microsoft Internet Explorer and Netscape Navigator contain proprietary extensions to the HTML standard. As a result, HTML documents are often rendered differently in each browser.

Navigator support is provided for script playback only. In addition, you should be aware of the following areas in which Navigator playback differs from Internet Explorer playback:

- ▶ The following Robot object types, which represent the Microsoft implementation of Dynamic HTML, are not recognized by Robot during Navigator playback:
 - HTMLActiveX
 - HTMLEmbed
 - HTMLScriptlet

- ▶ The following Robot object types do not support Robot action commands, such as Click, Drag, and Scroll, during Navigator playback:
 - HTMLTable
 - HTML
- ▶ HTMLImage is an all-inclusive object type encompassing image maps (both client- and server-side), input images, and regular images. Actions for server-side image maps are fully supported. However, when Robot plays back scripts in Navigator, coordinate-based actions for client-side image maps are not supported, and no actions are supported for input and regular images.
- ▶ Not all of the object properties recognized by Internet Explorer during recording are supported by the Navigator extension. In addition, other object properties may have different values because of browser differences.
- ▶ The current Robot implementation of Navigator playback requires a Web server that allows disk caching.

Recording Tips

This section provides suggestions to help you record scripts in Robot. It contains the following tips:

- ▶ Capturing the properties of Java applets in HTML pages
- ▶ Synchronizing pages
- ▶ Recording mouse movements
- ▶ Ensuring browser compatibility

Capturing the Properties of Java Applets in HTML Pages

There are two ways to capture the properties of a Java applet embedded in a Web page. You can either capture the properties of the entire window, or you can capture the properties of the applet itself. Either way, you should first click **Tools** → **Extension Manager** in Robot to verify that the Java extension is loaded.

To capture the entire window, including the applet:

1. Add an Object Properties verification point.
2. With the Object Finder tool, point to the title bar of the browser window until **Window** appears in the TestTip.

3. Click **OK**.

NOTE: Do not point to the `HTMLDocument` object type. This object type provides minimal detail about Java applets.

To capture only the properties of the Java applet:

1. Add an Object Properties verification point.
2. With the Object Finder tool, point to the Java applet until **JavaWindow** appears in the TestTip.
3. Click **OK**.

Synchronizing Pages

Whenever you click on a page for the first time, Robot inserts a `Browser NewPage` command into the script. This command causes Robot to wait for the contents of the page to fully load before continuing, and also helps prevent timing problems that could cause scripts to fail when they are played back. The following scenarios illustrate how this works.

For more information about page synchronization, see the description of the `Browser` command in the *SQABasic Language Reference*.

Capturing Properties or Data of Window Objects

Any time you plan to capture the data or properties of a window object, be sure to click in the window before inserting an Object Data or Object Properties verification point. Clicking in the window inserts a `Browser NewPage` command into the script and ensures proper synchronization between pages.

Using the Browser's Back and Forward Buttons

If you use the **Back** or **Forward** buttons to navigate to a previously viewed page while recording a script, you must perform some action on the page before you click the **Back** or **Forward** button again. Clicking on the page, for example, inserts the `Browser NewPage` command into the script, and just as with the previous example, ensures proper synchronization between pages.

Recording Transactions

When you submit a purchase order for an e-commerce transaction, there may be a substantial delay before the Web server responds with a confirmation. In fact, a Web server may send one or more interim pages while it is processing the transaction. Robot waits 30 seconds, by default, for this confirmation to arrive from the server. If the confirmation requires additional time, you will see the warning *New Page Not Found* in the LogViewer after you play back the script. To correct this problem, edit the script by adding a **Wait** value greater than 30 seconds to the Browser `NewPage` command, as in the following example:

```
Browser NewPage, "HTMLTitle=Thank you for your order!", "Wait=45"
```

In this example, the use of **HTMLTitle** in the recognition string allows Robot to identify the correct page at playback and skip over any interim pages. The **Wait** value causes Robot to wait 45 seconds for this specific page to be displayed.

Recording Mouse Movements

With Dynamic HTML, it is possible to cause a page to change color or to cause text on a page to update simply by moving the mouse over the page. To capture these mouse movements:

1. Start recording in Robot.
2. Navigate to the page that contains the Dynamic HTML.
3. Press CTRL+ SHIFT+ R to enter low-level recording mode.
4. Move the pointer over the portion of the page that is affected by the mouse movement.
5. Press CTRL+ SHIFT+ R to stop low-level recording mode.
6. Insert an Object Properties verification point.

For more information about low-level recording, see *Switching to Low-Level Recording* on page 4-22.

Ensuring Browser Compatibility

To help ensure that scripts recorded in Internet Explorer play back as expected in Navigator, observe the following recording tips:

- ▶ Do not click on the scroll bars in Internet Explorer during recording. If you need to scroll, pause the recording, scroll the window, and then resume recording.
- ▶ Avoid using the Forward or Back arrows during recording in Internet Explorer. If you find it necessary to use them, edit your script by replacing the arrows with the following commands:

- Browser Back, "", ""
- Browser Forward, "", ""

For more information, see the *SQABasic Language Reference*.

- ▶ Exit Internet Explorer by clicking the **Close Window** button in the upper-right corner of the window, rather than by clicking **File** → **Close**.

Enhancing Object Recognition of HTML Elements

Robot uses recognition methods to identify HTML elements in the application-under-test. These recognition methods are saved as arguments in scripts to help Robot identify these elements during playback. For example, Robot can identify a link by the visible text of the link — that is, the text that a user clicks on. If this text changes after a script has been recorded, the script may fail when it is played back.

The best way to ensure that Robot recognizes this link is to assign it an ID that will always remain the same, even if the visible text changes — for example:

```
See <A HREF="about.htm/" ID=about> About Our Product</A>
```

To enhance the recognition of image elements, it is best to use either ALT tags or ID tags, as shown in the following examples:

```
<A HREF="lookup.htm" ></A>
```

```
<A HREF="search.htm" ></A>
```

For more information about recognition methods, see the *SQABasic Language Reference*.

Testing Java Applets and Applications

Java is an object-oriented programming language that lets you write programs that can run on any computer that implements the Java Virtual Machine (JVM).

This chapter describes how to use Robot to test both Java applets running in a browser and standalone Java applications. It includes the following topics:

- ▶ About Robot support for Java
- ▶ Making Java applets and applications testable
- ▶ Setting up the sample Java applet
- ▶ Testing data in Java components
- ▶ Support for custom Java components
- ▶ Supported data tests for Java testing
- ▶ Testing properties of Java components
- ▶ Enhancing object recognition of Java components

For a good introduction to Java concepts and terminology, read the Java language overview at the following URL:

<http://java.sun.com/docs/overviews/java/java-overview-1.html>

About Robot Support for Java

Rational Robot provides comprehensive support for testing GUI components in both Java applets and standalone Java applications. With its Object Testing technology, Robot examines the data and properties of Java components. This means that Robot can do the following:

- ▶ Determine the names of components in your program, and use those names for object recognition.
- ▶ Capture properties of Java components with the Object Properties verification point.
- ▶ Capture data in Java components with the Object Data verification point.

Robot includes several Java-specific object types for testing Java components, including `JavaPanel`, `JavaWindow`, `JavaSplitPane`, `JavaMenu`, `JavaPopupMenu`, `JavaTable`, `JavaTableHeader`, and `JavaTree`.

Robot scripts can be played back on a variety of Windows platforms, including Windows NT 4.0, Windows 2000, Windows 98, and Windows 95, and are transportable across the various Java platforms.

The following matrix presents an overview of the Java support in Robot:

		Java Virtual Machines (JVM)		
		Sun JDK 1.1, 1.2	Microsoft Java SDK 3.1	Netscape 4.05 and later (Applets only)
Supported Class Libraries	AWT		AWT	AWT
	JFC/Swing		JFC/Swing	JFC/Swing
			WFC	
	Symantac Visual Cafe		Symantac Visual Cafe	Symantac Visual Cafe
	KL Group		KL Group	KL Group
	Extensibility		Extensibility	Extensibility

For information about support for additional Java environments and foundation classes, see the Rational Web site at www.rational.com.

Robot Support for Testing Java Applets and Applications

A Java **applet** is a Java program embedded inside a Web page or displayed with an applet viewer. **Java applications** are standalone programs that require a Java Virtual Machine (JVM) in order to run. Java applications create their own windows and are not embedded inside Web pages.

Support for Testing Java Applets

Robot provides support for testing Java applets that run in the following environments:

- ▶ Internet Explorer 4.0 or later with the Microsoft Java Virtual Machine (JVM) or with the Sun plug-in
- ▶ Netscape 4.05 or later with the Netscape JVM or with the Sun plug-in
- ▶ Microsoft Appletviewer
- ▶ Sun Appletviewer from the Java Developer Kit (JDK) 1.1.4 or later

Support for Testing Java Applications

Robot provides support for testing standalone Java applications that run in the following environments:

- ▶ Sun JDK 1.1.4 or later
- ▶ Sun Java Runtime Environment (JRE) 1.1.4 or later
- ▶ Microsoft JVM (jview)

You can create Robot scripts that play back Java applications under Rational Quantify and PureCoverage. For information, see *Setting Diagnostic Tools Options* on page 9-11.

Supported Foundation Class Libraries

With Robot you can test Java objects that are instances of the following class libraries or objects that are derived from any of these class libraries:

- ▶ Abstract Windowing Toolkit (AWT)

AWT is a collection of core graphical user interface classes that will run on any supported Java platform.

- ▶ Java Foundation Class (JFC)

JFC is an extension to AWT that provides additional graphical user interface classes, such as:

- Swing
- Accessibility
- Drag and Drop
- Java-2D, Java-3D
- Windows Foundation Classes (WFC). Java programs using WFC components run only on Microsoft JVMs.

- ▶ Symantec Visual Cafe

- ▶ KL Group

In future releases, Rational will provide support for additional foundation class libraries. For information about the most recent enhancements, see the Rational Web site.

NOTE: For information about how to enable your computer to support other foundation classes such as Swing, see *Setting Up the Sample Java Applet* on page 20-7.

Making Java Applets and Applications Testable

To make Java applets and applications testable, you need to:

- ▶ Run the Java Enabler. The **Java Enabler** makes each host environment testable.
- ▶ Verify that the Java extension is loaded. The **Java extension** includes those additions to Robot that allow Robot to test Java.

These tasks are described in the following sections.

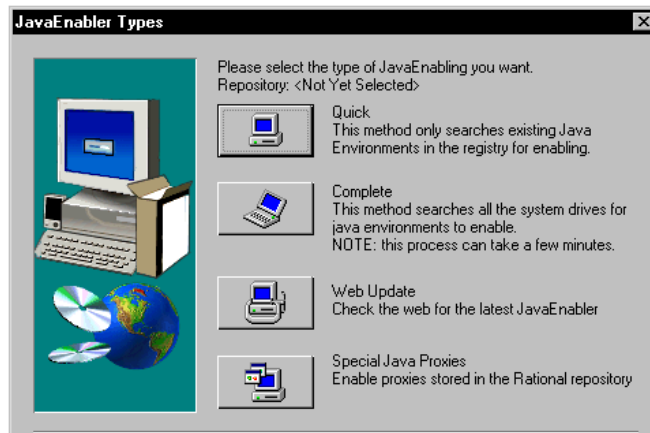
Running the Java Enabler

By default, Java testing is disabled in Robot. To enable Java testing, you need to run the Java Enabler. The Java Enabler is a wizard that scans your hard drive looking for Java environments such as Web browsers and Sun JDK installations that Robot supports. The Java Enabler only enables those environments that are currently installed.

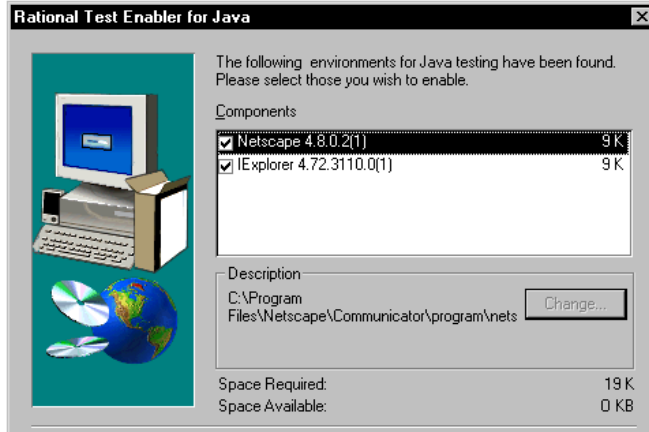
NOTE: If you install a new Java environment such as a new release of a browser or JDK, you must rerun the Enabler after you complete the installation of the Java environment. You can download updated versions of the Java Enabler from the Rational Web site whenever support is added for new environments. To obtain the most up-to-date Java support, simply rerun the Java Enabler.

To run the Java Enabler:

1. Make sure that Robot is closed.
2. Click **Start** → **Programs** → *Rational product name* → **Rational Test** → **Java Enabler**.
3. Select one of the available Java enabling types.



4. Select the environments to enable.



5. Click **Next**.
6. Click **Yes** to view the log file.

NOTE: If the Java Enabler does not find your environment, you must upgrade to one of the supported versions and rerun the Java Enabler. For a list of supported environments, see *About Robot Support for Java* on page 20-2.

The following table describes what the Java Enabler does to update the various Java environments:

With this Java environment	The Java Enabler
Sun JDK 1.1	U pdates the CLASSPATH on Windows 95 and Windows 98 and the system CLASSPATH on NT 4.0 and Windows 2000 to include the path to the sqarobot.jar file.
Sun JRE 1.1	U pdates the CLASSPATH on Windows 95 and Windows 98 and the system CLASSPATH on NT 4.0 and Windows 2000. If the CLASSPATH is not used by the Java application, you will need to manually add the sqarobot.jar file to the CLASSPATH used by the application.
Sun JDK 1.2	<ul style="list-style-type: none"> ▶ Copies the sqarobot.jar file to the Jre\Lib\Ext directory. ▶ U pdates the accessibility.properties file to reference the Robot runtime monitor class.

(Continued)

With this Java environment	The Java Enabler
Microsoft JVM	Updates the trusted CLASSPATH in the registry.
Netscape JVM	<ul style="list-style-type: none"> ▶ Copies the sqarobot.jar file to the Java\Classes directory within the Netscape directory structure. ▶ Copies the sqajava.dll file to the Java\Bin directory. ▶ Updates the awt.properties file in the Java\Classes directory.

Verifying that the Java Extension Is Loaded

To test Java, you must first make sure that the Java extension is loaded in Robot.

To verify that the Java extension is loaded:

1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **Java** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Setting Up the Sample Java Applet

After you complete the tasks in the proceeding section, you can start testing your Java applets or standalone applications, or you can learn more about testing Java by installing the sample Java applet that Rational provides.

To use the sample applet, you must perform the following tasks:

1. Install the sample Java applet.
2. Install Sun's Swing foundation classes. (Swing is a subset of JFC.)
3. Start the sample Java applet.

These tasks are described in the following sections.

Installing the Sample Java Applet

To install the sample Java applet:

1. Click **Start** → **Programs** → *Rational product name* → **Rational Test** → **Set Up Rational Test Samples**.
2. Select **Java Sample**.
3. Click **Next** to complete the installation.

Installing the Swing Foundation Classes

The sample Java applet requires Sun's JFC 1.1 or Swing foundation classes version 1.1 or later. You can download JFC/Swing from www.javasoft.com/products/jfc. After you download the Swing foundation classes, you need to install them.

NOTE: When you run the Java Enabler, the Swing foundation classes are automatically installed. For information, see *Running the Java Enabler* on page 20-5.

Installing Swing Under Windows NT 4.0

To install the Swing foundation classes on a computer running Windows NT 4.0:

1. Double-click the file that you just downloaded to install Swing on your computer.
2. Click **Start** → **Settings** → **Control Panel**.
3. Double-click **System**.
4. Click the **Environment** tab.
5. Click **CLASSPATH** under System Variables.
6. Move the cursor to the end of the **Value** box. Type a semi-colon and the full path to the swingall.jar file.

This file is installed when you install Swing. For example, if you installed Swing-1.1 at the root of your C drive, you would type the following:

```
;c:\swing-1.1\swingall.jar
```

7. Click **Set**.
8. Click **OK**.

Installing Swing Under Windows 2000

To install the Swing foundation classes on a computer running Windows 2000:

1. Double-click the file that you just downloaded to install Swing on your computer.
2. Click **Start** → **Settings** → **Control Panel**.
3. Double-click **System**.
4. Click the **Advanced** tab. Click **Environment Variables**.
5. Click **CLASSPATH** under System Variables, and click **Edit**.
6. Move the cursor to the end of the **Variable Value** box. Type a semi-colon and the full path to the swingall.jar file.

This file is installed when you install Swing. For example, if you installed Swing-1.1 at the root of your C drive, you would type the following:

```
;c:\swing-1.1\swingall.jar
```

7. Click **OK**.
8. Click **OK**.

Installing Swing Under Windows 98 and Windows 95

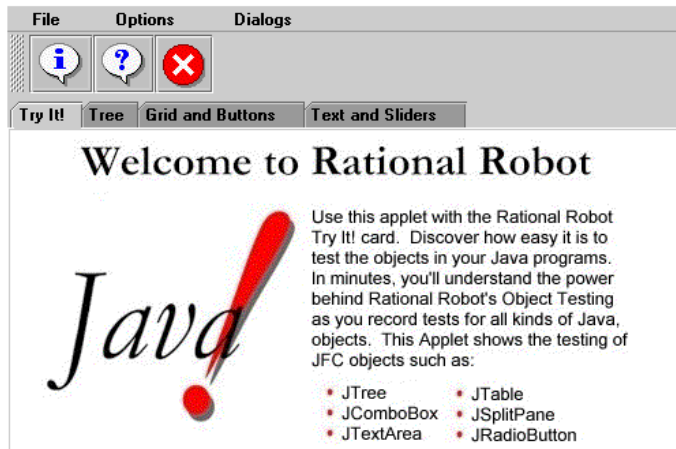
To install the Swing foundation classes on a computer running Windows 98 or Windows 95:

1. Double-click the file that you just downloaded to install Swing on your computer.
2. Edit the CLASSPATH environment variable in your workstation's **autoexec.bat** file to include the path to the swingall.jar file.

Starting the Sample Java Applet

To run your default browser and load the sample Java applet:

- ▶ Click **Start** → **Programs** → **Rational Test Samples** → **Java**.



For instructions about using the Java applet, see the *Try it! with Java* card.

Testing Data in Java Components

Use the Object Data verification point to test the data in Java components. For more information about verification points, see Chapter 6, *Creating Verification Points in GUI Scripts* and the Robot Help.

To test a Java component's data:



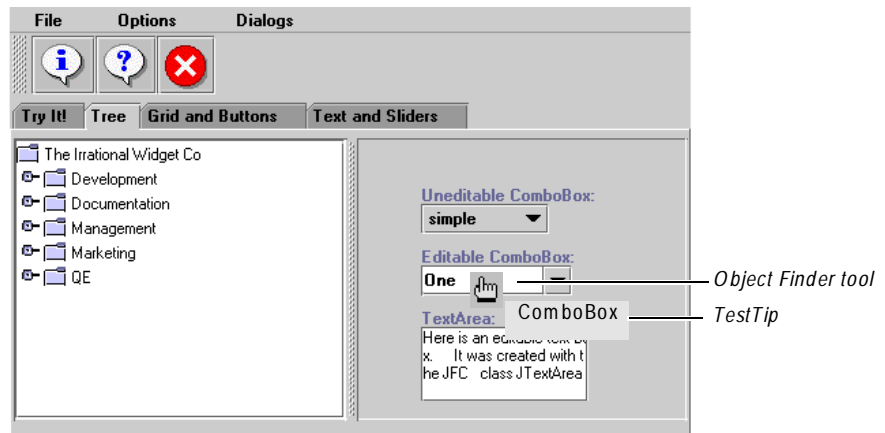
1. Start recording in Robot. For details, see *Recording a New GUI Script* on page 4-16.
2. Open the Java applet or application that you want to test.

If you plan to play back the script under Rational Quantify or PureCoverage to test a Java application (not an applet), use the **Start Java Application** button on the GUI Insert toolbar. For information about Quantify and PureCoverage, see *Setting Diagnostic Tools Options* on page 9-11. For information about starting a Java application, see *Starting an Application* on page 5-1.

3. Navigate to the page that you want to test.
4. Start creating the Object Data verification point. (For instructions, see *Object Data verification point* in the Robot Help Index.)

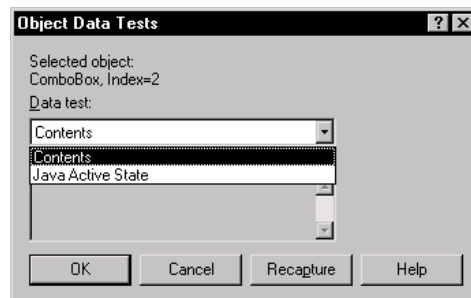
5. Assign a name, wait state, and expected result for the verification point and then click **OK**, as described in *Tasks Associated with Creating a Verification Point* on page 6-6.
6. In the Select Object dialog box, drag the Object Finder tool over the page until the component you want to test appears in the TestTip, as described in *Selecting and Identifying the Object to Test* on page 6-10.

For example, to test that a particular item in a ComboBox is selected, drag the Object Finder tool over the page until **ComboBox** appears in the TestTip.



7. Release the mouse button.
8. If the dialog box is still open, click **OK**.
9. If the Object Data Tests dialog box appears, select the data test to use and click **OK**.

For example, to test that a particular element in a ComboBox is selected, select the **JavaActiveState** data test. To test all of the elements in a ComboBox, select the **Contents** data test.



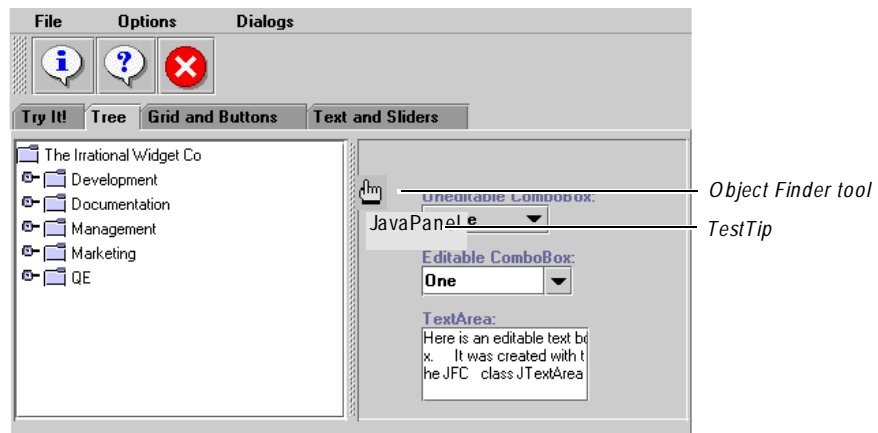
10. Complete the verification point as usual.

Testing the Contents of a Java Panel

A feature unique to Java testing is the ability to collect and test the data for all the known components on a Java panel. A **panel** is a container of components and other panels that you have grouped together.

To test the contents of a Java panel:

1. Repeat steps 1 – 5 from the section *Testing Data in Java Components* on page 20-10.
2. In the Select Object dialog box, drag the Object Finder tool over the page until **JavaPanel** appears in the TestTip, as described in *Selecting and Identifying the Object to Test* on page 6-10.



You can use the Object Data verification point to capture the active state of each component in the panel. Robot inserts only the fields with dynamically changing data, such as `EditText`, `RadioButton` or `ComboBox`, into the panel's Object Data verification point. Components without active state (`JavaPanels`, `PushButtons`, or `Labels`) are not saved in the verification point.

3. Repeat steps 7 – 10 from the section *Testing Data in Java Components* on page 20-10.

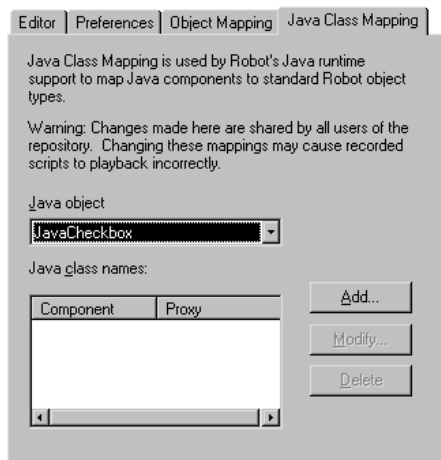
Support for Custom Java Components

With Java, you can create your own user-defined classes. For example, you can create a new kind of button class that acts the same way as an existing AWT Button. If you derive your new button from the AWT button class, Robot can map the new class correctly.

Beyond the standard class libraries, Robot supports custom Java components in other class libraries, such as those available with IBM Visual Age and other IDEs. This extended Java support is provided through the Robot Java Open API.

By mapping a standard SQABasic object type (such as a push button or Java panel) to a custom Java component, and by using the Java Open API to create a proxy interface for the custom component, you can use Robot to test the custom component.

You map standard SQABasic object types with custom Java components in the **Java Class Mapping** tab of the Robot General Options dialog box, as shown here:



To display the Java Class Mapping tab in Robot:

- ▶ Click **Tools** → **General Options**. Click the **Java Class Mapping** tab.

For More Information About Java Support

For information about mapping standard SQABasic object types to custom Java components, click the **Help** button in the **Java Class Mapping** tab to display Robot Help for that topic, or search the Robot Help index for *Java Class Mapping tab*.

For information about the Robot Java Open API, open the online document overview-summary.htm. By default, this file is located in the following path:

Program Files\Rational\Rational Test 7\JavaEnabler\api

You can also open this file through the Robot and SQABasic Help systems wherever you see the link to the *Robot Java API Overview*.

Supported Data Tests for Java Testing

The following table describes the data tests that are available for each Robot object type supported in the Java environment.

Robot object type	Supported data test	Description of data test
PushButton, Label	Contents	Captures and compares visible text.
CheckBox RadioButton	Contents	Captures and compares visible text.
	Java Active State	Displays Selected or Not Selected depending on the current state.
EditBox	Contents	Captures and compares visible text.
	Java Active State	Captures and compares selected text.
ScrollBar, TrackBar	Contents	Displays the current value.
	Java Active State	Displays the Value , Unit Increment , and Block Increment .
ListBox, ComboBox ComboListBox	Contents	Captures and displays all elements in the list box, combo box, or combo list box.
	Java Active State	Captures and displays the currently selected elements.
TabControl	Contents	Captures and displays all of the tabs.
	Java Active State	Captures and displays the currently selected tab.
JavaTree	Contents	Captures and displays the contents of the Java tree.
	Java Active State	Captures and displays the currently selected elements in the Java tree.

(Continued)

Robot object type	Supported data test	Description of data test
JavaSplitPane	Contents	Captures and displays the current scroll value for the pane.
JavaTable	Contents	Captures and displays each cell in the table.
JavaTableHeader	Contents	Displays each column header in the table.
JavaPanel JavaWindow	Contents	Captures and displays the data-sensitive objects in the object and its children. A data-sensitive object is a child or nested child of a selected object that contains dynamically-generated data. Examples include EditBoxes, CheckBoxes, and RadioButtons, but not PushButtons.
JavaMenu JavaPopupMenu	Contents Java Active State Menu Test	Captures and displays the menu header and items for each menu. Captures and displays the currently selected menu item. Captures and displays the menu header and items for each menu, along with accelerator keys and mnemonic specifiers.

Testing Properties of Java Components

You can use the Object Properties verification point to test the properties of Java components.

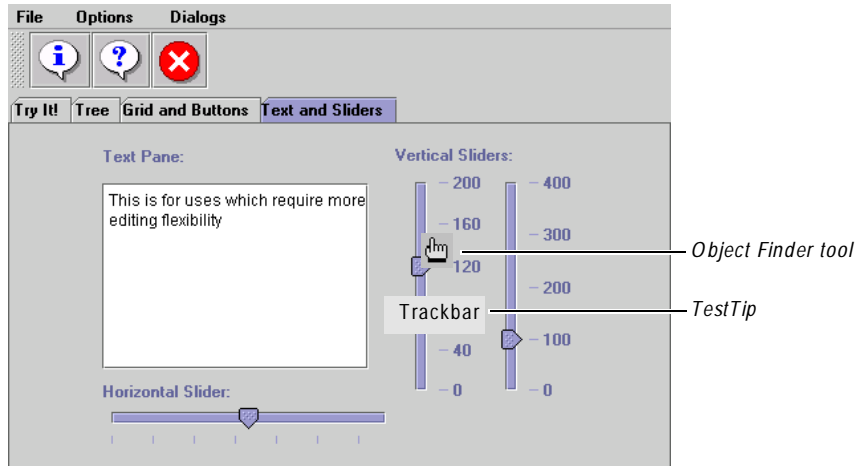
Properties describe a component's characteristics such as its appearance, state, behavior, and data. The Rational Object Testing technology inspects and verifies all properties of the components in your application, including hidden properties that cannot be tested manually.

To test the properties of Java components:

1. Start recording in Robot. For details, see *Recording a New GUI Script* on page 4-16.
2. Navigate to the page that contains the component you want to test.
3. Start creating the Object Properties verification point as usual. (For instructions, see *Object Properties verification point* in the Robot Help Index.)
4. Assign a name, wait state, and expected result for the verification point and then click **OK**, as described in *Tasks Associated with Creating a Verification Point* on page 6-6.

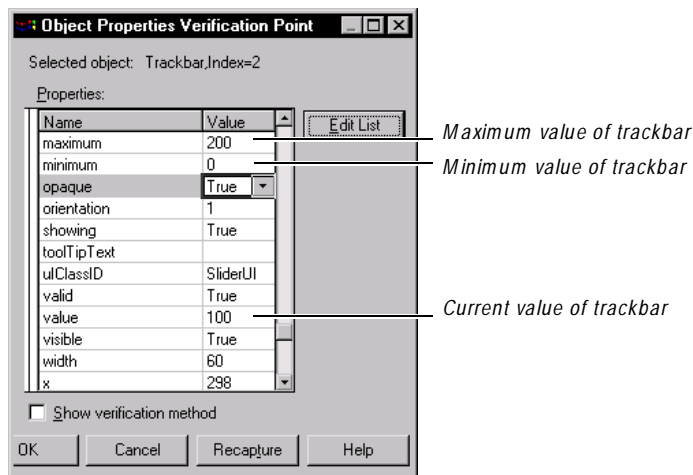
- In the Select Object dialog box, drag the Object Finder tool over the page until the component that you want to test appears in the TestTip, as described in *Selecting and Identifying the Object to Test* on page 6-10.

For example, to test for the current, minimum, and maximum values in a trackbar, drag the Object Finder tool over the component until **Trackbar** appears in the TestTip. See the following example.



- Release the mouse button. If the Select Object dialog box is still open, click **OK**.

The object's properties appear:



7. Click **OK** to complete the verification point.

NOTE: When you create an Object Properties verification point, you can edit the list of properties that are saved with the component. Robot saves the list relative to the Java class name (for example, `Java.awt.Button`), not the Robot command name (for example, `PushButton`). This allows you to save derived classes with different lists of properties. For more information about adding and removing properties from the properties list, see the Robot Help.

Enhancing Object Recognition of Java Components

Robot uses recognition methods to identify components in the application-under-test. These recognition methods are saved as arguments in scripts so Robot can correctly identify these component during playback. For example, Robot can identify a Java Button by the visible text displayed on the button. If the text changes after the script has been recorded, the script may fail when it is played back.

The best way to make sure that Robot recognizes a Java component is to assign a name to the object in the Java code. Although Java supports several ways of doing this, Robot works as follows:

- ▶ If the component exports a `public String getName()` method and this method returns a name that starts with a dot (`.`) character, Robot uses this name to uniquely identify the component. The dot prefix is necessary to make sure that the name has been explicitly set by the user and not set to a default value by the browser.

This recognition method is available with all `java.awt.Component` derived classes.

- ▶ If the component contains an `accessibleContext.accessibleName` property, Robot will use it to recognize the component.

This recognition method is available only with JFC-derived classes.

By assigning unique names to your Java components, you can make your scripts more resilient.

Testing Java Applets and Applications

Testing PowerBuilder Applications

This chapter describes how to test 32-bit PowerBuilder applications with Rational Robot. It includes the following topics:

- ▶ About Robot support for PowerBuilder applications
- ▶ Verifying that the PowerBuilder extension is loaded
- ▶ Try it! with PowerBuilder
- ▶ Recording actions on DataWindows
- ▶ Testing an expression value of a DataWindow property
- ▶ Testing DataStore controls and hidden DataWindows
- ▶ Capturing data in a DropDownDataWindow and DropDownListBox
- ▶ Testing the value of a DataWindow computed field

About Robot Support for PowerBuilder Applications

Rational Robot provides comprehensive support for testing 32-bit applications built with PowerBuilder 5.0 through 7.0. Robot supports the testing of applications that you migrate from one PowerBuilder version to another, and allows for the reuse of scripts between Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

With its Object Testing technology, Robot examines data and properties that are not visible to the user. Robot uses Object-Oriented Recording to recognize a PowerBuilder object by its internal name.

You can use Robot to test all PowerBuilder and third-party components, including:

- ▶ DataStore controls and hidden DataWindows
- ▶ ActiveX controls
- ▶ RichTextEdit controls
- ▶ DataWindows with RichText presentation style
- ▶ All properties of a DataWindow computed field, including *expression* and *value*

Verifying that the PowerBuilder Extension Is Loaded

To test PowerBuilder applications, you should first verify that the Robot PowerBuilder extension is loaded in Robot.

To verify that the extension is loaded:

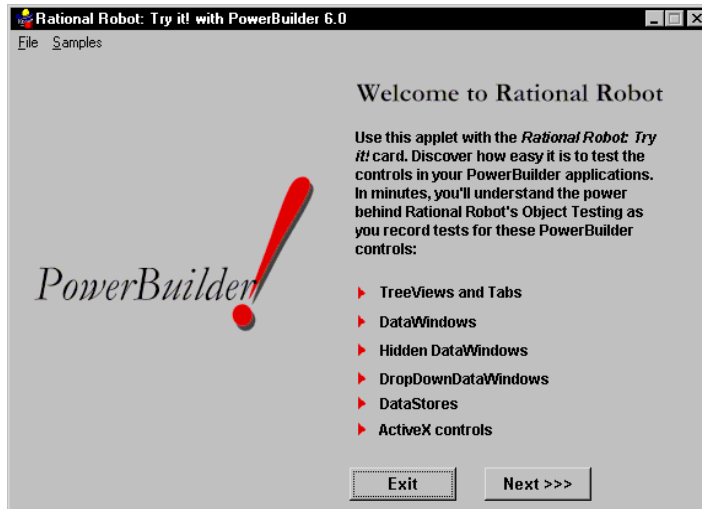
1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **PowerBuilder** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Try it! with PowerBuilder

Robot comes with the *Try it! with PowerBuilder* card and sample applet.

The *Try it!* card provides quick instructions for recording tests on objects in the PowerBuilder applet. The applet contains objects that are specific to the PowerBuilder development environment. For example, you can test the properties and data of a DataWindow that contains DropDownDataWindow and computed field.



Recording Actions on DataWindows

Robot uses certain action parameters to identify a DataWindow row if the action is a mouse click. These parameters help make your scripts more reliable and readable by reducing the dependency on absolute positions.

These parameters are used when you record actions on:

- ▶ DataWindows
- ▶ DataWindow sub-objects

NOTE: For detailed information about DataWindow action parameters, see the *SQABasic Language Reference*.

Parameters for a Mouse-Click Action

The following action parameters identify a DataWindow row if the action is a mouse click:

Action parameter	Action during playback
Col=%;Value=x	Clicks the row specified by the column/value pair. Col is the numeric position of a DataWindow column and Value is the contents of the cell located at the intersection of column Col and the clicked row. (See the next section, <i>Value-Based Recording</i> .)
ColName=\$;Value=x	Clicks the row specified by the column/value pair. ColName is the developer-assigned name of a DataWindow column, and Value is the contents of the cell located at the intersection of column ColName and the clicked row. (See the next section, <i>Value-Based Recording</i> .)
CurrentRow	Clicks the currently selected row in the DataWindow.
LastRow	Clicks the last row in the DataWindow.
Row=%	Clicks the row specified by the number (first row = 1).
Text	Clicks the row identified by the visible text in the DataWindow row.
VisibleRow=%	Clicks the visible row specified by the number. The range of row numbers begins with the first visible row.

Robot can record these action parameters for the following types of multi-row DataWindow and DropDownDataWindow presentation styles: FreeForm, Grid, OLE, RichText, and Tabular.

Robot records coordinates for all other DataWindow presentation styles and for DropDownListBoxes.

Value-Based Recording

By using value-based recording, Robot lets you record actions on a DataWindow and lets you play back those actions regardless of the position of records in the database. Value-based action parameters are written to a script as column/value pairs, using either the column number or the column name.

Testing an Expression Value of a DataWindow Property

In the following example, when you select a cell in the tabular DataWindow, Robot records a column/value pair to uniquely identify the row that was clicked.

If you select this cell ...

ID	Customer Name	Address	City	State	Zipcode	Telephone
1	ABC Manufacturing, Inc.	23 Broadway	Dayton	OH	44234	216 888 1234
2	Acme Metal Working Corp.	909 East Greenway	Concord	NH	09634	603 742 2934
3	A1 Woodworking Ltd.	542 Great Avenue	Needham	MA	02165	617 444 8439
4	Barrymore Company	B431 Main Street	New York	NY	10015	212 783 9000

... Robot writes this in the script.

```
DataWindow Click,  
"Name=dw_customer_info;\;Name=customername",  
"ColName=customerid;Value=1"
```

The column/value pair identifies the clicked row.

This script shows that Robot:

- Recorded a Click in the DataWindow dw_customer_info
- In the column customername
- In the row with a customerid of 1

Robot detects which columns in a DataWindow are key columns, and then uses the key columns in the column/value pairs. If there are no key columns, Robot uses as many column/value pairs as necessary to uniquely identify the clicked row, starting with the leftmost column.

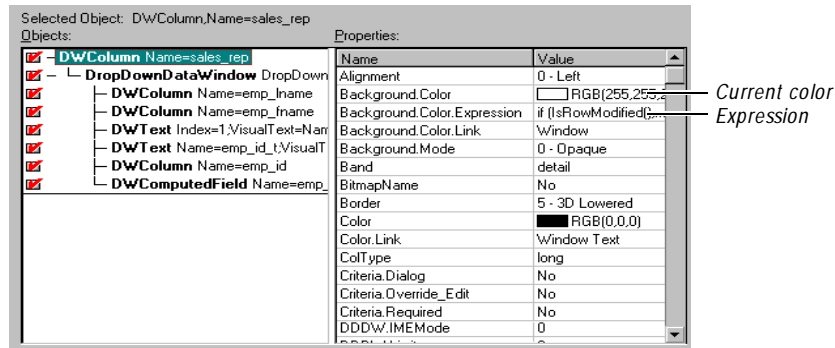
Testing an Expression Value of a DataWindow Property

In PowerBuilder, the value of any property of a DataWindow or DataWindow sub-object can be an expression. For example, you can have the background color of a DataWindow dynamically vary based on a formula or comparison.

If a property's value comes from an expression, an Object Properties verification point performed on a DataWindow returns both the current value and an **.Expression** property whose value is the expression. In this case, you can test the actual result and the expression itself.

For example, if the value of **Background.Color** is an expression, an Object Properties verification point returns both:

- ▶ The **Background.Color** property with a value that is the actual result of the expression.
- ▶ The **Background.Color.Expression** property with a value that is the expression. (The value of an **.Expression** property is always a string.)

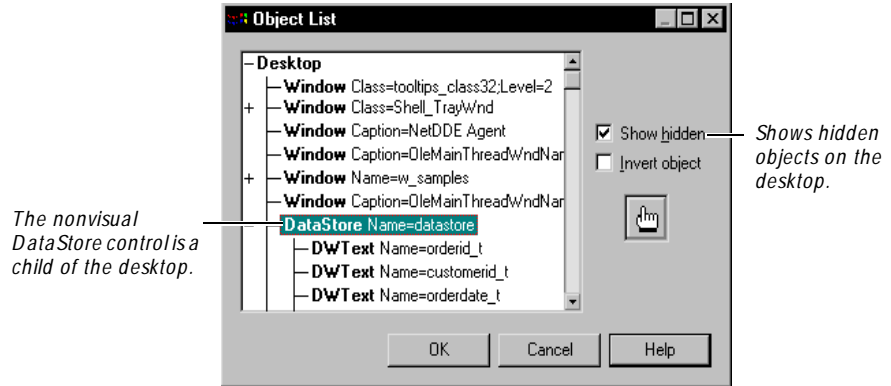


Expressions that are used against objects within the detail band of a DataWindow are reevaluated for every row. When Robot returns the values of such properties, they are based on the state of the current row of the DataWindow. If you select a different row and recapture the properties, you may get different values.

Testing DataStore Controls and Hidden DataWindows

You can test the properties and data of PowerBuilder objects even though they are not visible in the application. By selecting from a list of all objects on the Windows desktop, you can test:

- ▶ Nonvisual DataStore controls. (DataStore controls always appear as direct children of the Windows desktop.)
- ▶ Hidden DataWindows (Visible property is False).



For information about selecting objects from the Windows desktop, see *Selecting and Identifying the Object to Test* on page 6-10.

Capturing Data in a DropDownDataWindow/ListBox

To capture the data in a DropDownDataWindow or DropDownListBox, use the Object Data verification point as follows:

1. Start creating an Object Data verification point.
2. In the Select Object dialog box, drag the Object Finder tool to the DWColumn that contains the data.

If the DWColumn has a child dropdown, the TestTip shows **DWColumn (Contains DropDownDataWindow)** or **(Contains DropDownListBox)**.

3. Release the mouse button. If the Select Object dialog box still appears, click **OK**.

The Object Data Tests dialog box appears.

4. To capture the data stored in the *child dropdown* of the DataWindow or ListBox, select the **DropDown Contents** data test.

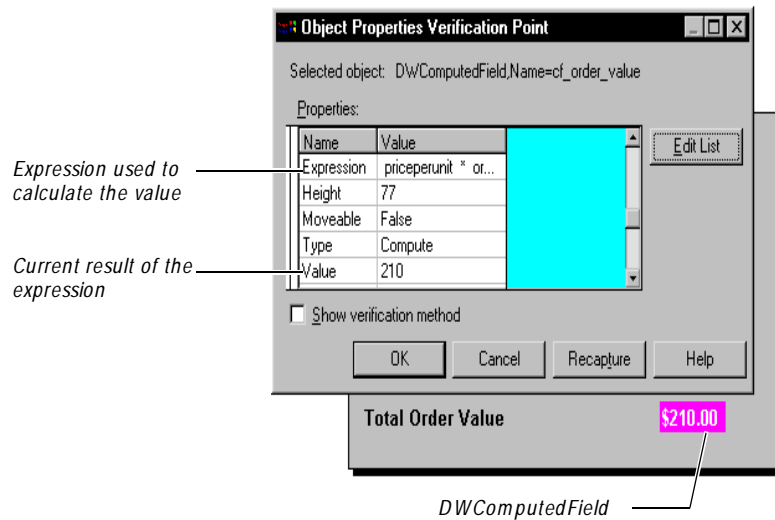
To capture the data in the *DataWindow*, select the **DataWindow Contents** test. (If the DWColumn does not contain a child dropdown, the data in the DataWindow is captured automatically after step 3.)

5. Click **OK**.
6. Continue creating the verification point as usual.

Testing the Value of a DataWindow Computed Field

The Object Properties verification point supports a Value property for a computed field (DWComputedField) within a DataWindow. The Value property contains the current result of the expression assigned to the computed field.

You can also test the Expression property, which contains the expression used to calculate the value of the computed field.



Testing PeopleTools Applications

This chapter describes the support that Rational Robot provides for testing PeopleTools applications. It includes the following topics:

- ▶ About Robot support for PeopleTools applications
- ▶ Verifying that the PeopleTools extension is loaded
- ▶ Testing a component's properties
- ▶ Testing a component's data
- ▶ PeopleTools commands

About Robot Support for PeopleTools Applications

Rational Robot provides comprehensive support for testing applications built with PeopleTools versions 6.0 through 8.x. With its Object Testing technology, Robot examines data and properties that are not visible to the user. Robot uses Object-Oriented Recording to recognize a PeopleTools component by its field name in the database.

Robot provides end-to-end automated testing of your PeopleTools applications, as follows:

If you have purchased	Then
A PeopleTools application	Your PeopleTools package includes a PeopleTools-only version of Robot.
Rational Robot	Support for testing PeopleTools applications with Robot is available in addition to all the other features of Robot.
Rational TestFoundation	You have access to a testing methodology and a comprehensive collection of pre-recorded scripts for PeopleTools applications.

You can use Robot to test the following PeopleTools components:

Grids	Tab controls
Menu objects	Toolbars
Navigator Panels	Trees
Panels	Tree Views
Spin controls	

Verifying that the PeopleTools Extension Is Loaded

To test PeopleTools applications, you should first verify that the Robot PeopleTools extension is loaded in Robot.

To verify that the extension is loaded:

1. Start Robot.
2. Click **Tools** → **Extension Manager**.
3. Verify that **PeopleTools** is selected. If not, select it.
4. To improve the performance of Robot, clear the check boxes of all environments that you do not plan to test.
5. Exit Robot.

The next time you start Robot, only the extensions for the selected environments are loaded.

Testing a Component's Properties

There are two methods that you can use to test the properties (attributes) of a PeopleTools component:

- ▶ **Object Properties verification point** – Use to test properties while recording a script.

NOTE: To test the properties of the entire panel, point to the window title bar when selecting the object to test.

- ▶ **Object Scripting commands** – Use to test properties programmatically while editing a script. (For information, see the *SQABasic Language Reference*.)

Testing a Component's Data

You can use the Object Data verification point to test the data in PeopleTools data-aware components.

NOTE: To test the data in the entire panel, point to the panel when selecting the object to test.

You can use the Object Data Test Definition to define user data tests. (For information, see Appendix B, *Working with Data Tests*.)

PeopleTools Commands

The following commands are included in the SQABasic scripting language to support the PeopleTools development environment. (For detailed information about these commands, see the *SQABasic Language Reference*.)

Command	Description
PSGrid	Performs an action on a PeopleTools grid.
PSGridHeader	Performs an action on a column header in a PeopleTools grid.
PSGridHeaderVP	Creates a verification point on a column header in a PeopleTools grid.
PSGridVP	Creates a verification point on a PeopleTools grid.
PSMenu	Performs an action on a PeopleTools menu object.
PSMenuVP	Creates a verification point on a PeopleTools menu object.
PSNavigator	Performs an action on a PeopleTools Navigator window or a navigator map in the PeopleTools Business Process Designer.
PSNavigatorVP	Creates a verification point on a PeopleTools Navigator window or a navigator map in the PeopleTools Business Process Designer.
PSPanel	Performs an action on a PeopleTools panel.
PSPanelVP	Creates a verification point on a PeopleTools panel.
PSSpin	Performs an action on a PeopleTools spin control.

Testing PeopleTools Applications

(Continued)

Command	Description
PSSpinVP	Creates a verification point on a PeopleTools spin control.
PSTree	Performs an action on a PeopleTools tree object.
PSTreeHeader	Performs an action on a column header in a PeopleTools tree object.
PSTreeHeaderVP	Creates a verification point on a column header in a PeopleTools tree object.
PSTreeVP	Creates a verification point on a PeopleTools tree object.

▶ ▶ ▶ Part VII

Appendixes

▶ ▶ ▶ A P P E N D I X A

Working With Toolbars

Each component of Rational Robot has two default toolbars:

- ▶ **Standard** – Contains buttons for choosing the most frequently used commands for that component.
- ▶ **Tools** – Contains buttons for choosing other components.

Robot has several additional toolbars: GUI Record, GUI Playback, GUI Insert, VU Record, and VU Insert.

All toolbar buttons correspond to menu commands. Click a toolbar button to immediately access the menu command. The toolbar buttons are dynamic. This means that some toolbar buttons are enabled only when you select related menu or toolbar commands.

Viewing Information About Toolbar Buttons

There are several ways to view information about a toolbar button and its corresponding menu command:

- ▶ To see the name of the button in a yellow ToolTip, point to the button and pause.
- ▶ To see a brief description in the status bar, point to the button or menu command.
- ▶ To see more detailed information about the button or menu command, do one of the following:
 - Point to the button or menu command and press **F1**.
 - Click the **Help Pointer** icon on the right side of the Standard toolbar, and then point to the button or menu command and click the mouse button.



Displaying Toolbars

To display or hide a toolbar:

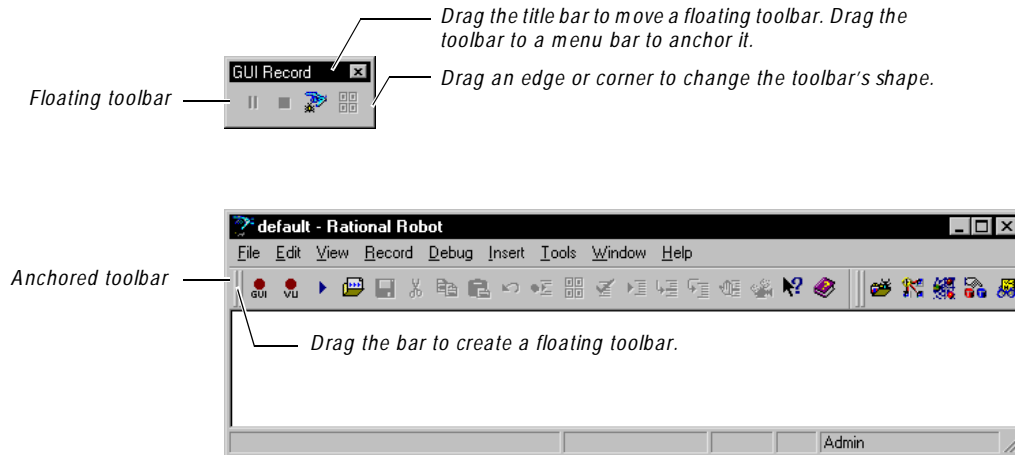
- ▶ Click **View** → **Toolbars**, and then click the name of the toolbar you want to display or hide.

A check mark appears in front of the name of each displayed toolbar.

Anchoring and Floating Toolbars

The Standard and Tools toolbars are anchored (docked) within each component's main window below the menu bar. However, you can drag an anchored toolbar from within a window and make it a floating (undocked) toolbar, which you can position and resize independently of the main window. When you do this, the toolbar remains visible even when you minimize a component or when the component is hidden behind another application. You can also drag a floating toolbar and anchor it inside of the window.

Floating toolbars are always on top of all other windows. This ensures that they are never hidden. The following figure shows a floating toolbar and an anchored toolbar.



Setting Toolbar Options

To set the toolbar options:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. In the **Toolbars** tab, select or clear the appropriate check boxes:
 - Show ToolTips** – Displays a ToolTip when you point to a button and pause.
 - Cool Look** – Changes the appearance of the toolbar buttons so that they have no borders. It does not change the behavior of the buttons.
 - Large Buttons** – Changes the size of the toolbar buttons.
3. Click **OK**.

Adding, Deleting, and Moving Toolbar Buttons

To add, delete or move a toolbar button:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. Click the **Commands** tab.
3. To add a button, click a menu name from the **Categories** list. Each name in the list represents a menu in the menu bar. Click a button to see its description. Drag the button to the toolbar. Make sure you release the mouse button within the toolbar.
4. To delete a button, drag it anywhere outside the toolbar.
5. To move a button, drag it to a new location.
6. Click **OK**.

Creating Your Own Toolbar

To create a custom toolbar that contains just the buttons you want:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. Click the **Toolbars** tab.
3. Click **New**.

4. Type the name for the new toolbar and click **OK**.
5. Click the **Commands** tab.
6. Click a menu name from **Categories**.
7. Click a button to see its description. Drag the button to the new toolbar. Make sure you release the mouse button within the new toolbar.
8. Repeat steps 6 and 7 until you have finished adding buttons.
9. Click **OK**.

Resetting and Deleting Toolbars

To restore a default toolbar to its original configuration or to delete a custom toolbar:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. In the **Toolbars** tab, do one of the following:
 - To reset a default toolbar to its original configuration, highlight the toolbar in the list and click **Reset**.
 - To delete a custom toolbar, highlight the toolbar in the list and click **Delete**.

The name of this button changes depending on the type of toolbar that is selected.
3. Click **OK**.

▶▶▶ APPENDIX B

Working with Data Tests

This appendix describes how to work with data tests, which are used with the Object Data verification point. This appendix includes the following topics:

- ▶ About data tests
- ▶ An example of a data test
- ▶ Creating or editing a custom data test
- ▶ Copying, renaming, or deleting a data test

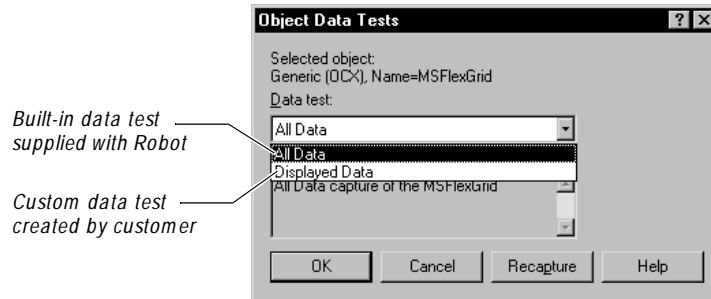
For information about the Object Data verification point, see *Object Data verification point* in the Robot Help Index.

About Data Tests

The Object Data verification point supports data tests to capture the data in objects. In general, there are two types of data tests:

- ▶ **Built-in data tests** – Delivered with Robot. Built-in tests are available to all users no matter which repository they are using. These tests cannot be edited, renamed, or deleted, but they can be copied and viewed.
- ▶ **Custom data tests** – Created within your organization. Each custom data test is stored in the repository that was active when the data test was created. If you switch to a different repository, the custom data tests are not available unless you recreate them in the new repository. Custom data tests can be edited, renamed, and deleted.

When you use the Object Data verification point to test an object that has more than one data test, Robot displays a list of all of the object's data tests (built-in and custom).



You can select any of the tests in the list depending on what you want to capture and test. For example, you might have data tests defined for a grid that let you:

- ▶ Capture all of the data in the grid including fixed columns and rows, even if the data is not visible on the screen.
- ▶ Capture only selected or displayed data in the grid.

Before you create a custom data test, make sure you have the following:

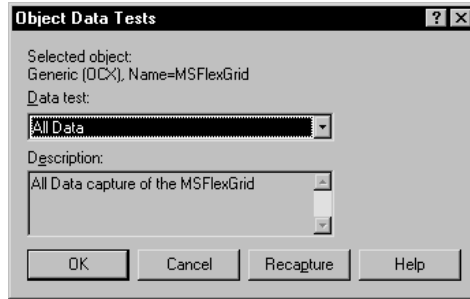
- ▶ Access to the documentation or Help that came with the object that you want to test.
- ▶ A good understanding of the object's properties and how the properties relate.
- ▶ A good understanding of the Object Data verification point. For information, see *Object Data verification point* in the Robot Help Index.

An Example of a Data Test

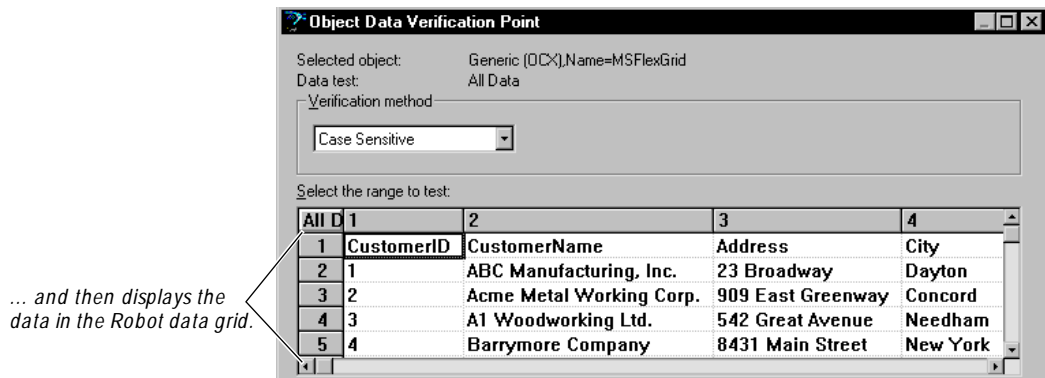
One way to understand how to create a data test is to look at a built-in data test. This section explains the *All Data* test for the MSFlexGrid control.

What the *All Data* Test Does

When you create an Object Data verification point on the MSFlexGrid, you can select the All Data test in the dialog box that appears, as shown in the following figure.



When you use this data test, Robot captures the data from every cell in the MSFlexGrid control, as shown in the following figures.

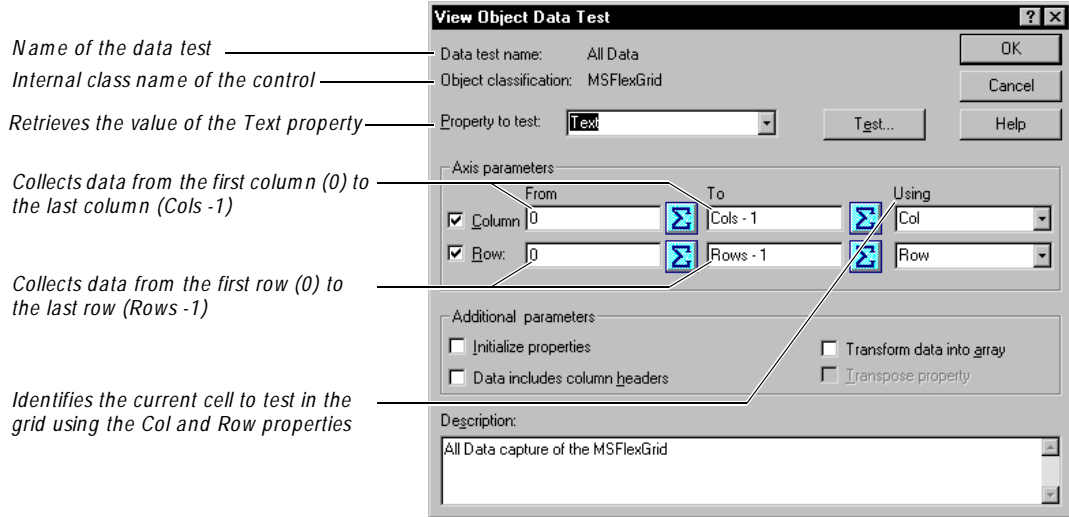


The Definition of the All Data Test

You cannot edit the *All Data* test, because it is a built-in test. However, you can view the test's definition for the MSFlexGrid by looking at the data test in the View Object Data Test dialog box.

Working with Data Tests

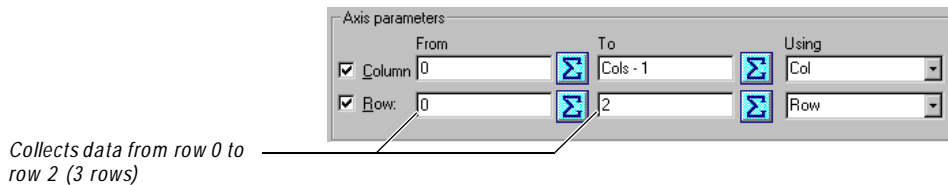
The following figure shows the main information in the dialog box:



Because the MSFlexGrid control is a zero-based grid, the numbering for columns and rows actually begins with zero. Therefore, the **From** box contain 0 as the first column and row, and the **To** box subtracts 1 from the total number of columns and rows.

Changing a Data Test Definition

Suppose you want to test only the first three rows in the control instead of all the rows. You could do this by making a copy of the *All Data* test, and then editing the copy so that the rows range from 0 to 2.



First 3 rows of data —

3 row	1	2	3	4
1	CustomerID	CustomerName	Address	City
2	1	ABC Manufacturing, Inc.	23 Broadway	Dayton
3	2	Acme Metal Working Corp.	909 East Greenway	Concord

Creating or Editing a Custom Data Test

Data tests must be created before you begin to test your application. The tests that you create are stored in the repository that is currently active when you create the tests. If you switch to a different repository, the data tests will not be available unless you recreate the data test in the new repository.

You can edit any custom data test. If you change the parameters of an existing data test, it affects the behavior of all verification points that depend on the data test.

To create or edit a custom data test:

1. Display the object for which you want to create the data test.
2. In Robot, click **Tools** → **Object Data Test Definition**.
3. Click **Select** to open the Select Object dialog box.
4. Select the object for which you want to create the data test in one of the following ways:
 - Drag the Object Finder tool over the object and release the mouse button.

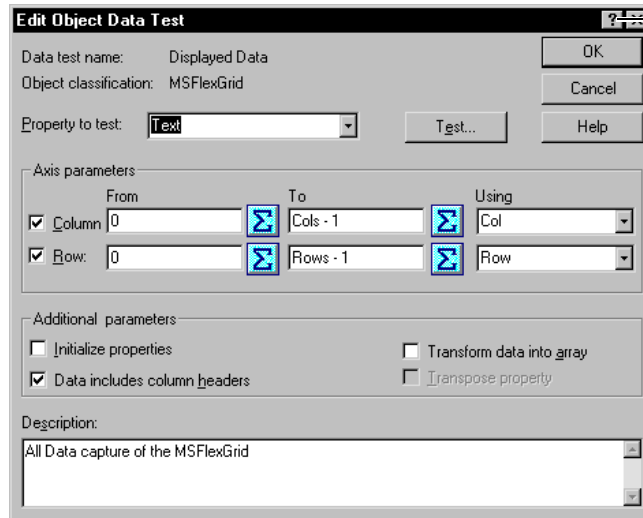
As you move the Object Finder tool over an object, the object type appears in the yellow TestTip.
 - Click **Browse** to open the Object List dialog box, select the object from the list, and click **OK**.

The Object List dialog box shows a hierarchical list of all objects on the Windows desktop, including hidden objects.
5. If the Select Object dialog box is still open, click **OK** to close it.


The object classification of the selected object and its data tests appear in the Object Data Test Definition dialog box.

If the object is Unknown (not defined), the Define Object dialog box appears. Select an object type and click **OK** to open the Object Data Test Definition dialog box. (For information about defining an object, see *Defining Unknown Objects During Recording* on page 4-21.)
6. Do one of the following to display the Create/Edit Object Data Test dialog box:
 - To create a new test, type a name (50 characters maximum) in the **Data test name** box and click **New**.
 - To edit a custom test, select the test from the list and click **Edit**.
 - To copy a test and edit the copy, select the test and click **Copy**. Type the new name and click **OK**. Then, click **Edit**.

For example, if you copied the *All Data* test to a new test named *Displayed Data*, and clicked **Edit**, the following dialog box would appear:



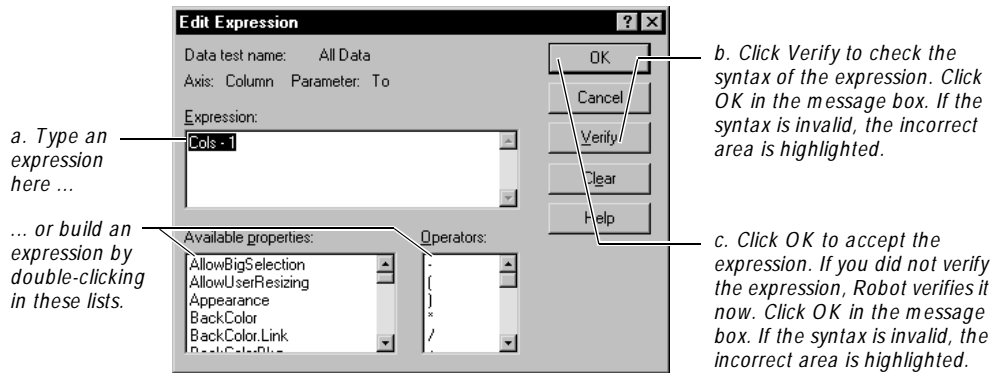
For detailed information about an item, click the question mark, and then click the item.

7. Select a property from the **Property to test** list. This property is the one whose values you want to capture in the data test.
8. Select the **Column** check box to add parameters for the vertical axis. Select the **Row** check box to add parameters for the horizontal axis.
9.  Type an expression in the **From** and **To** boxes, or click the **Expression** button to the right of each box to build the expression.

An expression is a single value or property, or a combination of values, properties, and operators.

If you click the **Expression** button, the Edit Expression dialog box appears.

Do the following if you clicked **Expression**:



10. In the **Using** box (in the Create/Edit Object Data Test dialog box), type a property or select it from the list to further define the property that you are capturing and testing.

The **Using** box specifies what property Robot will modify to affect its iteration. For example, to iterate from row 0 to row Rows-1, Robot will set the Row property.

11. Select the check boxes under **Additional parameters** as needed.
12. In the **Description** box, type a description that indicates what the data test does.
13. Optionally, click **Test** to do the following:
 - Verify the syntax of the data test before you save it.
 - If the syntax is correct, watch Robot perform the data test on the selected object.

When the test has ended, Robot opens a dialog box with the captured data. Click **OK** to close the dialog box.

14. Click **OK** to save the test and automatically verify it.

If the syntax of the expression is incorrect, the incorrect area is highlighted so you can correct it and then resave the test.

Copying, Renaming, or Deleting a Data Test

You can copy any built-in or custom data test to back it up or to create a new test from an existing one. When you copy a data test, you can use the test only for objects of the class for which the original data test was created.

You can rename any custom data test. However, scripts that contain the data test under its original name will fail on playback unless you change the name in the scripts.

You can delete any custom data test. However, scripts that contain the data test will fail on playback unless you delete the test from the scripts.

To copy, rename, or delete a data test:

1. Click **Tools** → **Object Data Test Definition**.
2. Select the data test.
3. Do one of the following:
 - To copy the test, click **Copy**. Type a new name (50 characters maximum) and click **OK**.
 - To rename the test, click **Rename**. Type a new name (50 characters maximum) and click **OK**.
 - To delete the test, click **Delete**. Click **OK** to confirm the deletion.

If you renamed or deleted the data test, be sure to rename it or delete it in any scripts that use that data test.

▶ ▶ ▶ A P P E N D I X C

Standard Datapool Data Types

This appendix contains:

- ▶ A table of standard data types
- ▶ A table of minimum and maximum ranges for the standard data types

Standard Data Type Table

Data types supply datapool columns with their values. You assign data types to datapool columns when you define the columns in the Datapool Specification dialog box.

The standard data types listed in the following table are included with your Rational Test software. Use these data types to help populate the datapools that you create.

The standard data types (plus any user-defined data types that you create) are listed in the Datapool Specification dialog box under the heading **Type**. **Type** and the other datapool column definitions (such as **Length** and **Interval**) referenced in the following table are some of the definitions that you set in this dialog box.

Note that related data types (such as cities and states) are designed to supply appropriate pairings of values in a given datapool row. For example, if the Cities - U.S. data type supplies the value Boston to a row, the State Abbrev. - U.S. data type supplies the value MA to the row.

Standard Datapool Data Types

Standard data type name	Description	Examples
Address - Street	Street numbers and names. No period after abbreviations.	20 Maguire Road 860 S Los Angeles St 8th Fl 75 Wall St 22nd Fl
Cities - U.S.	Names of U.S. cities.	Lexington Cupertino Raleigh
Company Name	Company names (including designations such as Co and Inc where appropriate).	Rational Software Corp TSC Div Harper Lloyd Inc Sofinnova Inc
Date - Aug 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma (,) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>Oct 8, 1997 Jun 17, 1964 Nov 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"Oct 8, 1997" "Jun 17, 1964" "Nov 10, 1978"</p>
Date - August 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma (,) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>October 8, 1997 June 17, 1964 November 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"October 8, 1997" "June 17, 1964" "November 10, 1978"</p>

(Continued)

Standard data type name	Description	Examples
Date - MM/DD/YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>10/08/97 06/17/64 11/10/78</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows:</p> <p>"10/08/97" "06/17/64" "11/10/78"</p>
Date - MM/DD/YYYY	<p>Dates in the format shown.</p> <p>To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>10/08/1997 06/17/1964 11/10/1978</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows:</p> <p>"10/08/1997" "06/17/1964" "11/10/1978"</p>
Date - MMDDYY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>100897 061764 111078</p>
Date - MM-DD-YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>10-08-97 06-17-64 11-10-78</p>

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Date - MMDDYYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.	10081997 06171964 11101978
Date - MM-DD-YYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.	10-08-1997 06-17-1964 11-10-1978
Date - YYYY/MM/DD	Dates in the format shown. To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 19000101 and Maximum to 20501231.	1997/10/08 1964/06/17 1978/11/10 If the slash is the delimiter, the values are stored in the datapool as follows: "1997/10/08" "1964/06/17" "1978/11/10"
Date - YYYYMMDD	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 19000101 and Maximum to 20501231.	19971008 19640617 19781110
Date, Julian - DDDYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 00100 and Maximum to 36599.	28197 16964 31478
Date, Julian - DDDYYYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 0011900 and Maximum to 3652050.	2811997 1691964 3141978

(Continued)

Standard data type name	Description	Examples
Date, Julian - YYDDD	<p>Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032.</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 00001 and Maximum to 99365.</p>	<p>97281 64169 78314</p>
Date, Julian - YYYYDDD	<p>Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 1900001 and Maximum to 2050365.</p>	<p>1997281 1964169 1978314</p>
Float - X.XXX	<p>Positive and negative decimal numbers in the format shown.</p> <p>Set Length to the number of decimal places to allow (up to 6).</p> <p>Set Minimum and Maximum to the range of numbers to generate.</p> <p>To generate numbers with more than 9 digits (the maximum allowed with the Integers - Signed data type), use the Float - X.XXX data type and set Decimals to 0.</p>	<p>243.63918 -95.99 155075028157503</p>
Float - X.XXXE+ NN	<p>Positive and negative decimal numbers in the exponential notation format shown.</p> <p>Set Length to the number of decimal places to allow (up to 6).</p> <p>Set Minimum and Maximum to the range of numbers to generate.</p>	<p>4.0285177E+ 068 -3.2381443E+ 024 8.8373255E+ 119</p>
Gender	<p>Either M or F, with no following period.</p>	<p>M F</p>
Hexadecimal	<p>Hexadecimal numbers.</p>	<p>1d6b77 ff 3824e7d</p>

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Integers - Signed	<p>Positive and negative whole numbers. This is the default data type.</p> <p>To include negative numbers in the list of generated values, set Minimum to the lowest negative number you want to allow.</p> <p>Maximum range:</p> <ul style="list-style-type: none"> ▶ Minimum = -999999999 (-999,999,999) ▶ Maximum = 999999999 (999,999,999) <p>For larger numbers, use a float data type.</p> <p>If you do not specify a range, the default range is 0 through 999,999,999.</p> <p>Use this data type to generate unique data in a datapool column (for example, when you need a “key” field of unique data). You can also use Read From File and user-defined data types to generate unique data.</p>	<p>1349 -392993 441393316</p>
Name - Middle	<p>Masculine and feminine middle names.</p> <p>If the middle name is preceded by a field with masculine or feminine value (such as a masculine or feminine first name), the middle name is in the same gender category as the earlier field.</p>	<p>Richard Theresa Julius</p>
Name - Prefix (e.g., Mr)	<p>Mr or Ms, with no following period.</p> <p>If the name prefix is preceded by a field with masculine or feminine value (such as a masculine or feminine gender designation), the name prefix is in the same gender category as the earlier field.</p>	<p>Mr Ms</p>
Names - First	<p>Masculine and feminine first names.</p> <p>If the first name is preceded by a field with masculine or feminine value (such as a masculine or feminine name prefix), the first name is in the same gender category as the earlier field.</p>	<p>Richard Theresa Julius</p>
Names - Last	<p>Surnames.</p>	<p>Swidler Larned Buckingham</p>

(Continued)

Standard data type name	Description	Examples
Names - Middle Initial	Middle initials only, with no following period.	B M L
Packed Decimal	A number where each digit is represented by four bits. Digits are non-printable. Note that commas and other characters that may be used to represent a packed decimal number may cause unpredictable results when the datapool file is read.	Non-printable digits.
Phone - 10 Digit	Telephone area codes, appropriate exchanges, and numbers.	7816762400 4123818993 5052658498
Phone - Area Code	Telephone area codes. To generate correct area code lengths, set Length to 3.	781 412 505
Phone - Exchange	Telephone exchanges. To generate correct exchange lengths, set Length to 3.	676 381 265
Phone - Suffix	Four-digit telephone numbers (telephone numbers without area code or exchange). To generate correct telephone number suffix lengths, set Length to 4.	2400 8993 8498
Random Alphabetic String	Strings of random upper case and lower case letters. Length determines the number of characters generated.	AQSEFuOZUIUpAGsEM DESieAiRFiEqiEIDiicEw edEIDiIciseWsDIEdGP
Random Alphanumeric String	Strings of random upper case and lower case letters and digits. Length determines the number of characters generated.	AYcHI8WmeMeM0AK4 Hsk9vGAQU79esDE 7Eeis93k4ELXie7S32siDI4E

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Read From File	<p>Assigns values from an ASCII text file to the datapool column. For example, you could export a database column to a text file, and then use this data type to assign the values in the file to a datapool column.</p> <p>You can use this data type to generate unique data. You can also use the Integers - Signed and user-defined data types to generate unique data.</p> <p>For information about using this data type, see <i>Creating a Column of Values Outside Rational Test</i> on page 10-39.</p>	Any values in an ASCII text file.
Space Character	An empty string.	""
State Abbrev. - U.S.	Two-character state abbreviations.	MA CA NC
String Constant	A constant with the value of Seed . The datapool column is filled with this one alphanumeric value.	1234 AAA 1b1b
Time - HH.MM.SS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	00.00.00 (midnight) 11.14.38 21.44.19
Time - HH:MM:SS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To include the colons (:) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	00:00:00 (midnight) 11:14:38 21:44:19 If the colon is the delimiter, the values are stored in the datapool as follows: "00:00:00" (midnight) "11:14:38" "21:44:19"
Time - HHMMSS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	000000 (midnight) 111438 214419

(Continued)

Standard data type name	Description	Examples
Zip Code - 5 Digit	Five-digit U.S. postal zip codes. To generate the correct zip code lengths, set Length to 5.	02173 95401 84104
Zip Code - 9 Digit	Nine-digit U.S. postal zip codes.	021733104 954012694 841040190
Zip Code - 9 Digit with Dash	Nine-digit U.S. postal zip codes with a dash between the fifth and sixth digits.	02173-3104 95401-2694 84104-0190
Zoned Decimal	Zoned decimal numbers.	3086036 450 499658196

Data Type Ranges

The following table shows the minimum and maximum ranges for the standard data types:

Type of range	Limitation
Maximum hours	23
Maximum minutes	59
Maximum seconds	59
Maximum two-digit year	99
Maximum four-digit year	9999
Maximum months	12
Minimum six-digit date	010100 (January 1, 00)
Maximum six-digit date	123199 (December 31, 9999)
Minimum eight-digit date	01010000 (January 1, 0000)
Maximum eight-digit date	12319999 (December 31, 9999)
Minimum negative integer (Integers - Signed)	-99999999 (-999,999,999)

Standard Datapool Data Types

(Continued)

Type of range	Limitation
Maximum positive integer (Integers - Signed)	999999999 (999,999,999)
Maximum decimal places (Float data types)	6
Male/Female title	Mr, Ms
Gender designation	M, F

▶▶▶ A P P E N D I X D

Rational Robot Command-line Options

You can use the Rational Robot command-line options to log in, open a script, and play back the script.

SYNTAX

```
sqa7robo.exe [scriptname] [/user userid] [/password password]  
             [/repository repopath] [/project projectname] [/play]  
             [/purify] [/quantify] [/coverage] [/close] [/nolog]
```

Syntax Element	Description
sqa7robo.exe	Rational Robot executable file.
<i>scriptname</i>	Name of the script to run.
/user <i>userid</i>	User name for log in.
/password <i>password</i>	Optional password for log in. Do not use this parameter if there is no password.
/repository <i>repopath</i>	Path of the repository that contains the script referenced in <i>scriptname</i> .
/project <i>projectname</i>	Name of the project that contains the script referenced in <i>scriptname</i> .
/play	If this keyword is specified, plays the script referenced in <i>scriptname</i> . If not specified, the script opens in the editor.
/purify	Used with /play. Plays back the script referenced in <i>scriptname</i> under Rational Purify.

Syntax Element	Description
/quantify	Used with /play. Plays back the script referenced in <i>scriptname</i> under Rational Quantify.
/coverage	Used with /play. Plays back the script referenced in <i>scriptname</i> under Rational PureCoverage.
/close	Closes Robot after playing back the script.
/nolog	Does not log any output while playing back the script.

COMMENTS

Use a space between each keyword and between each variable.

If a variable contains spaces, enclose the variable in quotation marks.

If you log the output (by omitting /nolog), then the Robot log options (set in the GUI Playback Options dialog box) determine whether the default log information is used or whether you are prompted at the start of playback.

If you intend to run Robot unattended in batch mode, be sure to specify the following options to get past the Rational Repository Login dialog box:

```

/user userid
/password password
/repository repopath
/project projectname

```

EXAMPLE

```

sqa7robo.exe VBMenus /user admin /repository "c:\Sample Files\Repo"
/project Default /play /close

```

In this example, the user “admin” opens the script “VBMenus”, which is in the project “Default” and in the repository “c:\Sample Files\Repo”. The script is opened for playback, and then it is closed when playback ends. The results are logged.

▶ ▶ ▶ A P P E N D I X E

Working with Manual and External Scripts

This appendix explains how to create and run manual and external scripts in TestManager. It includes the following topics:

- ▶ About manual scripting
- ▶ Working with manual scripts in TestManager
- ▶ Working with manual scripts on the Web
- ▶ Working with external scripts

About Manual Scripting

The manual scripting feature in TestManager lets you create and run scripts for tests that you cannot automate. A **manual script** is a set of testing instructions to be run by a human tester. The script can consist of steps and verification points that you type into an editor.

A **step** is an instruction to be carried out by the tester when a manual script is run. This could be as simple as a single sentence (such as “Reboot the computer”) or as complex as a whole document. In general, a step consists of one or two sentences.

Within a manual script, a **verification point** is a question about the state of the application (for example, “Did the application start?”). A verification point can consist of any amount of text but is likely to be one or two sentences, usually ending with a question mark.

After you create a manual script, you can run the script in TestManager or on the Web.

When you run a manual script, you perform each step and indicate whether each verification point passed or failed. You can then open the LogViewer and see the results. If all of the verification points passed, then the script passes. If any verification points failed, then the script fails.

As with other types of scripts, you can include your manual scripts in TestManager reports.

Example of a Manual Script

The following manual script contains five steps and four verification points.

- ▶ The steps are actions for you to take when you run the script.
- ▶ The verification points are questions for you to answer.

	Type	Note	Description
1			Loosen the 4 thumb screws on the rear of the case. Slide open the two side panels to reveal the drive cage and motherboard housing.
2			Detach the power supply, the power LED, keylock switch, hardware reset switch, power on, HDD LEF, hardware suspend, and speaker connectors from the motherboard. Slide out the motherboard cage.
3			Record the serial number on the motherboard.
4			Is the motherboard a BP6e model with a revision number greater than 1.0a?
5			Detach the processor fan power connector from the motherboard.
6			Remove the processor by pushing the lever on the socket outward and then pulling upward. The processor should shift slightly.
7			Inspect the CPU. Are any of the pins on the bottom side bent or missing?
8			Inspect the capacitors near the CPU socket. There should be 12 of them grouped together. Are any of them cracked?
9			Are any of the traces running from the CPU socket damaged (burned or severed)?

Working with Manual Scripts in TestManager

The following table lists the tasks that you perform to work with manual scripts in TestManager and where you can find information about each task.

Task	For more information, see
1. Set the default editor.	The next section, <i>Setting the Default Editor for Manual Scripts</i>
2. Plan the manual script.	<i>Planning a Manual Script</i> on page E-3
3. Create the manual script by entering the steps and verification points.	<i>Creating a Manual Script</i> on page E-4
4. Run the manual script.	<i>Running a Manual Script in TestManager</i> on page E-7

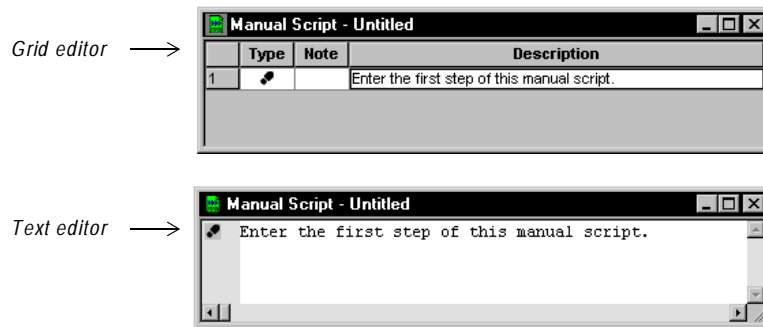
Setting the Default Editor for Manual Scripts

You can use either the grid editor or the text editor when you create a manual script. The grid editor is a structured editor that makes it easy to enter your steps and verifications points. The text editor is a free-form editor that makes it easy to manipulate text.

To set the default editor in TestManager:

1. Click **Tools** → **Options**. Click the **Manual Script** tab.
2. Click **Grid** or **Text**, and then click **OK**.

This setting takes effect the next time you create or open a manual script.



Planning and Creating a Manual Script

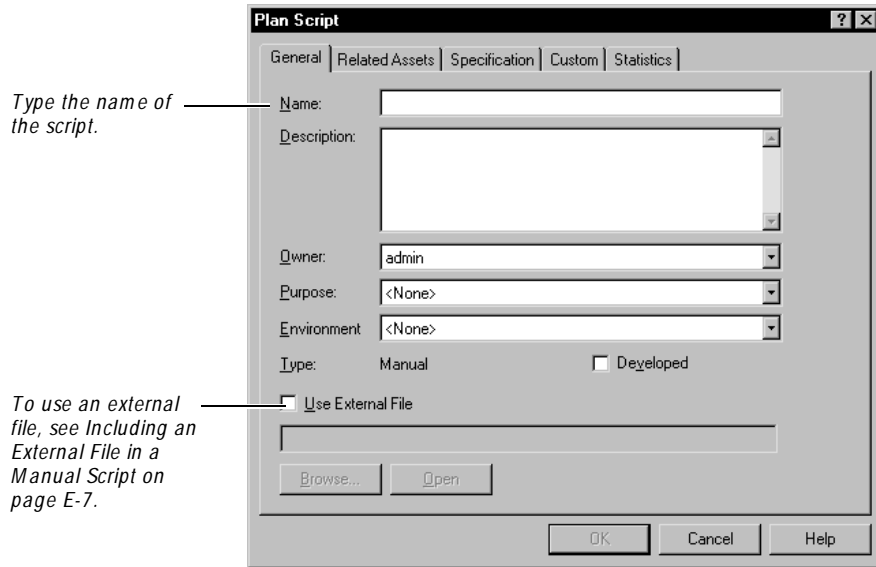
When you plan a manual script in TestManager, you name it and set its properties. You can also attach the script to a test requirement. After you plan the script, you open it and create the script by entering the steps and verification points.

Planning a Manual Script

To plan a manual script:

1. Do one of the following:
 - To plan a script and attach it to a requirement, open the Requirements Hierarchy. Right-click the requirement, and click **Plan** → **Manual Script**.
 - To plan a script without attaching it to a requirement, make sure that no requirement is selected. Click **File** → **Plan** → **Manual Script**.

2. In the Plan Script dialog box, type a name. Fill in other fields as appropriate.



3. Click **OK**. The script is now planned.

NOTE: For detailed information about planning a script, see *Planning Scripts* on page 2-10.

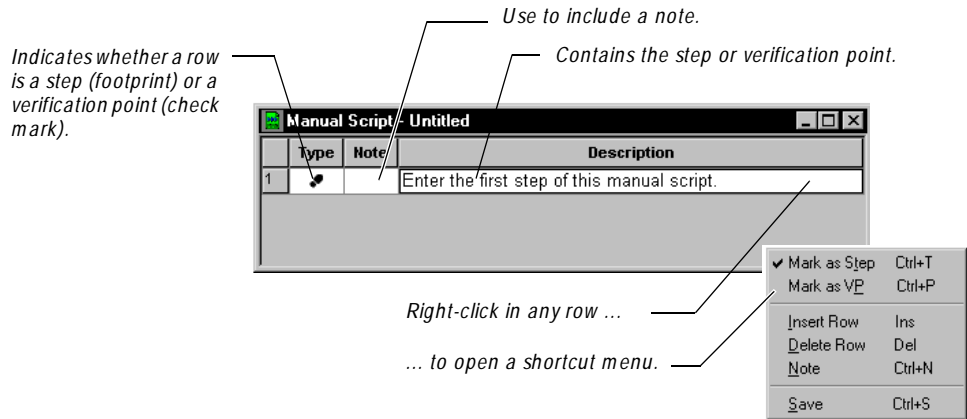
Creating a Manual Script

To create a manual script after you have planned it:

1. Do one of the following:
 - Right-click the script in either the Requirements Hierarchy or the Script Query window and click **Open**.
 - Click **File** → **Open** → **Script**. Click the script and click **OK**.

The script opens in the default editor.

NOTE: The grid editor is used in the following figure and in the rest of this procedure. For information about the text editor, see the next section, *Using the Text Editor*.



A manual script consists of items that can be steps (user actions) or verification points. Steps are in black type, and verification points are in blue type.

2. Type the first item. To create a new line in the same item, press CTRL+ ENTER. The footprint icon at the beginning of the line identifies this as a step. If this item is a verification point, click the Type cell to change the icon to a check mark.
3. To type a note, click the Note cell. Type the text and click **OK**. The Note icon appears in the cell. To view or edit the note, click the Note icon.
4. To start a new row, press the ENTER key or the down-arrow key. By default, the ENTER key creates a new row. To change the function of ENTER so that it creates a new line in the same item, open the TestManager Options dialog box. In the Manual Script tab, clear **Enter key moves focus to next row**.
5. Type the next step or verification point. If the text does *not* end with a question mark, then the type is set as a step (footprint) when you start the next row. If the text does end in a question mark, then the type is set as a verification point (check mark) when you start the next row. To change the type, click the Type cell.
6. Continue creating steps and verification points as needed.

7. When finished, click **File** → **Save**.

When you save the script, the **Developed** check box in the Script Properties dialog box is automatically selected. This setting is used by TestManager when you run a Development Coverage report. For information about coverage reports, see *Coverage Reports* on page 16-2.

8. Click **File** → **Close**.

NOTE: To create a manual script without first planning it, click **File** → **New** → **Manual Script**. After saving the script, you can set the properties.

Using the Text Editor

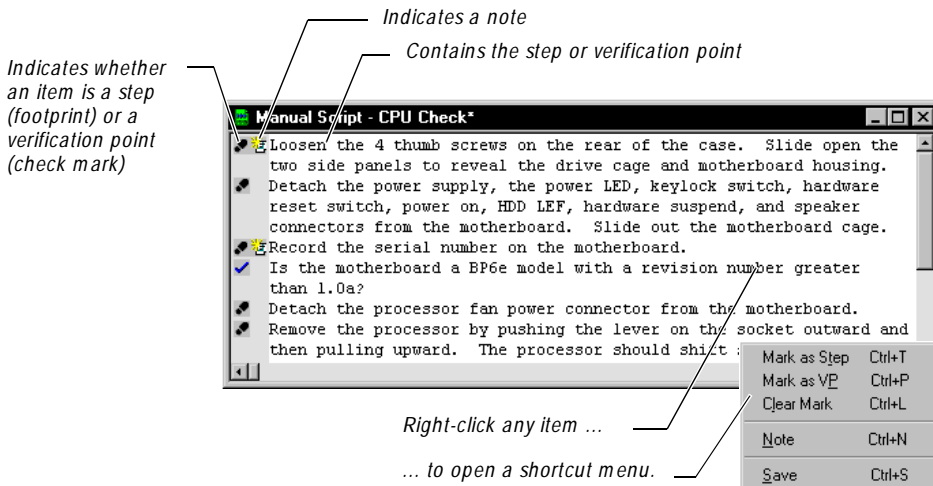
The text editor is a free-form editor that is useful when you want to manipulate text easily. For example, you can drag-and-drop words or lines in the text editor.

To set the text editor as the default editor in TestManager:

1. Click **Tools** → **Options**. Click the **Manual Script** tab.
2. Click **Text** and click **OK**.

The next time you create or open a manual script, it opens in the text editor.

In the text editor, you use a shortcut menu to mark items as steps and verification points, and to create and view notes.



The start of an item (step or verification point) is indicated by the footprint or check mark icon. All lines that do not begin with either of these icons are part of the previous item.

Including an External File in a Manual Script

Instead of typing the steps and verification points into the grid editor or text editor, you can include an external file when you *plan* the script. This file can contain all of the instructions to be used in the manual script. You can create this file with Notepad, Microsoft Word, or any other program for which there is a file association.

To include an external file when you plan a script:

1. Fill in the Plan Script dialog box. (For information, see *Planning a Manual Script* on page E-3.)
2. In the **General** tab, select **Use external file**.
3. Type the full path and name of the file that contains the instructions, or use **Browse** to locate the file.
4. Optionally, click **Open** to view the file.
5. Select the **Developed** check box.

This setting is used by TestManager when you run a Development Coverage report. For information about coverage reports, see *Coverage Reports* on page 16-2.

6. Click **OK**.

When you run the script, TestManager opens the external file so that you can follow the instructions. For more information, see *Running a Manual Script that Includes an External File* on page E-9.

NOTE: When you run a manual script that includes an external file, you can indicate the results of the entire script (Pass or Fail), but you cannot indicate the results of individual verification points.

Running a Manual Script in TestManager

If you used the grid or text editor when you created a manual script, you do the following when you run the script in TestManager:

- ▶ Indicate that you have performed each step.
- ▶ Indicate whether each verification point passed or failed.

For information, see the next section, *Running a Manual Script Created with the Grid or Text Editor*.

If you included an external file when you created the manual script, you do the following after you run the script in TestManager:

- ▶ Indicate whether the entire script passed or failed.

For information, see *Running a Manual Script that Includes an External File* on page E-9.

Running a Manual Script Created with the Grid or Text Editor

If you created the manual script using the grid or text editor, follow these steps to run the script:

1. Click **Tools** → **Options**. Click the **Manual Script** tab.
2. Select or clear **Log unchecked steps as warnings** and click **OK**.

If this is selected, **Warning** appears in the Result column of the log for each unchecked step. If this is cleared, the Result column is blank for each unchecked step. (**Complete** always appears for each checked step.)
3. Do one of the following:
 - Right-click the script in either the Requirements Hierarchy or the Script Query window and click **Run**.
 - Click **File** → **Run** → **Script**. Click the script to run and click **OK**.
4. Optionally, select **Show Only Verification Points** to show all of the verification points and hide all of the steps.
5. Optionally, click **All Pass** to set the default result to Pass for all of the verification points and to put checkmarks next to the steps.
6. Perform the steps and verification points indicated in the manual script.
 - For a step, select the check box in the Result column to indicate that you have performed the step. When you view the log, **Complete** appears in the Result column of each checked step.
 - For a verification point, click the cell in the Result column (where it says **None**) and click **None**, **Pass**, or **Fail**. When you view the log, the result appears in the Result column.

To view a note for a row, click the **Note** icon. You cannot edit notes.

7. To type a comment, click the Comment cell. Type the text and click **OK**. The Comment icon appears in the cell. To view or edit the comment, click the Comment icon.

When you view the log, the comment appears in the **Result** tab of the log event.

8. When finished, click **Done**.
9. Fill in the Specify Log Information dialog box and click **OK**.
10. Click **File** → **Close**.

To view the results, see *Viewing the Results in the LogViewer* on page E-10.

Example of Running a Manual Script

The following figure shows the results of running a manual script. When this manual script was run, the first verification point failed. The Comment icon indicates that there is a comment about the failure. When you view the log, you will be able to see the failure and the comment.

Type	Note	Description	Result	Comment
•	👉	Loosen the 4 thumb screws on the rear of the case. Slide open the two side panels to reveal the drive cage and motherboard housing.	☑	
•		Detach the power supply, the power LED, keylock switch, hardware reset switch, power on, HDD LEF, hardware suspend, and speaker connectors from the motherboard. Slide out the motherboard cage.	☑	
•	👉	Record the serial number on the motherboard.	☑	
✓		Is the motherboard a BP6e model with a revision number greater than 1.0a?	Fail	👉
•		Detach the processor fan power connector from the motherboard.	☑	
•		Remove the processor by pushing the lever on the socket outward and then pulling upward. The processor should shift slightly.	☑	

Indicates that the step was performed

Indicates that the verification point failed

Click to see a comment about the failure.

Running a Manual Script that Includes an External File

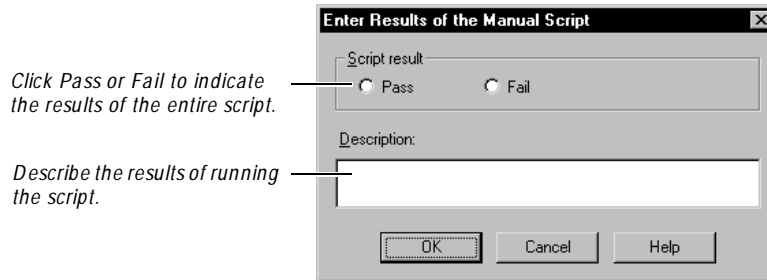
If you included an external file when you planned the script, follow these steps to run the script:

1. Do one of the following:
 - Right-click the script in either the Requirements Hierarchy or the Script Query window and click **Run**.
 - Click **File** → **Run** → **Script**. Click the script to run and click **OK**.

The external file opens.

2. Follow the instructions in the file and determine whether the script passed or failed.
3. Minimize or close the external file.
4. Fill in the **Enter Results of the Manual Script** dialog box.

You can indicate the results of the entire script but not of individual verification points.



5. When finished, click **OK**.
6. Fill in the Specify Log Information dialog box and click **OK**.

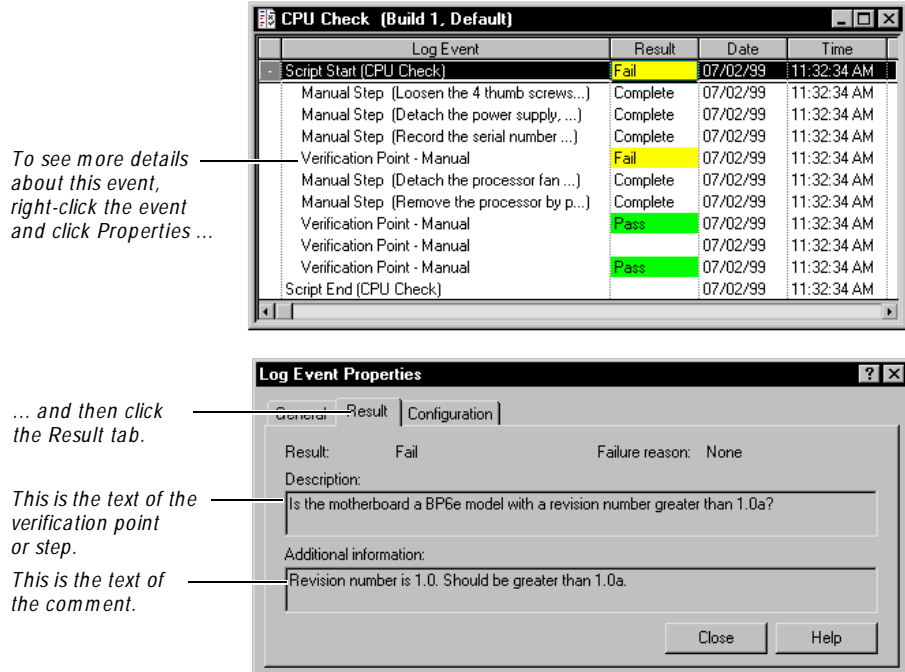
To view the result, see the next section, *Viewing the Results in the LogViewer*.

Viewing the Results in the LogViewer

To view the results of running a manual script:

1. Open the Asset Browser (click **View** → **Asset Browser**).
2. Expand the Builds tree until the log name appears.

3. Double-click the log to open it in the LogViewer.



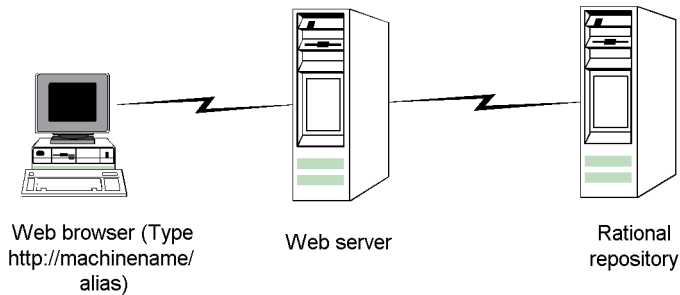
Working with Manual Scripts on the Web

After you create a manual script, you can run the script in TestManager or on the Web. When you run a manual script on the Web using the **Rational TestManager - Manual Script Execution** component, you use a Web browser over the Internet to access a Rational repository. This feature of TestManager lets you:

- ▶ Run a manual script remotely over the Internet.
- ▶ Remotely record results and add comments as you perform each task in a manual script.
- ▶ Include your manual script results in a TestManager coverage report.

Your Web server can be on the same computer as your Rational repository or on another computer. Using a Web browser, you type the *machinename* and *alias* of the Web server to run a manual script stored in a repository. For more information about setting up a Web browser and installing a Web server, see *Installing a Web Server* on page E-14 and *Setting Up a Web Browser* on page E-21.

The following figure shows a Web browser accessing the Web server to run a manual script stored in a shared repository:



Overview of Tasks

The following table lists the tasks you perform to run a manual script on the Web and where you can find information about each task.

Task	For more information, see
1. Check your software requirements.	The next section, <i>Software Requirements</i> on page E-13
2. Install a Web Server.	<i>Installing a Web Server</i> on page E-14
3. Configure either Microsoft Internet Information Server or the Microsoft Personal Web Server.	<i>Configuring a Microsoft Internet Information Server</i> on page E-16 or <i>Configuring a Microsoft Personal Web Server</i> on page E-19
4. Set up a Web browser.	<i>Setting Up a Web Browser</i> on page E-21
5. Use TestManager to create a manual script.	<i>Planning and Creating a Manual Script</i> on page E-3
6. Run a manual script on the Web.	<i>Running a Manual Script on the Web</i> on page E-23
7. View the results in the LogViewer.	<i>Viewing the Results in the LogViewer</i> on page E-10

Software Requirements

Make sure that your server conforms to the following minimum requirements for a Web server:

Software Requirements

For the Web server:

- ▶ Windows 2000 Professional, Windows 2000 Server, Windows 2000 Advanced Server, Windows 98, Windows 95, Windows NT 4.0 Workstation, or Windows NT 4.0 Server.
- ▶ Microsoft Internet Explorer 5.0 or later.

For	Install
All versions of Windows 2000	Microsoft Internet Information Services 5.0 (IIS 5.0) from the Windows 2000 CD
Windows 98, Windows 95, or the Windows NT 4.0 Workstation	Microsoft Personal Web Server (PWS) from the Windows NT 4.0 Option Pack (available from Microsoft, www.microsoft.com)
Windows NT 4.0 Server	Microsoft Internet Information Server (IIS) from the Windows NT 4.0 Option Pack (available from Microsoft, www.microsoft.com)

NOTE: We recommend that you use Windows 2000 Server, Windows 2000 Advanced Server, or Windows NT 4.0 Server, to run manual scripts on the Web. You can not use a shared or networked repository with the Windows 2000 Professional or Windows NT 4.0 Workstation.

To access the Web server as a client, use one of the following Web browsers:

- ▶ Netscape Navigator 4.0 or later
- ▶ Microsoft Internet Explorer 4.0 or later

About Shared Repositories

We recommend that when you create a repository, you make it a shared repository so others can access your manual scripts over the Web. To share a repository, create the repository in a shared directory and use the Uniform Naming Convention (UNC) for the directory name. (For more information about creating a shared directory, see the *Using the Rational Administrator* manual or the Rational Administrator Help.)

Installing a Web Server

To install a Web server to run a manual script on the Web:

1. Install the **Rational TestM anager - Manual Script Execution** component.
 - a. Turn on the computer to start Windows.
 - b. Log in using an account with Administrator privileges.
 - c. Quit all applications before installing any Rational products.
 - d. Insert the CD into your CD drive.
The installation starts automatically.

If the CD does not start automatically, click **Start** → **Run**. Type *drive:\SETUP.EXE*, and then click **OK**.
 - e. At the **Rational Software Setup** page, click **Next**.
 - f. Click **Install Rational Product**, and then click **Next**.
 - g. Select the license you want, and then click **Next**. For information about software licenses, click **Help**, or see the *Administering Licenses for Rational Software* manual or the Rational License Administrator Help.
 - h. Click one of the following Rational testing products from the list of available products, and then click **Next**:
 - Rational LoadTest
 - Rational Robot
 - Rational Suite Enterprise
 - Rational Suite PerformanceStudio
 - Rational Suite TestStudio
 - Rational TeamTest

- i. Read the terms of the license agreement carefully, select one of the following, and then click **Next**.
 - **Yes, I accept the agreement.** If you accept the agreement, the Setup program continues.
 - **No, I don't accept the agreement.** If you do not accept the agreement, the Setup program ends.
- j. At the **Setup Configuration** page, click **Custom/Full**, and then click **Next**.
- k. Optionally, clear all check boxes on the **Choose Features** page.

Note: If you want to create manual scripts on the Web server, then you must install Rational Robot on the Web server.

- l. Select the **Web Server Components** check box.
 - m. Select the **Rational TestManager/Web** check box.
Installs the **Rational TestManager - Manual Script Execution** component.
 - n. Follow the on-screen instructions to complete the installation.
The Rational Setup wizard displays a series of dialog boxes that guide you through the installation process and prompt you for information.
 - o. Click **Finish** when done.
Your system restarts.
2. Install Microsoft Windows NT 4.0 Option Pack or Microsoft Internet Information Services 5.0 on the Web server by doing one of the following:
 - For a Web server running Windows 2000, install the Microsoft Internet Information Services 5.0 from the Windows 2000 CD.
 - For a Web server running Windows NT 4.0 Server, install the Microsoft Internet Information Server (IIS) from the Windows NT 4.0 Option Pack.
 - For a Web server running Windows 98, Windows 95, or Windows NT 4.0 Workstation, install the Microsoft Personal Web Server (PWS) from the Windows NT 4.0 Option Pack.
 3. Install Microsoft Internet Explorer 5.0 on the Web server.
 4. Configure the Microsoft Access Driver to run a manual script on the Web by doing the following
 - a. Click **Start** → **Settings** → **Control Panel**.

5. Right-click, and then click **New** → **Virtual Directory**.
6. Do one of the following:
 - For Windows 2000, click **Next**, and then go to the next step.
 - For Windows NT 4.0, go to the next step.
7. Type an alias for the TestManager Web site.

For example: TM

NOTE: Write down this alias. You must use this alias to run manual scripts on this Web server through a Web browser.

8. Click **Next**.
9. Type the drive and path of the location where you installed your Rational software, or click **Browse** to select the drive and path.

For example, the default location is:

```
c:\Program Files\Rational\Rational Test 7\www>manual script
```

10. Click **Next**.

11. Do one of the following:

For Windows 2000, select the following options to allow permissions to the Rational repository:

- **Read**
- **Run scripts (such as ASP)**

For Windows NT 4.0 Server, select the following options to allow permissions to the Rational repository:

- **Allow Read Access**
- **Allow Script Access**
- **Allow Execute Access**

12. Do one of the following:

- For Windows 2000, click **Next**, and then click **Finish**.
- For Windows NT, click **Finish**.

13. Right-click the new alias, and then click **Properties**.

14. Click the **Documents** tab.

Make sure that the following files appear under the **Enable Default Document** box:

- **Default.htm**
- **Default.asp**

If these file do not appear, do the following:

- a. Click **Add**.
- b. Type **Default.htm**, and then click **OK**.
- c. Type **Default.asp**, and then click **OK**.

15. Click the **Directory Security** tab.

16. Under **Anonymous and Authentication Control**, click **Edit**.

17. Make sure that you select the **Allow Anonymous Access** check box, and then click **Edit**.

18. Do one of the following:

- For Windows 2000, clear the **Allow IIS to control password** check box, and then go to the next step.
- For Windows NT 4.0, Clear the **Enable Automatic Password Synchronization** check box, and then go to the next step.

19. Type the **Username** and **Password** for the user account either on this Web server (if the repository is on the Web server), or on the domain (to access shared repositories on other systems in the domain).

This user account must have permission to read and write into the repository. All Web clients will use this account to access the repository through this Web server.

NOTE: By configuring the permissions to this account, you can restrict access to certain shared repositories. For more information about setting permissions, see your Microsoft Windows 2000 or Windows NT 4.0 documentation.

20. Click **OK** to close all windows.

Configuring a Microsoft Personal Web Server

Note: If you use Windows NT 4.0 Workstation or Windows 2000 Professional with the Personal Web Server, you can access only local repositories. To access shared repositories with Windows 2000 Professional, Windows 98, or Windows 95, you must run PWS under a domain user account.

To configure the Microsoft Personal Web Server (PWS) on a Windows 2000 Professional, Windows 98, Windows 95, or Windows NT 4.0 Workstation server:

1. Do one of the following:
 - For Windows 2000 Professional, be sure to install the Internet Information Services 5.0 (IIS 5.0) from the Windows 2000 Professional CD.
 - For Windows 98, Windows 95, or Windows NT 4.0 Workstation, be sure to install the Microsoft Personal Web Server (PWS) from the Windows NT 4.0 Option Pack (available from Microsoft, www.microsoft.com).
2. Do one of the following:
 - For Windows 2000 Professional, click **Start** → **Settings** → **Control Panel**. Double-click **Administrative Tools**. Double-click **Personal Web Manager**.
 - For Windows 98, Windows 95, or Windows NT 4.0 Workstation, click **Start** → **Programs** → **Windows NT 4.0 Option Pack** → **Microsoft Personal Web Server** → **Personal Web Manager**.
3. Click **Advanced**.
4. Select < **Home**> .
5. Click **Add**.
6. Under **Directory**, type the drive and path of the location where you installed your Rational software, or click **Browse** to select the drive and path.
For example, the default location is:
`c:\Program Files\Rational\Rational Test 7\www\manual script`
7. Under **Alias**, type the alias for the Web site.
For example: TM

NOTE: Write down this alias. You must use this alias to run manual scripts on this Web server through a Web browser.

8. Do one of the following:
 - For Windows 2000 Professional – Under **Access permissions**, select **Read** and **Script Source Access**. Under **Application permissions**, select **Execute (including scripts)**.
 - For Windows 98, Windows 95, or Windows NT 4.0 Workstation – Under **Access**, click all of the following: **Read**, **Execute**, and **Scripts**.
9. Click **OK**.
10. Do one of the following:
 - For Windows 98, Windows 95, or Windows NT 4.0 Workstation, click **Properties** → **Exit**.
 - For Windows 2000 Professional, click **File** → **Close**.
11. Complete the steps for either Windows 2000 Professional or for Windows 98, Windows 95, or Windows NT 4.0 Workstation
For Windows 2000 Professional:
 - a. Click **Start** → **Settings** → **Control Panel**.
 - b. Double-click **Users and Passwords**.
 - c. Under the **User Name** column, select:
`IUSR_machinename`
where *machinename* is the name of the Web server.
 - d. Click **Properties**.
 - e. Click the **Group Membership** tab.
 - f. Under **Other**, select **Administrators**.
 - g. Click **OK**. Click **OK** again.
 - h. Restart the system.For Windows 98, Windows 95, or Windows NT 4.0 Workstation:
 - a. Click **Start** → **Programs** → **Administrative Tools (Common)** → **User Manager**.
 - b. Under the **Username** column, select:
`IUSR_machinename`
where *machinename* is the name of the Web server.
 - c. Click **User** → **Properties**.

- d. Click **Groups**.
- e. Under **Not members of**, select **Administrators**, and then click **Add**.
- f. Click **OK**. Click **OK** again.
- g. Click **User** → **Exit**.

Setting Up a Web Browser

You can use Netscape Navigator 4.0 (or later) or Microsoft Internet Explorer 4.0 (or later) as your Web browser for running manual scripts on the Web. You can use your Web browser on a system running Microsoft Windows, UNIX, or Apple Macintosh operating system software.

Netscape Navigator

To set up a Netscape Navigator browser for running manual scripts on the Web:

1. Start Netscape Navigator.
2. Click **Edit** → **Preferences**. Under Category, click **Advanced**.
3. Double-click **Advanced**, and click **Cache** to display the Cache panel.
4. In the Cache panel, click **Every time**.
5. Click **OK**.

Microsoft Internet Explorer

To set up a Microsoft Internet Explorer browser for running manual scripts on the Web:

1. Start Internet Explorer.
2. For Internet Explorer 5.0, click **Tools** → **Internet Options**.
For Internet Explorer 4.0, click **View** → **Internet Options**.
3. Click the **General** tab.
4. Under **Temporary Internet files**, click **Settings**.
5. Under **Check for newer versions of stored pages**, click **Every visit to the page**.
6. Click **OK**. Click **OK** again.

Troubleshooting for Manual Scripting on the Web

This section lists some problems you may experience when running manual scripts on the Web, a description of each problem, and the solution to take to correct each problem.

Error message – None.

Problem – You cannot connect from a Web browser to a Web server running the Microsoft Personal Web Server (PWS).

Solution – If you restart a Web server running PWS, PWS may not start automatically when the server restarts. This is an intermittent problem. To fix the problem, restart PWS.

To restart PWS:

1. Click **Start** → **Programs** → **Windows NT 4.0 Option Pack** → **Microsoft Personal Web Server** → **Personal Web Manager**.
2. Under **Publishing**, click **Start**.
3. Click **Properties** → **Exit**.

Error message – Unable to connect to repository.

Problem – When you log into a repository from a Web browser, you get this error message.

Solution – Make sure the Web server security permissions are set correctly. For information about setting security permissions, see *Configuring a Microsoft Internet Information Server* on page E-16 or *Configuring a Microsoft Personal Web Server* on page E-19.

Error message – Error message that includes `Server.ObjectCreate` in the message.

Problem – You get an error message when you select a manual script.

Solution – Make sure that you or the Web server administrator installs Microsoft Internet Explorer 5.0 on the Web server.

Error message – None.

Problem – When you type text in a dialog box and submit it, you get erratic behavior. Alternatively, when you open a script, results and comments are already filled in from the last session.

Solution – Disable caching on your Web browser. For information about disabling caching, see *Setting Up a Web Browser* on page E-21.

Error message – None.

Problem – After you connect to the Web server, a Login dialog box appears. In the Login dialog box, the repository select list is empty.

Solution – Create a repository and create manual scripts, or register an existing repository that contains manual scripts.

To create a repository or register an existing repository:

1. Do one of the following:
 - For IIS, log into the user account of the virtual directory that you configured for manual scripting on the Web. For information, see *Configuring a Microsoft Internet Information Server* on page E-16.
 - For PWS, log into the user account that the Web server runs under. For information, see *Configuring a Microsoft Personal Web Server* on page E-19.
2. Start the Rational Administrator and create a new repository, or register an existing repository. For information about creating or registering a repository, see the *Using the Rational Administrator* manual or the Rational Administrator online Help.
3. If you create or register a shared repository, make sure that the permissions for the repository directory are set for the virtual directory user account for IIS, or for the user account that the Web server runs under for PWS.
4. Restart the Web server.

Troubleshooting Your Web Server

If you have problems with your Web server, check to make sure that your Web server meets the software requirements. For information, see *Software Requirements* on page E-13.

Running a Manual Script on the Web

The manual script feature of TestManager lets you remotely run a manual script over the Internet. You use a Web browser over the Internet to access a Rational repository. You can remotely indicate results and add comments as you perform each task in a manual script.

You can run any manual script on the Web if it was created using the grid or text editor. If the manual script includes an external file, you can run it only in TestManager.

To run a manual script on the Web:

1. Start a Web browser, either Netscape Navigator 4.0 (or later) or Microsoft Internet Explorer 4.0 (or later).

2. Connect to the Web server by typing the following:

`http://machinename/alias`

where *machinename* is the network name of the Web server, and *alias* is the name of an alias that you or your administrator set up on the Web server.

For example:

`http://dell1300/TM`

For information about setting up an alias for a Web server running all versions of Windows 2000, Windows 98, Windows 95, or Window NT 4.0 Workstation, see *Configuring a Microsoft Personal Web Server* on page E-19. For a Web server running Windows NT 4.0 Server, see *Configuring a Microsoft Internet Information Server* on page E-16.

3. Log into Manual Script Execution on the Web.
 - a. Type the user ID and password of the repository that contains the manual scripts that you want to run. If you do not know the ID and password, see your repository administrator.
 - b. Select or type the repository path or the Uniform Naming Convention (UNC) for a shared repository. You need permission to access a shared repository. (For information about creating a shared repository, see the *Using the Rational Administrator* manual or the Rational Administrator Help.)

For example:

`c:\defects\repo`

or

`\\dell1300\defects\repo`

- c. If you select a repository from the list of repositories, a list of projects appears. If you type in a repository path or UNC, click **OK** or press the TAB key to display a list of projects in the **Project** box.
 - d. Select a project.
 - e. Click **OK**.
4. Select a manual script from the list, and click **OK**.
 5. To change the options for running a manual script on the Web, click **Options**, select the options, and click **OK**. The options are:

Show script without graphics – Improves the display speed of the manual script Web page. Displays only text on the Web page.

Show only verification points – Shows verification points and hides all steps.

Log unchecked steps as warnings – Unchecked steps appear as warnings in the log.

Timeout interval in minutes (20-1440) – Sets the length of time, in minutes, before the Web browser session times out. You can set the time from 20 to 1440 minutes. The default time is set to 120 minutes.

NOTE: You must submit your results before the Web browser session times out or you will lose all results and comments that you enter. (See step 9.)

6. To view a note for a row, click the Note icon. You cannot edit notes.
7. Perform the steps and verification points indicated in the manual script.
 - For a step, select the check box in the Result column to indicate that you have performed the step. When you view the log, **Complete** appears in the Result column of each step that you checked.
 - For a verification point, click the cell in the Result column (where it says **None**) and click **None**, **Pass**, or **Fail**. When you view the log, the result appears in the Result column.

To pass all verification points and steps, click **Pass Results**.

To clear all results, click **Reset Results**.

8. To type a comment, click the Comment cell. Type the text and click **OK**. The Comment icon appears in the cell. To view or edit the comment, click the Comment icon.

When you view the log, the comment appears in the **Result** tab of the log event.

9. When finished, click **Submit Results**.
10. Fill in the Specify Log Information dialog box and click **OK**.
11. Click **Log Off** or **Select Script** to run another manual script.

To view the results, see *Viewing the Results in the LogViewer* on page E-10.

Working with External Scripts

Using TestManager, you can create an external script that lets you run an existing executable program or a test program created with any tool. External testing lets you:

- ▶ Run an external script using TestManager.
- ▶ Integrate an external script with TestManager to record and view test results using the LogViewer.
- ▶ Include your external script in TestManager reports.

Planning an External Script

When you plan an external script, you name it, set its properties, and then type the name of the external test that you want to run. You can also attach the script to a test requirement.

To plan an external script in TestManager:

1. Do one of the following to plan the script:
 - To plan a script and attach it to a requirement, open the Requirements Hierarchy. Right-click the requirement, and click **Plan** → **External Script**.
 - To plan a script without attaching it to a requirement, make sure that no requirement is selected. Click **File** → **Plan** → **External Script**.
2. Click the **General** tab and fill in the Plan Script dialog box.
3. Be sure to type the drive and path of the existing external test that you want to run in the **Command** box, or click **Browse** to select the path. This command runs the external test.
4. Select the **Developed** check box.

This setting is used by TestManager when you run a Development Coverage report. For information about coverage reports, see *Coverage Reports* on page 16-2.

5. Click **OK**.

Logging Results of an External Script

To log the results of running an external script, you use the **Rtresult.exe** file provided with your Rational software. You need to call the **Rtresult.exe** file from the source code of your external test.

In your source code, use the following command:

```
RTRESULT . EXE /parameters
```

The following table lists the parameters that you can use to log the results of an external script:

Syntax Element	Description
/result	Specifies whether the script passed or failed. Valid values: <ul style="list-style-type: none"> pass – The default. The expected result is that the script passed. fail – The expected result is that the script failed.
/note	Specifies any additional text describing the results of an external script. You must enclose the text within quotation marks if there is a space within the text. You can enter up to 255 alphanumeric characters. To view these notes: <ol style="list-style-type: none"> 1. In the LogViewer, right-click the event. 2. Click Properties. 3. Click the Result tab. The note appears in the Description box.
/ID	Specifies that you are running more than one external script at the same time. For information, see the next section, <i>Logging Results of Several External Scripts</i> .
/customparam	You can create your own custom parameters for an external script. You must enclose the text within quotation marks if there is a space within the text. You can enter up to 255 alphanumeric characters. To view a custom parameter: <ol style="list-style-type: none"> 1. In the LogViewer, right-click the event. 2. Click Properties. 3. Click the Configuration tab.

Comments

Values that contain spaces must be enclosed within quotation marks.

Example

```
rtresult.exe /result fail /note "This manual script should fail to meet requirements." /DB2 "Indicates the database the manual script runs against."
```

Logging Results of Several External Scripts

If you run more than one external test at the same time, you must supply an ID for each external test to **Rtresult.exe** in order to store the test results in the appropriate log.

When TestManager starts your external script, TestManager passes this ID to your external test.

To log test results when running more than one external test at the same time:

1. In the source code of each external test, you must store the ID value passed from TestManager and use it when calling **Rtresult.exe**.
2. Add the **Rtresult.exe** command and the ID parameter to the source code of each external test. For information about each parameter, see *Logging Results of an External Script* on page E-26.

For example: `rtresult.exe /ID N /result pass`

where *N* equals the ID passed to your test from TestManager.

3. In TestManager, add %ID to the command line that starts an external test:
 - a. Start TestManager.
 - b. Click **File** → **Plan** → **External Script**.
 - c. Click the **General** tab.
 - d. Be sure to type the drive and path of the existing external test that you want to run in the **Command** box, or click **Browse** to select the path. Then type %ID after the drive and path.

For example: `c:\mytests\extest.exe %ID`

TestManager replaces %ID with a real ID when the external test executes.
 - e. Fill in the rest of the Plan Script dialog box.
 - f. Click **OK**.
4. Run the scripts. For information, see the next section, *Running an External Script*.

Running an External Script

To run an external script:

1. Do one of the following:
 - Right-click the script in either the Requirements Hierarchy or the Script Query window and click **Run**.
 - Click **File** → **Run** → **Script**. Click the script to run, and then click **OK**.
2. Fill in the Specify Log Information dialog box and click **OK**.

The external script runs the external test and stores the results in the LogViewer, if you set up the external script to log the results. For information, see *Logging Results of an External Script* on page E-26 and *Logging Results of Several External Scripts* on page E-28. To view the results of an external script, see the next section, *Viewing the Results of Running an External Script*.

Viewing the Results of Running an External Script

To view the results of running an external script:

1. Open the Asset Browser (click **View** → **Asset Browser**).
2. Expand the Builds tree until the log name appears.
3. Double-click the log to open it in the LogViewer.

The **Result** column shows whether the external script passed or failed.

4. To see more details about an event, right-click the event and click **Properties**. Click the **Result** tab.

The **Description** box shows the text of the note when you call **Rtresult.exe** with the `/note` parameter. The **Additional information** box is not used for external scripts.

Click the **Configuration** tab.

Text appears in this tab when you call **Rtresult.exe** with the `/custom` parameter. The **Setting** column displays the name of the custom parameter. The **Value** column displays the text of the custom parameter.

For information about using the `/note` and `/custom` parameter, see *Logging Results of an External Script* on page E-26.

Glossary

action object – In TestFactory, an object in the application map that represents an action to which a control in the application responds. Typical actions are mouse left-click, mouse right-click, and mouse left-double-click; the corresponding action objects in the application map are LeftClick, RightClick, and LeftDoubleClick.

ActiveX control – A reusable software control that takes advantage of Object Linking and Embedding (OLE) and Component Object Modeling (COM) technologies. Developers can use ActiveX controls to add specialized functions to applications, software development tools, and Web pages. Robot can test ActiveX controls in applications.

actual results – In a functional test, the outcome of testing an object through a verification point in a GUI script. Actual results that vary from the recorded baseline results are defects or intentional changes in the application. See also *baseline results*.

Administrator – See *Rational Administrator*.

Agent computer – In LoadTest, a computer that has the Rational Agent software installed and that plays back a virtual user or GUI script. In a LoadTest schedule, you can identify the Agent computer on which to run a script. See also *Rational Agent*.

API recording – In Robot, a virtual user recording method that captures API calls between a specific client application and a server. These calls are captured on the client computer.

application map – In TestFactory, a hierarchical list of controls and actions in the application-under-test, as well as the states of the application-under-test and the transitions between those states. An application map can include UI objects and action objects, as well as TestFactory objects such as Pilots, Test Suites, and scripts.

application-under-test – The software being tested. See also *system-under-test*.

Asset Browser – A window that displays testing resources such as builds, queries, scripts, schedules, reports, report output, and logs. The Asset Browser is available in TestManager and LoadTest.

AUT – See *application-under-test*.

automated testing – A testing technique in which you use software tools to replace repetitive and error-prone manual work. Automated testing saves time and enables a reliable, predictable, and accurate testing process.

AutoPilot – In TestFactory, a tool for running scripts, Test Suites, and Pilots. The scripts and Test Suites can run on your local computer or on computers in the Test Lab. The Pilots run on your local computer, and the scripts they generate can run on your local computer or on computers in the Test Lab.

base state – In TestFactory, the known, stable state in which you expect the application-under-test to be at the start of each script segment. See also *script segment*.

baseline results – In a functional test, the outcome of testing an object through a verification point in a GUI script. The baseline results become the expected state of the object during playback of the script. Actual test results that vary from the baseline results are defects or intentional changes in the application. See also *actual results*.

best script – In TestFactory, an optimized script generated by a Pilot. A best script contains the fewest number of script segments that provide the most coverage of the source code or user interface in the application-under-test.

breakpoint – A feature of the Robot debugger. When you assign a breakpoint to a line of code, and then run the script in the debugger environment, the script stops executing at that line of code. Control returns to you, and the breakpoint line is displayed. From here you can view variables, perform other debugging activities, and continue executing the script.

build – A version of the application-under-test. Typically, developers add new features or enhancements to each incremental build. As team members test a build, they enter defects against those features that do not behave as expected. You use TestManager to define and manage builds.

built-in data test – A data test that comes with Robot and is used with the Object Data verification point. A data test uses a specific property of the object, in conjunction with other parameters, to determine the data to capture. Although built-in data tests cannot be edited, renamed, or deleted, they can be copied and then edited, and they can be viewed. See also *custom data test*.

ClearQuest – See *Rational ClearQuest*.

client/server – An architecture for cooperative processing in which the software tasks are split between server tasks and client tasks. The client computer sends requests to the server, and the server responds.

code coverage – In TestFactory, the percentage of code that is tested by a script. This percentage is based on the portion of the code that a script touches, relative to all code in the application-under-test. A Pilot can use code coverage to determine the best script for a run. See also *UI coverage*.

command ID – In LoadTest’s VU language, an identifier for a command. Robot automatically assigns a unique command ID, composed of an alphanumeric prefix and a three-digit number, to each emulation command. Because command IDs appear in both the virtual user script and the LoadTest report output, they enable you to determine the relationship between an emulation command and its response times.

command ID prefix – In LoadTest, a prefix for a unique emulation command ID. The prefix defaults to the script name (up to the first seven characters). However, you can define the prefix in the Generator tab of the Virtual User Record Options dialog box.

custom data test – A customer-defined data test used with the Object Data verification point. A data test uses a specific property of the object, in conjunction with other parameters, to determine the data to capture. Custom data tests are created within your organization and are stored in the repositories that were active when they were created. They can be edited, renamed, and deleted. See also *built-in data test*.

data test – A test that captures the data of an object with the Object Data verification point. See also *built-in data test* and *custom data test*.

datapool – A source of test data that GUI scripts and virtual user scripts can draw from during playback. You can automatically generate datapools using TestManager, or you can import datapool data from other sources such as your database.

dependency – In LoadTest, a method of coordinating an object in a schedule with an event. For example, if the script Query is dependent upon the script Connect, then Connect must finish executing before Query can begin executing. See also *event*.

distributed architecture – Architecture in which computer systems work together and communicate with each other across LAN, WAN, or other types of networks. A client/server system is an example of distributed architecture.

distributed functional test – In LoadTest, a test that uses multiple Agent computers to execute multiple GUI scripts written in the SQABasic language.

dynamic load balancing selector – A type of selector in a LoadTest schedule. Items in the selector, such as scripts, are executed according to a weight that you set.

emulation commands – VU language statements or commands that emulate client activity, evaluate the server’s responses, and perform communication and timing operations. LoadTest stores the results of emulation commands in a log file, which you can view from the LogViewer.

emulation functions – VU language functions that emulate client activity and evaluate the server's responses. Unlike emulation commands, emulation functions do not perform communication and timing operations, and they are not logged.

environment control commands – VU language commands that let you control a virtual user's environment by changing the VU environment variables. For example, you can set the level of detail that is logged or the number of times that virtual users attempt to connect to a server.

event – An item in a LoadTest schedule upon which another item is dependent. For example, if the script Connect sets an event and the script Query depends on this event, Connect must finish executing before Query can begin executing. See also *dependency*.

external script – A script that runs a program created with any tool. You plan and run external scripts in TestManager.

fixed user group – In LoadTest, a group that contains a scalable number of users. When you create a fixed user group, you indicate the maximum number of users that you will run in the group. Typically, you use fixed user groups in functional tests, which do not add a workload to the system.

flow control statements – In the VU and SQABasic languages, statements that let you add conditional execution structures and looping structures to a script.

functional test – A test to determine whether a system functions as intended. Functional tests are performed on GUI objects and objects such as hidden DataWindows and Visual Basic hidden controls.

Grid Comparator – The Robot component for reviewing, analyzing, and editing data files for text and numeric verification points in grid formats. The Grid Comparator displays the differences between the recorded baseline data and the actual data captured during playback.

GUI script – A type of script written in the SQABasic language. It contains GUI actions such as keystrokes and mouse clicks. Typically, a GUI script also contains verification points for testing objects over successive builds of the application-under-test.

GUI user – The type of user that is emulated when a GUI script is executed. Only one GUI user at a time can run on a computer.

hidden object – An object that is not visible through the user interface. Hidden objects include objects with a visible property of False and objects with no GUI component.

IDE – Integrated Development Environment. This environment consists of a set of integrated tools that are used to develop a software application. Examples of IDEs supported by Robot include Oracle Forms, PowerBuilder, Visual Basic, and Java.

Image Comparator – The Robot component for reviewing and analyzing bitmap image files for Region Image and Window Image verification points. The Image Comparator displays differences between the recorded baseline image and the actual image captured during playback. The Image Comparator also displays unexpected active windows that appear during playback.

instrumentation – In TestFactory, the process of inserting code coverage counters into the application-under-test. These counters record how much code is executed during a script run. See also *object code instrumentation* and *source code instrumentation*.

load – See *workload*.

load balancing – See *workload balancing*.

LoadTest – See *Rational LoadTest*.

log – A repository object that contains the record of events that occur while playing back a script or running a schedule. A log includes the results of all verification points executed as well as performance data that can be used to analyze the system's performance.

LogViewer – See *Rational LogViewer*.

low-level recording – A recording mode that uses detailed mouse movements and keyboard actions to track screen coordinates and exact timing. During playback, all actions occur in real time, exactly as recorded.

manual script – A set of testing instructions to be run by a human tester. The script can consist of steps and verification points. You create manual scripts in TestManager.

Master computer – A computer that executes LoadTest. From this computer, you create, run, and monitor schedules. When the run is finished, you use it to analyze test results.

mix-ins – See *Pilot mix-ins*.

network recording – In Robot, a virtual user recording method that records packet-level traffic. This traffic is captured on the wire.

next available selector – In LoadTest schedules, a selector that distributes each item such as a script, delay, or other selector to an available computer or virtual user. This type of selector is used in a GUI schedule. The next available selector parcels out the items sequentially, based on which computers or virtual users are available.

object – An item on a screen, such as a window, dialog box, check box, label, or command button. An object has information (properties) associated with it and actions that can be performed on it. For example, information associated with the window object includes its type and size, and actions include clicking and scrolling. In some development environments, a term other than *object* is used. For example, the Java environment uses *component*, and the HTML environment uses *element*.

object code instrumentation – In TestFactory, the process of inserting code coverage counters into the executable file of the application-under-test. These counters record how much of the program a script tests. See also *instrumentation* and *source code instrumentation*.

Object-Oriented Recording® – A script recording mode that examines objects in the application-under-test at the Windows layer. Robot uses internal object names to identify objects, instead of using mouse movements or absolute screen coordinates.

Object Properties Comparator – The Robot component that you use to review, analyze, and edit the properties of objects captured by an Object Properties verification point. The Object Properties Comparator displays differences between recorded baseline data and the actual data captured during playback.

Object Scripting commands – A set of SQABasic commands for accessing an application's objects and object properties. You add Object Scripting commands manually when editing a script.

Object Testing® – A technology used by Robot to test any object in the application-under-test, including the object's properties and data. Object Testing lets you test standard Windows objects and IDE-specific objects, whether they are visible in the interface or hidden.

OCI – Object Code Insertion. The Rational technology used in TestFactory to instrument object code and measure how much of the application-under-test a script tests. See also *code coverage* and *object code instrumentation*.

performance test – A test that determines whether a multi-client system performs within user-defined standards under varying loads. Performance tests are always run from a schedule in LoadTest.

Pilot – In TestFactory, a tool for generating scripts automatically.

Pilot mix-ins – In TestFactory, a list of Pilots that are executed on a random basis during the run of a lead Pilot. Mix-ins are useful for randomly testing multiple areas of the application-under-test. To make tests more realistic, you can combine mix-ins and scenarios.

Pilot scenario – An ordered list of Pilots that are executed during the run of a Pilot. A Pilot scenario is useful for testing UI objects that need to be exercised in a specific order. To make tests more realistic, you can combine scenarios and mix-ins.

project – A collection of data, including test assets, defects, requirements, and models, that can facilitate the development and testing of one or more software components.

proxy recording – In Robot, a virtual user recording method that captures the client/server conversation on the network wire rather than on the client computer. Proxy recording allows Robot to capture network packets that are not visible to it during network recording — for example, if the client and server are in different network segments.

query – A request for information stored in the repository. A query consists of a filter and several visible attributes — the columns of data to display, the width of the column, and the sort order.

random selector – A type of selector in a LoadTest schedule. Items in the selector, such as scripts, are randomly executed. Random selectors can be with replacement, where the odds are the same, or without replacement, where the odds change with each iteration.

Rational Administrator – The component for creating and maintaining repositories, projects, users, groups, computers, and SQL Anywhere servers.

Rational Agent – The LoadTest software that resides on a shared network drive and runs on each computer where testing occurs. The entries specified in a schedule play back on the Agent computer, which reports on their progress and status as they run. See also *Agent computer*.

Rational ClearQuest – The Rational product for tracking and managing defects and change requests throughout the development process. With ClearQuest, you can manage every type of change activity associated with software development, including enhancement requests, defect reports, and documentation modifications.

Rational LoadTest – The Rational Test component for running performance, stress, scalability, multi-user, and distributed functional tests on multiple Agents connected by a network. With LoadTest, you can initiate test runs and monitor tests from a master computer that manages the test process. LoadTest is available only in Rational Suite PerformanceStudio.

Rational LogViewer – The Robot component for displaying logs, which contain the record of events that occur while playing back a script or running a schedule. Also, the component from which you start the four Comparators.

Rational PerformanceArchitect – The Rational component that lets you test the performance of COM/DCOM applications. With Rational PerformanceArchitect, you can create a Rose sequence or collaboration diagram, convert it to a virtual user script, and then use Rational Suite PerformanceStudio to edit the script and run the performance tests.

Rational repository – A database that stores application testing information, such as test requirements, scripts, and logs. All Rational Suite TestStudio and Rational Suite PerformanceStudio products and components on your computer update and retrieve data from the same connected repository. A repository can contain either a Microsoft Access or a Sybase SQL Anywhere database.

Rational RequisitePro – The Rational product for organizing, managing, and tracking the changing requirements of your system.

Rational Robot – The Rational product for recording, playing back, debugging, and editing scripts.

Rational SiteCheck – The Robot component for managing your intranet or World Wide Web site. You can use SiteCheck to visualize the structure of your Web site, and you can use it with Robot to automate Web site testing.

Rational Synchronizer – The Rational tool that ensures the consistency of data across several Rational products.

Rational TestAccelerator – An agent application that executes scripts. TestFactory uses computers running TestAccelerator as remote machines on which to run automated distributed tests.

Rational TestFactory – The Rational Test component for mapping an application-under-test and generating scripts automatically. TestFactory is available in Rational Suite TestStudio and Rational Suite PerformanceStudio.

Rational TestManager – The Robot component for managing the overall testing effort. You use it to define and store information about test documents, requirements, scripts, schedules, and sessions.

Report Layout Editor – The TestManager component for customizing the layout of reports.

repository – See *Rational repository*.

RequisitePro – See *Rational RequisitePro*.

Robot – See *Rational Robot*.

scalable user group – In LoadTest, a group that contains a varying number of users. When you create a scalable user group, you assign it a percentage of the total workload. Assume you have a scalable user group that is 50 percent of the workload. If you run a test with 10 users, the group will contain 5 users. If you run a test with 100 users, the group will contain 50 users.

scenario – In LoadTest, a modular group of scripts and other items in a schedule that is used by more than one user group. A scenario can contain scripts, delays, and synchronization points.

scenario – See *Pilot scenario*.

schedule – In LoadTest, structure that you create to specify how scripts should be played back. A schedule can contain GUI scripts and virtual user scripts, and can indicate the number of times to repeat a script and the computer on which the script will run. In performance testing, a schedule is used to create a workload. In distributed functional testing, a schedule is used to distribute scripts among various computers.

script – A set of instructions used to navigate through and test an application. You can generate scripts in a variety of ways. You can use Robot to record scripts used in functional testing and performance testing. You can also use TestManager to create and manage manual scripts, and to manage external scripts created with a third-party testing tool. A script can have properties associated with it, such as the purpose of the script and requirements for the script. See also *external script*, *GUI script*, *manual script*, and *virtual user script*.

script outline – In TestFactory, the readable version of a script. A script outline contains a description of the actions that Robot performs while running the script.

script segment – In TestFactory, a section of a script that tests a particular element of product functionality. A Pilot generates a script segment by starting the application-under-test in a base state, navigating through the part of the product that you are testing, and returning the application-under-test to the base state. See also *base state*.

seed – An initial number fed to a random number generator. Using the same seed produces the same series of random numbers. In LoadTest, you use seeds to generate think times.

selector – An item that you insert in a LoadTest schedule to indicate how often and in what order to run scripts.

sequential selector – In a LoadTest schedule, a type of selector that executes each script, delay, or other item in the same order in which it appears in the schedule.

session – In virtual user recording, one or more scripts that you record from the time you begin recording until the time you stop recording. Typically, the scripts in a session represent a logical flow of tasks for a particular user, with each script representing one task. For example, a session could be made up of three scripts: *login*, *testing*, and *logout*. In TestFactory, a session is the period of time that the TestFactory application or a window is open.

shared variable – An integer variable that multiple scripts and multiple virtual users can read and write to. You can see the value of a shared variable while monitoring a LoadTest schedule. For example, you can set a shared variable as a flag to end a playback session. Each script can check the flag to see if the session should end. When that flag is set, exit tasks can be performed.

shell script – A script that calls or groups several other GUI scripts and plays them back in sequence. Shell scripts provide the ability to create comprehensive tests and then store the results in a single log.

SiteCheck – See *Rational SiteCheck*.

source code instrumentation – In TestFactory, the process of inserting code into the source code of the application-under-test. This code measures how much of the source code a script tests. See also *instrumentation* and *object code instrumentation*.

SQABasic – The Robot scripting language for recording GUI actions and verifying GUI objects. SQABasic contains most of the syntax rules and core commands that are contained in the Microsoft Basic language. In addition, SQABasic has commands that are specifically designed for automated testing. See also *VU*.

stable load – In LoadTest, a condition that occurs when a specified number of virtual users have logged on to the system-under-test and are active. When the stable load criterion is met, LoadTest begins measuring the load.

streak – When running a virtual user schedule in LoadTest, a series of successes or failures for emulation commands. You can see a streak while monitoring a schedule.

structural test – A test to determine whether the structure of a Web site is consistent and complete. A structural test ensures that an application's interdependent objects are properly linked together. You perform a structural test using SiteCheck.

synchronization point – In LoadTest, a place where emulated virtual users stop and wait until all other synchronized users reach that point. When all users reach the synchronization point, they are released and continue executing.

Synchronizer – See *Rational Synchronizer*.

system tuning – In LoadTest, the process of optimizing a system's performance by changing hardware resources and software configuration parameters while using a constant workload.

system-under-test – The system being tested. This includes the computers and any software that can generate a load on the system, networks, user interfaces, CPU s, and memory. See also *application-under-test*.

test assets – The resources that facilitate the planning or development phases of the testing effort. Examples of test assets include scripts, schedules, sessions, test documents, and test requirements.

test development – The process of developing tests to verify the operation of a software application. This includes creating scripts that verify that the application-under-test functions properly. Test development lets you establish the baseline of expected behavior for the application-under-test.

test documents – Test plans, project schedules, resource requirements, and any other documents that are important to your project. You develop your test documents using your own word processing or scheduling program; you then reference the name and location of the document in TestManager. This lets members of the test and development team locate documents quickly.

Test Lab – A collection of computers on which TestAccelerator is running. In TestFactory, you can distribute the scripts associated with a Pilot, a Test Suite, or the AutoPilot to run on computers in the Test Lab. See also *Rational TestAccelerator*.

Test Suite – In TestFactory, a tool for running a collection of scripts as a group.

TestAccelerator – See *Rational TestAccelerator*.

TestFactory – See *Rational TestFactory*.

TestManager – See *Rational TestManager*.

Text Comparator – The Robot component for reviewing, analyzing, and editing data files for text and numeric verification points in any format except grids. The Text Comparator displays the differences between the recorded baseline results and the actual results.

think time – In virtual user and GUI scripts, think times are delays that simulate a user's pauses to type or think while using an application. With virtual user scripts, LoadTest calculates the think time at runtime, based on think time VU environment variables that are set in the script. You can set a maximum think time in Robot. With GUI scripts, Robot uses the actual delays captured between keystrokes, menu choices, and other actions.

transaction – In LoadTest, a logical unit of work performed against a server. For example, submitting a search query or submitting a completed form to a Web server are both transactions.

transaction rate – In LoadTest, the playback speed calculated as a function of number of transactions per unit of time. For example, if a script contains one transaction, and each script is started at half-second intervals, your transaction rate would be 2 per second.

transactor – In LoadTest, an item that you insert in a LoadTest schedule to indicate the number of user-defined transactions that a virtual user performs in a given time period.

UI coverage – In TestFactory, the percentage of objects in the application map that are tested by a Pilot-generated script. This percentage is the proportion of UI objects that the script touches, relative to all UI objects available to the Pilot. A Pilot can use UI coverage to determine the best script for a run. See also *code coverage*.

UI object properties – Attributes of object classes and UI objects that TestFactory uses to map applications and generate scripts.

unexpected active window – A window that appears during script playback that interrupts the script playback process and prevents the expected window from being active. For example, an error message generated by the application-under-test is an unexpected active window. You can view unexpected active windows in the Image Comparator.

user group – In LoadTest, a collection of users that execute similar tasks and generate the same basic workload. Accountants and data entry operators are examples of user groups.

verification – The process of comparing the test results from the current build of the software to its baseline results.

verification point – A point in an SQABasic script that confirms the state of one or more objects. During recording, a verification point captures object information from the application-under-test and stores it as the baseline. During playback, a verification point recaptures the object information and compares it to the baseline. In a manual script, a verification point is a question about the state of the application-under-test.

virtual user – In LoadTest, a type of user that is emulated when a virtual user script is executed. A computer can run multiple virtual users simultaneously.

virtual user script – A type of script written in the VU language. Virtual user scripts contain client/server requests and responses as well as user think times.

VU – The Robot scripting language for recording a client's requests to a server. VU provides most of the syntax rules and core commands available in the C programming language. In addition, VU has emulation commands and functions that are specifically designed for automated performance testing. See also *SQABasic*.

wait state – A delay or timing condition that handles time-dependent activities.

workload – In LoadTest, the set of all activities that users perform in an actual production setting of the system-under-test. You can use LoadTest to emulate a workload.

workload balancing – In LoadTest, the act of distributing activities so no one system or device becomes a bottleneck.

workload model – In LoadTest, the workload model is represented as a schedule. You can play back this schedule and analyze the response times.

▶ ▶ ▶ Index

A

- access order of datapool rows 8-3
- acknowledging results for GUI script playback 9-5
- adding
 - build states 3-9
 - custom fields to scripts 2-16
 - features to GUI scripts 5-1
 - masks during recording 14-17
 - properties in Object Properties Comparator 11-7
 - query filter statements 15-6, 15-7
- addresses data type C-2
- Administrator 1-2
- alphanumeric values, testing 6-3
- Alphanumeric verification point 6-3, 12-1
- animation mode for debugging 7-13
- applets, Java 20-3
- applications
 - Java 20-3
 - starting 5-1
- Apply a User-Defined DLL test function verification method 6-15
- ASCII text files 8-42
- Asset Browser, displaying builds in 3-4
- associating
 - schedules with test requirements 2-19
 - scripts with test requirements 2-13
 - variable names and datapool columns 8-42
- attaching
 - files to defects 10-19
 - LoadTest schedules to test requirements 2-19
 - scripts to test requirements 2-13

- Auto Mask feature in Image Comparator 14-12
- automatically generating values for user-defined data types 8-33
- automatically masking differences in Image Comparator 14-12
- autonaming GUI scripts 4-7

B

- baseline file, editing
 - in Grid Comparator 13-8
 - in Object Properties Comparator 11-9
 - in Text Comparator 12-5
- baseline file, replacing
 - in Grid Comparator 13-11
 - in Image Comparator 14-17
 - in Object Properties Comparator 11-12
 - in Text Comparator 12-7
- baseline file, saving
 - in Grid Comparator 13-11
 - in Image Comparator 14-18
 - in Object Properties Comparator 11-12
 - in Text Comparator 12-7
- batch compiling scripts and library source files 7-8
- breakpoints, setting and clearing 7-11
- Browser NewPage command 19-15
- browsers
 - playing back scripts in Internet Explorer 19-2
 - playing back scripts in Netscape Navigator 19-12
 - recording scripts in Internet Explorer 19-2
 - requirements for running manual scripts on the Web E-13

Index

- setting up Microsoft Internet Explorer E-21
- setting up Netscape Navigator E-21
- Build Listing reports 16-4
- build states 3-4, 3-9, 3-10
- Build tab of Output window 7-9
- builds
 - copying 3-7
 - creating 3-5
 - deleting 3-7
 - intentional changes to 10-10
 - organizing in the Asset Browser 3-4
 - renaming 3-7
- built-in data tests B-1
- By Content identification method 6-16
- By Key/Value identification method 6-17
- By Location identification method 6-16
- By Title identification method 6-16

C

- C++ applications
 - enabling for testing 4-5
 - recognition order preference 4-10
- C++ Recognition Order preference 4-10
- calling scripts from within scripts 5-3
- Case-Insensitive verification method 6-14
- Case-Sensitive verification method 6-14
- changing
 - color of masks and differences in Image Comparator 14-7
 - column order in LogViewer 10-8
 - how differences are determined in Image Comparator 14-7
 - identification methods in Object Properties Comparator 11-12
 - object class mappings 4-15
 - projects 1-16
 - script properties in Robot 4-24
 - verification method in Object Properties Comparator 11-11
- child requirements, inserting 2-7
- choosing query fields to display 15-5
- cities data type C-2
- clearing breakpoints 7-11
- ClearQuest 10-14
- Clipboard verification point 6-3
- clipboard, testing content of 6-3
- collapsing/expanding log events in LogViewer 10-7
- columns in data grids
 - changing widths 6-22
 - testing titles 6-20
 - transposing with rows 6-23
- columns in datapools
 - assigning data types to 8-20
 - assigning values from a text file 8-42, C-8
 - deleting 8-26
 - editing column definitions, in TestManager 8-25
 - editing values, in TestManager 8-26
 - example of column definition 8-23
 - field values and 8-39
 - length of 8-21
 - maximum number 8-3, 8-12, 8-18, 8-42
 - names correspond to script variables 8-20, 8-42
 - setting numeric ranges in 8-22
 - setting unique values in 8-21
 - unique 8-37
 - values supplied by data types 1-4, 8-6
- columns in LogViewer
 - changing order 10-8
 - changing widths 10-8
 - Log file section 10-5
- columns in queries 15-10
- command-line options D-1
- comma-separated-value format for datapools 8-39
- comments in GUI scripts 5-8
- company names data type C-2

- Comparators
 - Grid 13-1
 - Image 14-1
 - Object Properties 11-1
 - Text 12-1
 - viewing verification points in 10-9
- comparing differences
 - in Grid Comparator 13-6
 - in Image Comparator 14-6
 - in Object Properties Comparator 11-6
 - in Text Comparator 12-5
- compiling
 - locating errors 7-9
 - scripts and library source files 7-7
 - virtual user scripts 2-16
- Computer Listing reports 16-4
- configuring
 - Microsoft Internet Information Server E-16
 - Microsoft Personal Web Server E-19
- configuring Windows 2000 Web servers E-16
- constant value data type C-8
- constant values, examining 7-13
- Content identification method 6-16
- cookie prompt, disabling in Internet Explorer 19-2
- copying
 - builds 3-7
 - data from actual to baseline file in Grid Comparator 13-10
 - data from actual to baseline file in Object Properties Comparator 11-11
 - data from actual to baseline file in Text Comparator 12-6
 - data from baseline file in Grid Comparator 13-10
 - data from baseline file in Object Properties Comparator 11-10
 - data from baseline file in Text Comparator 12-6
 - data tests B-8
 - datapools 8-27
 - log filters in LogViewer 10-12
 - low-level scripts 7-5
 - masks in Image Comparator 14-11
 - OCR regions in Image Comparator 14-15
 - references to test documents 2-2
 - reports 16-17
 - user-defined data types 8-36
 - verification points 6-25
- Coverage reports
 - about 16-2
 - creating 16-7
 - opening 16-14
 - running 16-11, 16-13
- creating
 - builds 3-5
 - Coverage reports 16-7
 - datapools outside Rational Test 8-38
 - datapools, in TestManager 8-18
 - Listing reports 16-4
 - log filters in LogViewer 10-11
 - masks in Image Comparator 14-10
 - OCR regions in Image Comparator 14-13
 - queries 15-4
 - query filters 15-6, 15-7
 - references to test documents 2-2
 - Test Results Progress reports 16-15
 - user-defined data types 8-9
- credit card numbers 8-24
- .csv datapool files 8-3, 8-29
- cursors
 - datapool 8-3
 - disabling wrapping for unique row retrieval 8-37
 - synchronizing in Grid Comparator 13-6
- custom data tests B-1
- custom object classes 4-14
- customer support xxiii

Index

customizing

- field labels and values 2-23
- script properties 2-16
- scripts and schedules 2-22

cutting

- data from baseline file in Grid Comparator 13-10
- data from baseline file in Object Properties Comparator 11-10
- data from baseline file in Text Comparator 12-6
- masks in Image Comparator 14-11
- OCR regions in Image Comparator 14-15

D

data grids

- definition 6-15
- working with 6-19

Data options

- in Planning Coverage reports 16-8
- in Test Results Progress reports 16-16

data tests B-1

- copying B-8
- creating B-5
- deleting B-8
- example B-2
- expressions B-6
- renaming B-8

data types

- assigning to a datapool column 8-20
- copying 8-36
- creating 8-9
- deleting 8-36
- determining which data types you need 8-8
- editing values in standard data types 8-32
- editing values in user-defined data types 8-31
- importing user-defined 8-35
- list of standard data types C-1
- minimum and maximum values C-9

renaming 8-35

- role of 1-4, 8-6
- standard and user-defined 8-7

data, testing in objects 6-4

datapools

- access order 8-3
- adding commands to GUI scripts 8-13
- assigning data types to 8-20
- copying 8-27
- creating in TestManager 8-18
- creating outside Rational Test 8-38
- cursors 8-3
- data types 1-4, 8-6
- deleting 8-28
- deleting columns from 8-26
- editing column definitions, in TestManager 8-25
- editing values, in TestManager 8-26
- example of column definition 8-23
- example of value generation 8-24
- exporting 8-30
- field separators 8-39
- files 8-3, 8-29
- finding data types for 8-8
- generating data, in TestManager 8-18
- importing from another project 8-29
- importing from outside Rational Test 8-28
- limits 8-3
- maximum number of columns 8-3, 8-12, 8-18, 8-42
- numeric ranges in 8-22
- overview 8-4
- planning 8-4
- populating with values, in TestManager 8-18
- renaming 8-27
- role of 8-2, 8-4
- row access order 8-3
- separator characters 8-39
- setting unique values in 8-21

- datapools (continued)
 - structure 8-39
 - unique row retrieval 8-37
 - where stored 8-3
- DataWindow Contents data test 21-7
- DataWindows in PowerBuilder
 - computed fields 21-8
 - expression values 21-5
 - hidden 21-6
 - recording actions 21-3
- dates
 - Julian C-4, C-5
 - setting ranges C-2, C-3, C-4
- dates data types C-2, C-3, C-4
- debugging GUI scripts 7-9
 - animation mode 7-13
 - clearing breakpoints 7-11
 - executing to a selected line 7-13
 - setting breakpoints 7-11
 - stepping into scripts 7-11
- decimal numbers 8-21, C-5
- declarations, global 5-15
- Default object order preference 4-10
- defects
 - entering in ClearQuest 10-18
 - finding 10-21
 - generating from LogViewer 10-18
 - modifying 10-21
 - TestStudio defect form 10-16
 - tracking 10-14
- defining
 - custom field labels and values 2-23
 - datapool columns 8-25
 - default build view 3-5
- delay values
 - adding to scripts 5-10
 - options for GUI script playback 9-7
- deleting
 - build states 3-10
 - builds 3-7
 - columns in queries 15-10
 - data tests B-8
 - datapool column definitions 8-26
 - datapools 8-28
 - LoadTest schedules 2-21, 15-3
 - log files in LogViewer 10-6
 - log filters in LogViewer 10-12
 - low-level scripts 7-6
 - masks in Image Comparator 14-12
 - object class mappings 4-15
 - OCR regions in Image Comparator 14-16
 - properties in Object Properties Comparator 11-7
 - references to test documents 2-2
 - reports 16-17
 - requirements 2-22
 - scripts 2-18, 7-15, 15-3
 - sessions 15-3
 - user-defined data types 8-36
 - verification points 6-25
- delimiters for datapool fields 8-39
- design specifications, referencing 2-15
- detaching
 - schedules from test requirements 2-20
 - scripts from test requirements 2-15
- Development Coverage reports 16-2, 16-9
- diagnostic tools options 9-11
- differences
 - changing color of, in Image Comparator 14-7
 - changing how determined, in Image Comparator 14-7
 - comparing in Object Properties Comparator 11-6
 - displaying in Image Comparator 13-1, 14-6
 - locating in Grid Comparator 13-6
 - locating in Image Comparator 14-6

Index

differences (continued)

- locating in Object Properties Comparator 11-7
- locating in Text Comparator 12-5
- playback/environmental 10-10

disabling cookie prompt in Internet Explorer 19-2

display options

- in Development Coverage reports 16-9
- in Execution Coverage reports 16-10
- in Grid Comparator 13-5
- in Planning Coverage reports 16-8

displaying

- builds in Asset Browser 3-4
- log properties 3-9
- masks in Image Comparator 14-9
- schedule statistics 2-21
- script statistics 2-16

divide by zero errors, detecting 9-16

documenting GUI scripts 5-8

DropDown Contents data test 21-7

DropDownDataWindows in PowerBuilder 21-7

DropDownListBoxes in PowerBuilder 21-7

duplicating

- masks in Image Comparator 14-11
- OCR regions in Image Comparator 14-16

DWCColumns in PowerBuilder 21-7

E

editing

- build states 3-10
- data in baseline file in Grid Comparator 13-8
- data in data grid 6-21
- datapool column definitions in TestManager 8-25
- datapool values in TestManager 8-26
- log filters in LogViewer 10-11
- masks in Image Comparator 14-9
- menu items in Grid Comparator 13-9

references to test documents 2-2

requirement properties 2-7, 2-21

scripts 7-1, 7-2

standard data type values 8-32

user-defined data type definitions 8-32

user-defined data type values 8-31

values in Properties list in Comparator 11-9

verification points 6-23

editing baseline file

in Grid Comparator 13-8

in Object Properties Comparator 11-9

in Text Comparator 12-5

editor for manual scripts E-3

empty string data type C-8

Enabler for Oracle Forms 18-2

enabling applications for testing 4-5

ending recording of GUI scripts 4-23

entering defects 10-18

environment differences 10-10

error recovery options for GUI script playback 9-9

errors

detecting during playback 9-16

locating after compiling 7-9

evaluating failures in LogViewer 10-9

Excel, creating datapool files with 8-40

executable files, starting 5-1

executing to a selected line during debugging 7-13

Execution Coverage reports 16-2, 16-10, 16-13

expected results for verification points 6-9

exponential notation data type C-5

exporting datapools 8-30

expressions

in data tests B-6

in PowerBuilder applications 21-5

Extension Manager

Java 20-1

Oracle Forms 18-8

- PeopleTools 22-2
- PowerBuilder 21-2
- Visual Basic 17-3
- external C libraries 2-16
- external scripts
 - about E-26
 - logging results of E-26, E-28
 - planning E-26
 - running E-29
 - viewing results of run E-29

F

- failures
 - evaluating in LogViewer 10-9
 - setting error recovery options 9-9
 - See also* differences
- features, adding to GUI scripts 5-1
 - adding to existing GUI scripts 7-2
 - comments 5-8
 - delay values 5-10
 - inserting calls to scripts 5-3
 - log messages 5-9
 - starting applications 5-1
 - timers 5-6
 - verification points 5-4
- field separator characters for datapools 8-39
- fields in datapools. *See* columns in datapools
- File Comparison verification point 6-3
- File Existence verification point 6-3
- file types
 - .csv (datapool files) 8-3, 8-29
 - .ord (object order preference files) 4-14
 - .rec (as SQABasic library source files) 5-12
 - .rec (SQABasic script files) 2-10
 - .sbh (SQABasic header files) 5-14
 - .sbl (SQABasic library source files) 5-12
 - .sbx (SQABasic library runtime files) 5-13

- .spc (datapool specification files) 8-3, 8-29
- .sqa (LOV text files) 18-17
- files
 - comparing 6-3
 - datapool file location 8-3
 - testing existence of 6-3
- filtering Log Event column in LogViewer 10-11
- filters
 - creating complex query 15-7
 - creating simple query 15-6
 - for logs 10-11
- Find Sub String Case-Insensitive verification method 6-14
- Find Sub String Case-Sensitive verification method 6-14
- first names data type C-6
- float data types C-5
- floating point numbers 8-21, C-5
- functional specifications, referencing 2-15

G

- gender data type (M, F) C-5
- general protection faults, detecting 9-16
- generating
 - defects 10-18
 - log files 3-2
 - reports in LogViewer 10-13
 - values in datapools, in TestManager 8-18, 8-24
- Generic object type, defining unknown objects as 4-9, 4-21
- global
 - declarations 5-15
 - header files 5-15
 - library source files 5-13
- global.sbh file 5-15
- global.sbl file 5-13
- GPFs, detecting during playback 9-16

Index

Grid Comparator 13-1

- comparing actual and baseline files 13-6
- copying data from actual to baseline file 13-10
- cutting or copying data from baseline file 13-10
- editing baseline file 13-8
- editing data in baseline grid 13-8
- editing menu items 13-9
- grid window 13-4
- locating differences 13-6
- main window 13-3
- pasting data into baseline file 13-10
- replacing baseline file 13-11
- saving baseline file 13-11
- setting display options 13-5
- starting 13-2, 13-3
- synchronizing cursors 13-6
- synchronizing scroll bars 13-5, 13-6
- transposing grid data 13-5
- user interface 13-3
- using keys to compare data 13-7

grid editor for manual scripts E-3, E-6

grid window in Grid Comparator 13-4

grouping scripts for playback 4-26

GUI Insert toolbar 4-20

GUI playback options 9-4

GUI Record toolbar 4-20

GUI recording options 4-6

GUI scripts

- adding features 7-2
- adding user actions 7-2
- autonaming 4-7
- coding manually 4-24
- creating modular scripts 4-3
- datapools and 8-12
- debugging 7-9
- deleting 2-18

ending recording 4-23

pausing recording 4-20

playing back 9-1, 9-18

recording 4-1, 4-16

recording options 4-6

recording workflow 4-2

resuming recording 4-20

shell scripts 4-26

test environment 4-3

testing 4-25

viewing results of playback 9-20

when to use 2-12

See also scripts

GUI scripts and datapools 8-12

adding datapool commands 8-13

assigning datapool values to variables 8-15

associating variable names and datapool columns
8-15, 8-42

example script 8-16

substituting variables for literal values 8-14

tips during recording 8-12

H

header files 5-11, 5-14

help desk xxiii

hexadecimal data type C-5

hidden object, selecting 6-12

hot keys

restoring Robot window during recording 4-19

turning low-level recording on and off 4-22

hotline support xxiii

HTML support

testing applications 19-1

testing data in elements 19-4, 19-7

I

IDE applications

- enabling for testing 4-5

- HTML 19-1

- Java 20-1

- Oracle Forms 18-1

- PeopleTools 22-1

- PowerBuilder 21-1

- Visual Basic 17-1

identification methods

- changing in Object Properties Comparator 11-12

- for verification points 6-15

identifying objects to test 6-13

IIS. *See* Microsoft Internet Information Server

IIS and Windows 2000 E-16

Image Comparator 14-1

- automatically masking differences 14-12

- changing color of masks and differences 14-7

- changing how differences are determined 14-7

- displaying differences 13-1, 14-6

- image properties 14-8

- locating differences 14-6

- main window 14-4

- Mask/OCR list 14-5

- masks 14-9

- moving image 14-8

- replacing baseline file 14-17

- saving baseline file 14-18

- starting 14-2

- status bar 14-6

- unexpected active windows 14-18

- user interface 14-4

- zooming image 14-8

images, testing 6-5, 14-1

IME (Input Method Editor) 8-9, 8-11, 8-20, 8-23

importing

- datapools from another project 8-29

- datapools from outside Rational Test 8-28

- requirements from PowerBuilder libraries (.pbl)
2-7

- user-defined data types 8-35

include files 2-16

Input Method Editor 8-11, 8-20, 8-23

Input Method Editor (IME) 8-9

inserting

- child requirements 2-7

- columns in queries 15-10

- features in GUI scripts 5-1

- requirements 2-5

installing

- Microsoft Internet Information Server E-14

- Microsoft Personal Web Server E-14

- TestManager - Manual Script Execution
component E-14

- Web servers E-14

- Windows NT Option Pack 4.0 E-14

integer data type C-6

invalid op codes, detecting 9-16

J

Japanese characters 8-8, 8-11

Java applets and applications 20-3

Java Developer Kit (JDK) 20-3

Java Enabler 20-4

Java extension, enabling 20-1

Java Virtual Machine (JVM) 20-1

Julian date data types C-4, C-5

Index

K

- Kanji characters 8-11
- Katakana characters 8-11
- Key/Value identification method 6-17
- keyboard actions, tracking during recording 4-22
- keys
 - in unique datapool rows 8-36
 - using to compare data in columns 6-17, 13-7
- keystrokes, waiting for during playback 9-7

L

- labeling test requirement revisions 2-6
- last names data types C-6
- library source files
 - compiling 7-7
 - creating and editing 5-12
 - global 5-13
 - role of 5-11
- links, testing HTML 19-7
- listing builds sequentially or by state 3-4
- Listing reports
 - creating 16-4
 - definition 16-2
 - opening 16-7
 - running 16-5
- literal value data type C-8
- LoadTest schedules
 - attaching to test requirements 2-19
 - deleting 2-21
 - deleting from Query window 15-3
 - detaching from test requirements 2-20
 - displaying statistics for 2-21
 - planning 2-19

- locating differences
 - in Grid Comparator 13-6
 - in Image Comparator 14-6
 - in Object Properties Comparator 11-7
 - in Text Comparator 12-5
- locating failed log events in LogViewer 10-8
- Location identification method 6-16
- Log Event column 10-11
- log events in LogViewer
 - collapsing and expanding 10-7
 - filtering 10-11
 - locating failed events 10-8
 - properties 10-7
- log file section of LogViewer 10-5
- log messages, adding to GUI scripts 5-9
- log options for GUI script playback 9-5
- log window, modifying in LogViewer 10-7
- logs
 - definition 9-5
 - deleting in LogViewer 10-6
 - deleting in TestManager 3-8
 - displaying properties of 3-9
 - external scripts E-26, E-28
 - generating 3-2
 - in LogViewer 10-1
 - manual scripts E-10
 - opening from LogViewer 10-6
 - opening from TestManager 3-9
 - renaming in TestManager 3-8
- LogViewer 10-1
 - changing column order 10-8
 - collapsing log events 10-7
 - columns 10-5
 - deleting log files 10-6
 - entering defects 10-14, 10-18
 - evaluating failures 10-9
 - expanding log events 10-7

- filtering Log Event column 10-11
 - generating reports 10-13
 - locating failed log events 10-8
 - log filters 10-11
 - main window 10-5
 - modifying log window 10-7
 - opening from TestManager 3-9
 - opening log files 10-6
 - reports 10-13
 - setting default report layout 10-13
 - starting 10-3
 - usage scenarios 10-2
 - user interface 10-5
 - viewing log event properties 10-7
 - low-level recording
 - copying scripts 7-5
 - deleting scripts 7-6
 - hot keys for 4-22
 - renaming scripts 7-4
 - switching to 4-22
 - viewing scripts 7-4
- M**
- manual scripts
 - about E-1
 - editor E-3
 - external files in E-7, E-9
 - planning and creating E-3
 - running in TestManager E-7
 - viewing results of run E-10
 - manual scripts on the Web
 - about E-11
 - running E-23
 - setting options for E-24
 - troubleshooting E-22
 - Web browsers for E-21
 - mapping
 - object types and classes 4-14
 - scripts with test requirements 2-13
 - masks in Image Comparator 14-9
 - adding during recording 14-17
 - automatically masking differences 14-12
 - changing color of 14-7
 - creating 14-10
 - displaying 14-9
 - measuring duration of events 5-6
 - menu items, editing in Grid Comparator 13-9
 - Menu verification point 6-4
 - menus, testing 6-4, 6-20
 - Microsoft Excel, creating datapool files with 8-40
 - Microsoft Internet Explorer, setting up for manual scripts on the Web E-21
 - Microsoft Internet Information Server
 - configuring E-16
 - installing E-15
 - Microsoft Personal Web Server
 - configuring E-19
 - installing E-15
 - middle initials data type C-7
 - middle names data type C-6
 - modifying
 - build states 3-10
 - defects 10-21
 - object class mappings 4-15
 - Module Existence verification point 6-4
 - modules, testing existence of 6-4
 - mouse movements, tracking during recording 4-22
 - moving
 - image in Image Comparator 14-8
 - masks in Image Comparator 14-10
 - OCR regions in Image Comparator 14-14
 - schedules to another requirement 2-20
 - scripts to another requirement 2-15, 2-20

Index

multi-byte characters

- in user-defined data types 8-8, 8-9, 8-11
- typing in datapool column names 8-20, 8-23

N

names data types

- company names C-2
- first names C-6
- last names C-6
- middle initials C-7
- middle names C-6
- titles (Mr, Ms) C-6

naming scripts when recording 4-7

Netscape Navigator

- playing back scripts in 19-12
- setting up for manual scripts on the Web E-21

numbers data type C-6

Numeric Equivalence verification method 6-14

Numeric Range verification method 6-15

numeric values, testing 6-3

O

object data tests B-1

Object Data verification point 6-4, B-1

Object Finder tool 6-11

object mapping 4-14

object order preference

- creating 4-14
- selecting 4-10

Object Properties Comparator 11-1

- adding properties to test 11-7
- changing identification methods 11-12
- changing verification methods 11-11
- copying values from actual to baseline file 11-11
- copying values from baseline file 11-10
- displaying differences in baseline and actual files

11-6

- editing baseline file 11-9
- editing values in Properties list 11-9
- locating differences 11-7
- main window 11-3
- Objects hierarchy 11-4
- pasting values into baseline file 11-10, 11-11
- Properties list 11-4
- removing properties 11-7
- replacing baseline file 11-12
- saving baseline file 11-12
- starting from LogViewer 11-2
- starting from Robot 11-2
- user interface 11-3
- working in Properties list 11-6

Object Properties verification point 6-4

object recognition methods

- creating new order preference 4-14
- customizing the order 4-12
- selecting order preference 4-10

object types and classes, mapping 4-14

Object-Oriented Recording 4-22

objects

- identifying the object to test 6-13
- Object Properties Comparator 11-1
- selecting the object to test 6-10
- unknown 4-8, 4-21, B-5

Objects hierarchy in Object Properties Comparator 11-4

OCR regions

- copying 14-15
- creating 14-13
- cutting 14-15
- duplicating 14-16
- moving 14-14
- pasting 14-15
- resizing 14-14

- opening
 - Coverage reports 16-14
 - Listing reports 16-7
 - log files in LogViewer 10-6
 - Query Properties dialog box 15-4
 - scripts from LogViewer 10-10
 - test documents from TestManager 2-3
 - Test Results Progress reports 16-17

- Oracle Forms support 18-1
 - making applications testable 18-3
 - Rational Test Enabler 18-2
 - recording actions 18-9
 - testing base-table blocks and items 18-16
 - testing data 18-16
 - testing LOVs 18-17
 - testing objects 18-9
 - testing properties 18-12
 - testing record groups 18-17
 - Try it! sample applet 18-2
 - verifying that extension is loaded 18-8
- .ord files for object order preferences 4-14

P

- packed decimal data type C-7

- pasting

- data to baseline file in Grid Comparator 13-10
 - data to baseline file in Object Properties Comparator 11-10, 11-11
 - data to baseline file in Text Comparator 12-6
 - masks in Image Comparator 14-11
 - OCR regions in Image Comparator 14-15
- pausing recording of GUI scripts 4-20
- PeopleTools support 22-1

- commands 22-3
 - testing data 22-3
 - testing properties 22-2
 - verifying that extension is loaded 22-2

- PerformanceStudio 1-13

- phone numbers data types C-7

- planning

- custom fields in scripts 2-23
 - datapools 8-4
 - LoadTest schedules 2-19
 - scripts 2-10

- Planning Coverage reports 16-2, 16-8

- playback differences 10-10

- playing back GUI scripts 9-1, 9-18

- acknowledging results 9-5
 - delays between commands 9-7
 - delays between keystrokes 9-7
 - detecting GPFs 9-16
 - error recovery options 9-9
 - in debugging mode 7-10
 - log options 9-5
 - playback options 9-4
 - Trap options 9-16
 - under PureCoverage 9-11
 - under Purify 9-11
 - under Quantify 9-11
 - unexpected active window options 9-10
 - wait state options 9-7

- PlayJrnl command for low-level recording 4-23

- populating datapools

- example 8-24
 - in TestManager 8-18

- PowerBuilder support 21-1

- DataStore controls 21-6
 - DataWindows 21-3, 21-5, 21-6, 21-8
 - DropDownDataWindows/ListBoxes 21-7
 - importing requirements from a library (.pbl) 2-7
 - Try it! sample applet 21-2
 - verifying that extension is loaded 21-2

- prefixes, autonaming GUI scripts 4-7

- preprocessor directives 2-17

Index

printing

- reports in LogViewer 10-13

- scripts 7-7

- SQABasic files 7-7

Progress report 16-3

project header files, creating and editing 5-14

projects

- changing 1-16

- working with 2-4

properties

- adding and removing in Object Properties Comparator 11-7

- log event 10-7

- testing objects 6-4

Properties list in Object Properties Comparator 11-4

- editing values in 11-9

- working in 11-6

properties of scripts

- defining in Robot 4-24

- defining in TestManager 2-11

PureCoverage, using with Robot 9-11

Purify, using with Robot 9-11

PWS. *See* Microsoft Personal Web Server

Q

Quantify, using with Robot 9-11

queries

- choosing the fields to display 15-5

- creating complex filters 15-7

- creating simple filters 15-4

- running 15-2

- setting options 15-9

- specifying sort order 15-5

Query Update button 15-3

Query window 15-2, 15-3, 15-10

Quick Reports in LogViewer

- generating 10-13

- printing 10-13, 10-14

- setting default layout 10-13

- working with 10-13

- writing to a file 10-13

R

random alphabetic string data type C-7

random alphanumeric string data type C-7

random datapool access 8-3

random value seed 8-22

ranges in dates C-2, C-3, C-4

Rational Administrator 1-2

Rational ClearQuest 10-14

Rational LogViewer 10-1

Rational PureCoverage, using with Robot 9-11

Rational Purify, using with Robot 9-11

Rational Quantify, using with Robot 9-11

Rational repository

- about 1-2

- managing with Rational Administrator 1-2

- selecting 1-15

Rational RequisitePro 1-14, 2-4

Rational Suite PerformanceStudio 1-13

Rational Synchronizer 1-14

Rational technical support xxiii

Rational Test Enabler for Oracle Forms 18-3

Rational TestManager - Manual Script Execution

- running manual scripts on the Web E-11

Read From File data type 8-42, C-8

- unique values 8-44

.rec library files 5-12

.rec script files 2-10

- recording GUI scripts 4-1, 4-16
 - and unknown objects 4-8, 4-21
 - creating modular scripts 4-3
 - ending 4-23
 - mapping object types and classes 4-14
 - pausing 4-20
 - process 4-1
 - recording options 4-6
 - restoring main window 4-19
 - resuming 4-20
 - selecting object order preference 4-10
 - workflow 4-2
- records in datapools. *See* rows in datapools
- referencing
 - include files and external C libraries 2-16
 - specification files 2-15, 2-20
 - test documents 2-2
- Region Image verification point 6-5, 14-1
- regions of screen, testing 6-5
- regression testing phase 9-2
- renaming
 - builds 3-7
 - data tests B-8
 - datapools 8-27
 - log filters in LogViewer 10-12
 - low-level scripts 7-4
 - references to test documents 2-2
 - reports 16-17
 - user-defined data types 8-35
 - verification points 6-25
- replacing baseline file
 - in Grid Comparator 13-11
 - in Image Comparator 14-17
 - in Object Properties Comparator 11-12
 - in Text Comparator 12-7
- Report Layout Editor 16-5
- reports in LogViewer
 - generating 10-13
 - printing 10-14
 - setting default report layout 10-13
- reports in TestManager
 - copying 16-17
 - Coverage reports 16-7
 - deleting 16-17
 - layouts 16-5
 - Listing reports 16-4
 - renaming 16-17
 - Test Results Progress reports 16-15
- repositories, shared E-14
- repository
 - managing with Rational Administrator 1-2
 - selecting 1-2
- requirements
 - attaching schedules to 2-19
 - attaching scripts to 2-13
 - attributes and revisions of 2-6, 2-21
 - editing 2-7, 2-21
 - labeling revisions of 2-6
 - managing 2-21
 - types of 2-6
- Requirements Hierarchy
 - building 2-5
 - definition 2-3
 - expanding and collapsing 2-9
- RequisitePro 1-14, 2-4
- resizing
 - masks 14-11
 - OCR regions 14-14
- results of playback, viewing 9-20
- resuming recording of GUI scripts 4-20
- revisions, test requirement 2-6
- Robot main window, restoring during recording 4-19
- row access order 8-3
- rows in data grid, transposing with columns 6-23

Index

rows in datapools

- access order 8-3
- maximum number 8-3
- records and 8-39
- unique 8-37

rtresults.exe, logging results and E-26

running

- Execution Coverage reports 16-13
- external scripts E-29
- GUI scripts 9-1
- Listing reports 16-5
- manual scripts in TestManager E-7
- manual scripts on the Web E-23
- Planning Coverage or Development Coverage reports 16-11
- queries 15-2
- Test Results Progress reports 16-16

S

saving baseline file

- in Grid Comparator 13-11
- in Image Comparator 14-18
- in Object Properties Comparator 11-12
- in Text Comparator 12-7

saving scripts and SQABasic files 7-7

.sbh header files 5-14

.sbl library files 5-12

.sbx library runtime files 5-13

Schedule Listing reports 16-4

scientific notation data type C-5

script command failures 9-9

Script Listing reports 16-4

script properties

- defining in Robot 4-24
- defining in TestManager 2-11

scripts

- attaching to test requirements 2-13
- compiling 7-7
- compiling virtual user 2-16
- customizing properties 2-16
- deleting 2-18, 7-15
- deleting from Query window 15-3
- detaching from test requirements 2-15
- displaying statistics for 2-16
- editing 7-1
- external E-26
- manual E-1, E-11
- moving to another requirement 2-15, 2-20
- opening from LogViewer 10-10
- overview 2-10
- planning 2-10
- printing 7-7
- saving 7-7
- variable names and datapool column names 8-20
- See also* GUI scripts

scroll bars, synchronizing in Grid Comparator 13-5, 13-6

scrolling in Text Comparator 12-4

seed for random datapool values 8-22

selecting objects to test 6-10

selecting reports 16-3

separator characters for datapools 8-39

sequential datapool access 8-3

- unique row retrieval and 8-38

Session Listing reports 16-4

sessions, deleting from Query window 15-3

setting breakpoints 7-11

sharing repositories E-14

shell scripts 4-26

shuffle datapool access 8-3

- unique row retrieval and 8-38

single step execution during debugging 7-11

software requirements for Web servers E-13

- sorting queries 15-5
 - space data type C-8
 - .spc datapool specification files 8-3, 8-29
 - specification files, referencing 2-15
 - .sqa files for LOV objects 18-17
 - SQABasic files
 - compiling 7-7
 - header files 5-15
 - library source files 5-13
 - printing 7-7
 - saving 7-7
 - template file 5-16
 - SQABasic header files 5-11, 5-14
 - sqatrap.log 9-17
 - stack overflows, detecting 9-16
 - standard data types
 - editing values in 8-32
 - list of C-1
 - minimum and maximum values C-9
 - role of 8-7
 - when to use 8-8
 - Start Application command 5-2
 - Start Browser command 5-2
 - Start Java Application command 5-2
 - starting
 - applications 5-1
 - Grid Comparator 13-2
 - Image Comparator 14-2
 - LogViewer 10-3
 - Object Properties Comparator 11-2
 - Text Comparator 12-2
 - state abbreviations data type C-8
 - statistics, displaying 2-16
 - stepping into scripts during debugging 7-11
 - stepping out of called scripts during debugging 7-11
 - stepping over command lines during debugging 7-11
 - steps in manual scripts E-1
 - stopping recording of GUI scripts 4-23
 - street names data type C-2
 - string constant data type C-8
 - structure of datapools 8-39
 - support, technical xxiii
 - Swing foundation classes, installing 20-8
 - Synchronizer 1-14
 - synchronizing
 - cursors in Grid Comparator 13-6
 - GUI scripts with application 5-10
 - playback with application 9-7
 - scroll bars in Grid Comparator 13-5, 13-6
- ## T
- tables, testing HTML 19-7
 - technical support xxiii
 - template file 5-16
 - test development phase 9-2
 - Test Document Listing reports 16-4
 - test documents
 - creating 2-2
 - editing references 2-2
 - renaming references to 2-2
 - test environment
 - setting up for playback 9-3
 - setting up for recording 4-3
 - test plans 2-1
 - test requirements
 - attributes of 2-6
 - definition 2-3
 - See also* requirements
 - Test Results Progress reports
 - creating 16-15
 - definition 16-3
 - opening 16-17
 - running 16-16

Index

- TestManager - Manual Script Execution component
 - installing E-14
- TestManager reports 16-17
- testproc.tpl file 5-16
- TestStudio defect schema and form 10-15
- Text Comparator 12-1
 - comparing actual and baseline files 12-5
 - copying data from actual to baseline file 12-6
 - cutting or copying data from baseline file 12-6
 - editing baseline file 12-5
 - locating differences 12-5
 - main window 12-3
 - pasting data into baseline file 12-6
 - replacing baseline file 12-7
 - saving baseline file 12-7
 - scrolling in text windows 12-4
 - starting from LogViewer 12-3
 - starting from Robot 12-2
 - text windows 12-4
 - user interface 12-3
 - viewing verification point properties 12-5, 13-7
 - Word Wrap option 12-4
- text editor for manual scripts E-3, E-6
- text files, assigning values to a datapool column 8-42, C-8
- text windows of Text Comparator 12-4
- time data types C-8
- timeout values
 - for script playback 9-7
 - for verification points 6-8
- timers in GUI scripts 5-6
 - inserting 5-7
 - playing back scripts 5-8
 - uses for 5-6
- Title identification method 6-16
- toolbars, working with A-1
- top menus, testing 6-20

- transposing columns and rows in data grid 6-23
- transposing grid data in Grid Comparator 13-5
- Trap options for GUI script playback 9-16
- Trap utility
 - setting options to detect GPFs 9-16
- troubleshooting
 - manual scripts on the Web E-22
 - Web servers E-23
- types of reports 16-1

U

- U.S. cities data type C-2
- U.S. state abbreviations data type C-8
- UAEs, detecting 9-16
- UNC, using for directory names E-14, E-24
- unexpected active windows
 - definition 9-10
 - detecting during playback 9-10
 - image files in Image Comparator 14-18
 - options for GUI script playback 9-10
- Uniform Naming Convention. *See* UNC
- unique datapool rows
 - guidelines for 8-36
 - Read From File data type and 8-44
 - setting unique values 8-21
 - user-defined data types and 8-10
- unknown objects
 - controlling how Robot responds to 4-8
 - defining during recording 4-21
 - defining while creating data tests B-5
- unrecoverable application errors, detecting 9-16
- user actions
 - adding to existing GUI scripts 7-2
 - definition 4-1
- User Listing reports 16-4

- user-defined data types
 - automatically generating values for 8-33
 - copying 8-36
 - creating 8-9
 - deleting 8-36
 - editing definitions of 8-32
 - editing values in 8-31
 - importing 8-35
 - renaming 8-35
 - role of 8-7
 - unique values 8-10
 - when to use 8-8

User-Defined verification method 6-15

V

- Value identification method 6-17
- variable names, and datapool column names 8-20, 8-42
- variable values, examining 7-13
- Variables window 7-13
- verification methods
 - changing in Object Properties Comparator 11-11
 - for verification points 6-14
- verification point properties
 - in Object Properties Comparator 11-7
 - in Text Comparator 12-5
- verification points 6-1
 - adding 5-4
 - before you create 6-6
 - copying 6-25
 - definition 4-2
 - deleting 6-25
 - editing 6-23
 - expected results 6-9
 - failures 9-9
 - identification methods 6-15

- identifying objects to test 6-13
- library files and 5-11
- manual scripts E-1
- renaming 6-25
- selecting objects to test 6-10
- types 6-3
- verification methods 6-14
- viewing in Comparators 6-24, 9-21, 10-9
- wait state values 6-8

Verify that selected field is blank verification method 6-15

viewing

- datapool values in TestManager 8-26
- log event properties in LogViewer 10-7
- logs 3-9
- test documents 2-3
- user-defined data type values 8-31
- verification points in the Comparators 10-9

virtual user scripts 2-12, 2-16

Visual Basic support 17-1

- Try it! sample applet 17-3

- verifying that extension is loaded 17-3

VU compiler 2-16

W

wait state options for GUI script playback 9-7

Web browsers, setting up for manual scripts on the Web E-21

Web servers

- installing E-14

- troubleshooting E-23

Web Site Compare verification point 6-5

Web Site Scan verification point 6-5

Window Existence verification point 6-5

Window Image verification point 6-5, 14-1

Index

windows

- setting wait values for 9-7

- testing existence of 6-5

- testing images 6-5

- unexpected during playback 9-10

Windows 2000 and IIS (Internet Information Server)

- E-16

Word Wrap option in Text Comparator 12-4

writing reports to a file in LogViewer 10-13

Z

zip code data types C-9

zoned decimal data type C-9

zooming image in Image Comparator 14-8