Using Rational Robot

VERSION 2001.03.00

PART NUMBER 800-023885-000

support@rational.com
http://www.rational.com

Rational®
the e-development company™

▸ ▸ ▸ 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.

Audience

This manual is intended for application developers, quality assurance managers, and quality assurance engineers.

Other Resources

- This product contains online Help. From the main toolbar, choose an option from the **Help** menu.
- All manuals are available online, either in HTML or 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.

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:

Your Location	Telephone	Facsimile	E-mail
North America	(800) 433-5444 (toll-free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4545-201 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously reported problem)

▸ ▸ ▸ 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 projects, which store your testing information.
- **Rational TestManager Log** – 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 Rational Projects with the Administrator

You use the Rational Administrator to create and manage projects.

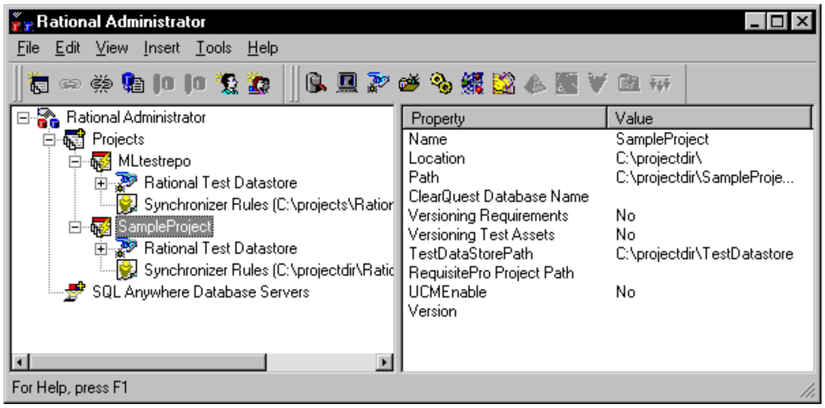
Rational projects store application testing information, such as scripts, verification points, queries, and defects. Each project consists of a database and several directories of files. All Rational Test components on your computer update and retrieve data from the same active project.

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

Use the Administrator to:

- Create a project under configuration management.
- Create a project outside of configuration management.
- Connect to a project.
- See projects that are not on your machine (register a project).
- Delete a project.
- Create and manage users and groups for a Rational Test datastore.
- Create and manage projects containing Rational RequisitePro projects and Rational Rose models.
- Manage security privileges for the entire Rational project.
- Configure a SQL Anywhere database server.

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



Property	Value
Name	SampleProject
Location	C:\projectdir\
Path	C:\projectdir\SampleProje...
ClearQuest Database Name	
Versioning Requirements	No
Versioning Test Assets	No
TestDataStorePath	C:\projectdir\TestDatastore
RequisitePro Project Path	
UCMEnable	No
Version	

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

Developing Tests in Robot

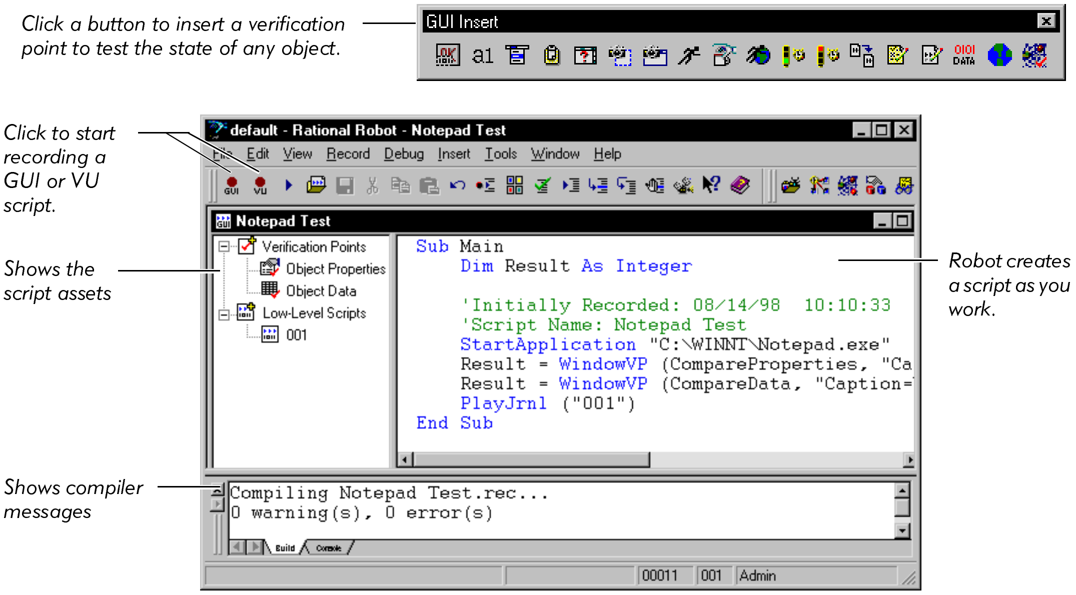
You use Robot to develop two kinds of scripts: GUI scripts for functional testing and sessions for performance testing.

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 TestManager together to record and play back sessions 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 used with sessions in performance testing.)
- **Test applications developed with IDEs** such as Java, HTML, Visual Basic, Oracle Forms, Delphi, 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.


Click a button to insert a verification point to test the state of any object.
GUI Insert
Click to start recording a GUI or VU script.
default - Rational Robot - Notepad Test
File Edit View Record Debug Insert Tools Window Help
Notepad Test
Verification Points
Object Properties
Object Data
Low-Level Scripts
001
Shows the script assets
Robot creates a script as you work.
Shows compiler messages
Build
Console
00011
001
Admin

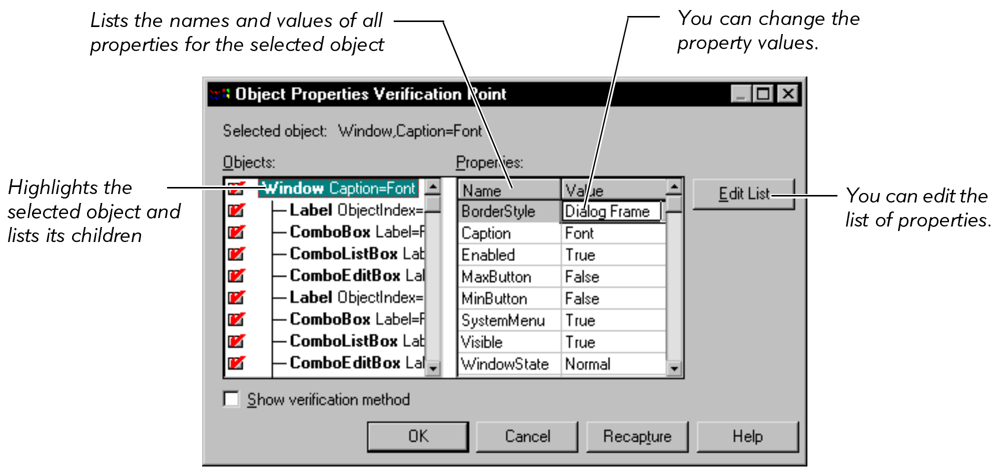
```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 08/14/98  10:10:33
    'Script Name: Notepad Test
    StartApplication "C:\WINNT\Notepad.exe"
    Result = WindowVP (CompareProperties, "Ca
    Result = WindowVP (CompareData, "Caption="
    PlayJrnl ("001")
End Sub
```

```
Compiling Notepad Test.rec...
0 warning(s), 0 error(s)
```

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.


Lists the names and values of all properties for the selected object
You can change the property values.
Highlights the selected object and lists its children
You can edit the list of properties.
Object Properties Verification Point
Objects:
Properties:
Edit List
Show verification method
OK
Cancel
Recapture
Help

Selected object: Window,Caption=Font

Objects:

- Window Caption=Font
  - Label ObjectIndex=
  - ComboBox Label=F
  - ComboListBox Lat
  - ComboEditBox Lal
  - Label ObjectIndex=
  - ComboBox Label=F
  - ComboListBox Lat
  - ComboEditBox Lal

Name	Value
BorderStyle	Dialog Frame
Caption	Font
Enabled	True
MaxButton	False
MinButton	False
SystemMenu	True
Visible	True
WindowState	Normal

Creating Datapools

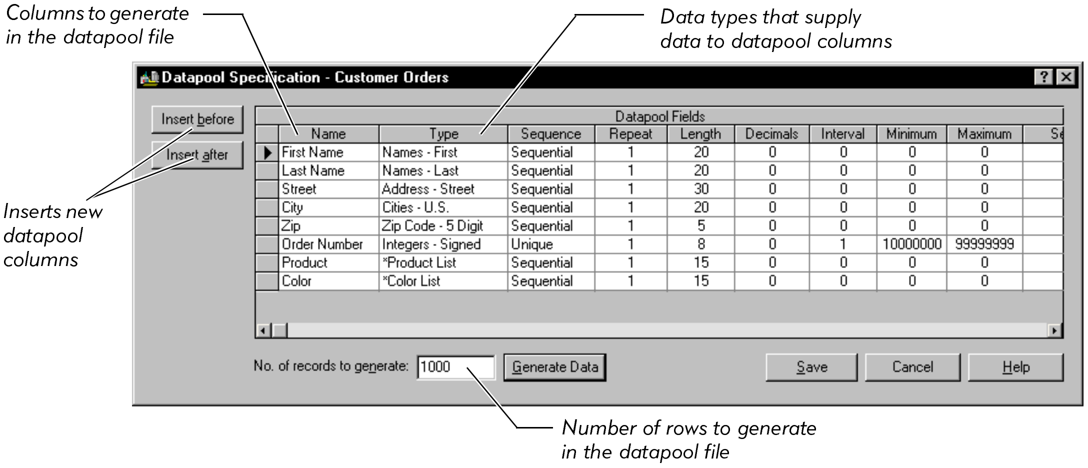
A datapool is a source of variable test data that scripts can draw from during playback.

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 exact data you recorded) is sent each time the transaction is executed.

TestManager is shipped with many commonly used data types, such as cities, states, names, and telephone area codes. In addition, TestManager lets you create your own data types.

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 or product names. When you finish defining the datapool, TestManager automatically generates the number of rows of data that you specify.

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
Datapool Specification - Customer Orders
Insert before
Insert after
Datapool Fields
Generate Data
Save
Cancel
Help

Name	Type	Sequence	Repeat	Length	Decimals	Interval	Minimum	Maximum	Se
First Name	Names - First	Sequential	1	20	0	0	0	0	
Last Name	Names - Last	Sequential	1	20	0	0	0	0	
Street	Address - Street	Sequential	1	30	0	0	0	0	
City	Cities - U.S.	Sequential	1	20	0	0	0	0	
Zip	Zip Code - 5 Digit	Sequential	1	5	0	0	0	0	
Order Number	Integers - Signed	Unique	1	8	0	1	10000000	99999999	
Product	*Product List	Sequential	1	15	0	0	0	0	
Color	*Color List	Sequential	1	15	0	0	0	0	

No. of records to generate: 1000

Analyzing Results in the Log and Comparators

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

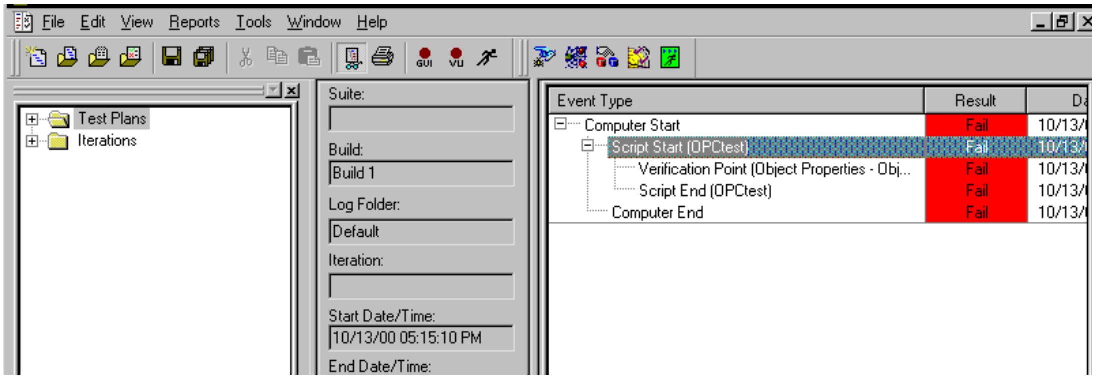
Use the log to:

- **View the results of running a script**, including verification point failures, procedural failures, aborts, and any additional playback information. Reviewing the results in the log 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 that contains a failed Object Properties verification point.



Suite:

Build: Build 1

Log Folder: Default

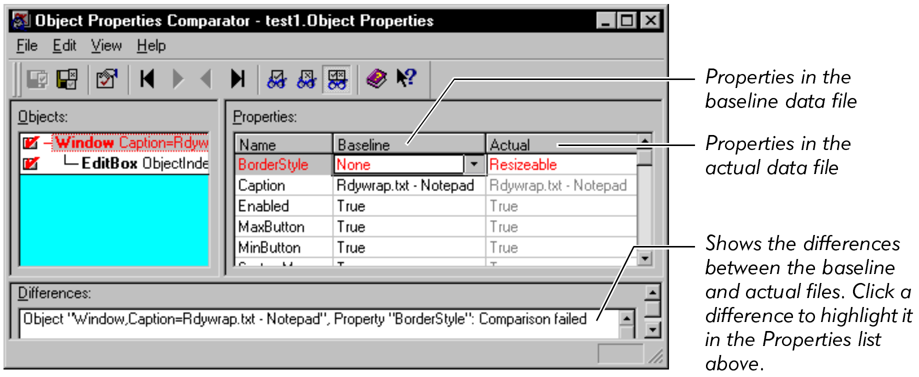
Iteration:

Start Date/Time: 10/13/00 05:15:10 PM

End Date/Time:

Event Type	Result	Da
Computer Start	Fail	10/13/
Script Start (OPCtest)	Fail	10/13/
Verification Point (Object Properties - Obj...	Fail	10/13/
Script End (OPCtest)	Fail	10/13/
Computer End	Fail	10/13/

When you select the line that contains the failed Object Properties verification point and click **View** → **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.


Object Properties Comparator - test1.Object Properties
File Edit View Help
Objects:
Window Caption=Rdyw
EditBox ObjectInde
Properties:
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.

Name	Baseline	Actual
BorderStyle	None	Resizeable
Caption	Rdywrap.txt - Notepad	Rdywrap.txt - Notepad
Enabled	True	True
MaxButton	True	True
MinButton	True	True

Differences:

Object "Window,Caption=Rdywrap.txt - Notepad", Property "BorderStyle": Comparison failed

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:

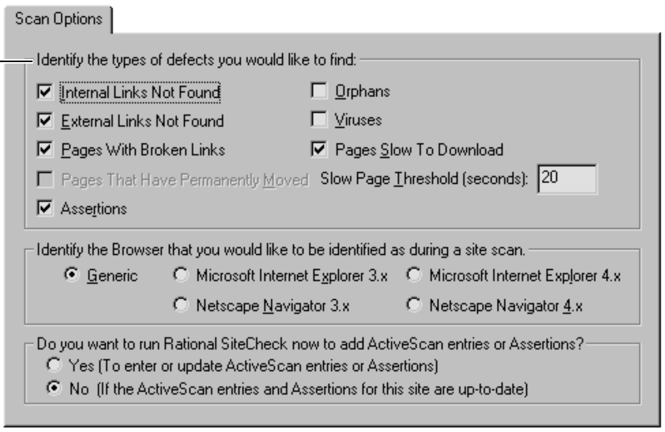
- **Visualize the structure of your Web site** and display the relationship between each page and the rest of the site.
- **Identify and analyze Web pages with active content**, such as forms, Java, JavaScript, ActiveX, and Visual Basic Script (VBScript).
- **Filter information** so that you can inspect 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.


Scan Options

Identify the types of defects you would like to find:

- [✓] Internal Links Not Found
- [✓] External Links Not Found
- [✓] Pages With Broken Links
- [ ] Pages That Have Permanently Moved
- [✓] Assertions
- [ ] Orphans
- [ ] Viruses
- [✓] Pages Slow To Download

Slow Page Threshold (seconds): 20

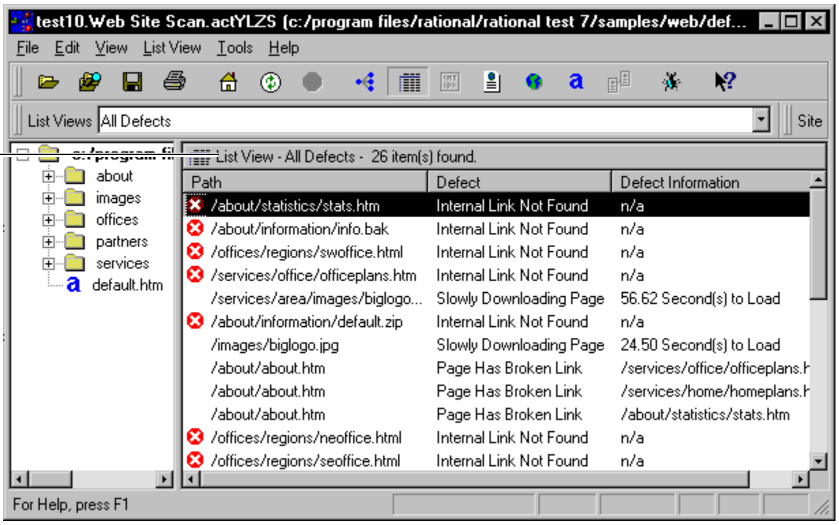
Identify the Browser that you would like to be identified as during a site scan.

[✓ Generic] Microsoft Internet Explorer 3.x Microsoft Internet Explorer 4.x Netscape Navigator 3.x Netscape Navigator 4.x

Do you want to run Rational SiteCheck now to add ActiveScan entries or Assertions?

Yes (To enter or update ActiveScan entries or Assertions) [✓ No (If the ActiveScan entries and Assertions for this site are up-to-date)]

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


test10.Web Site Scan.actYLZS (c:/program files/rational/rational test 7/samples/web/def...
File Edit View List View Tools Help
List Views All Defects
Site
c:/program fil
about
images
offices
partners
services
default.htm
List View - All Defects - 26 item(s) found.
For Help, press F1

Path	Defect	Defect Information
/about/statistics/stats.htm	Internal Link Not Found	n/a
/about/information/info.bak	Internal Link Not Found	n/a
/offices/regions/swoffice.html	Internal Link Not Found	n/a
/services/office/officeplans.htm	Internal Link Not Found	n/a
/services/area/images/biglogo...	Slowly Downloading Page	56.62 Second(s) to Load
/about/information/default.zip	Internal Link Not Found	n/a
/images/biglogo.jpg	Slowly Downloading Page	24.50 Second(s) to Load
/about/about.htm	Page Has Broken Link	/services/office/officeplans.h
/about/about.htm	Page Has Broken Link	/services/home/homeplans.h
/about/about.htm	Page Has Broken Link	/about/statistics/stats.htm
/offices/regions/neoffice.html	Internal Link Not Found	n/a
/offices/regions/seoffice.html	Internal Link Not Found	n/a

For more information about SiteCheck, see 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 TestManager, TestFactory, ClearQuest, Purify, Quantify, PureCoverage, and RequisitePro. The products and components are available based on what you have installed.

Planning and Managing Tests in TestManager

Rational TestManager is the one place to manage all testing activities--planning, design, implementation, execution, and analysis. TestManager ties testing with the rest of the development effort, joining your testing assets and tools to provide a single point from which to understand the exact state of your project.

Test Manager supports five testing activities:

- **Plan Test.** The activity of test planning is primarily answering the question, "What do I have to test?" When you complete your test planning, you end up with a test plan that defines what you are going to test. In TestManager, a test plan can contain test cases. The test cases can be organized based on test case folders.

- **Design Test**. The activity of test designing is primarily answering the question, "How am I going to do a test?" When you complete your test designing, you end up with a test design that helps you understand how you are going to perform the test case. In TestManager, you can design your test cases by indicating the actual steps that need to occur in that test. You also specify the pre-conditions, post-conditions, and acceptance criteria.
- **Implement Test**. The activity of implementing your tests is primarily creating reusable scripts. In TestManager, you can implement your tests by creating manual scripts. You can also implement automated tests by using Rational Robot. You can extend TestManager through APIs so that you can access your own implementation tools from TestManager. Because of this extensibility, you can implement your tests by building scripts in whatever tools are appropriate in your situation and organization.
- **Execute Tests**. The activity of executing your tests is primarily running your scripts to make sure that the system functions correctly. In TestManager, you can run any of the following: (1) an individual script, which runs a single implementation; (2) one or more test cases, which run the implementations of the test cases; (3) a suite, which runs test cases and their implementations across multiple computers and users.
- **Evaluate Tests**. The activity of evaluating tests is determining the quality of the system-under-test. In TestManager, you can evaluate tests by examining the results of test execution in the test log, and by running various reports.

Planning and managing tests is only one part of Rational TestManager. You also use TestManager to view the logs created by Robot. For information on using the log, see *Analyzing Results in the Log and Comparators* on page 1-6. TestManager is also used to do performance testing. See *Performance Testing with Rational TestManager* on page 1-12.

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.
- 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 TestManager log or SiteCheck.
- Modify and track defects and change requests.
- Analyze project progress by running queries, charts, and reports.

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 TestManager

Rational Testmanager is a sophisticated tool that can be used 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.

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

Doing performance testing with TestManager, 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 performance testing, see the TestManager manual 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.

Starting Robot and Its Components

Before you start using Robot, you need to have:

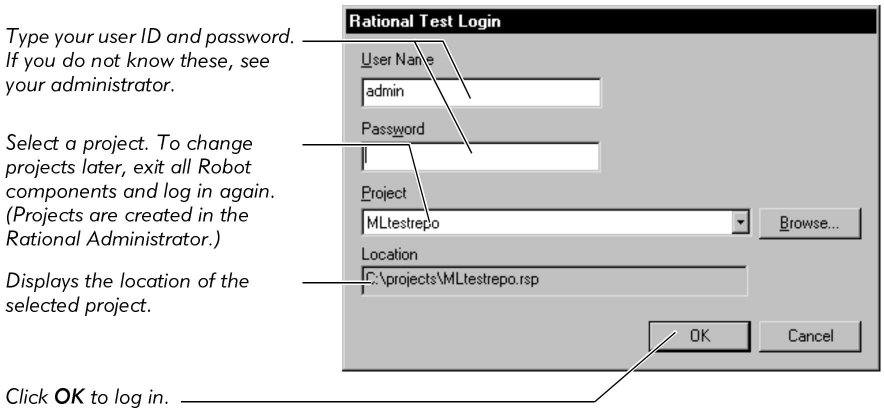
- Rational Robot installed. For information, see the *Installing Rational Testing Products* manual.
- A Rational project. For information, see the *Using the Rational Administrator* manual.

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 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 Login dialog box.

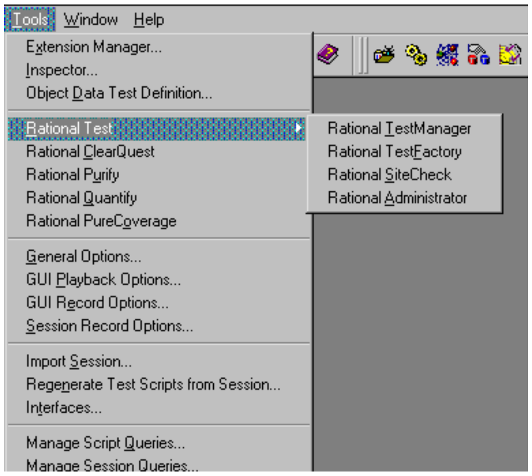

Type your user ID and password. If you do not know these, see your administrator.
Select a project. To change projects later, exit all Robot components and log in again. (Projects are created in the Rational Administrator.)
Displays the location of the selected project.
Click **OK** to log in.
Rational Test Login
Browse...
OK
Cancel

User Name: admin
Password:
Project: MLtestrepo
Location: C:\projects\MLtestrepo.rsp

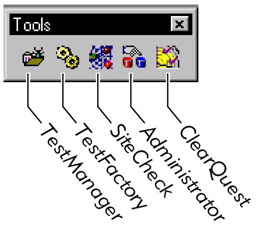
Opening Other Rational Products and Components

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	TestManager Help
Record GUI scripts	Robot	Chapter 2, *Recording GUI Scripts* Chapter 3, *Adding Features to GUI Scripts*
Create verification points to test the state of objects	Robot	Chapter 4, *Creating Verification Points in GUI Scripts*
Edit, compile, and debug scripts	Robot	Chapter 5, *Editing, Compiling, and Debugging Scripts*
Supply data values to the variables in a script during playback	Robot TestManager	*Chapter 10, Working with Datapools*
Play back GUI scripts	Robot	Chapter 9, *Playing Back GUI Scripts*
Review and analyze test results, and enter defects	TestManager	TestManager Help
View and analyze the results of verification points	Object Properties Comparator Text Comparator Grid Comparator Image Comparator	Object Properties Comparator Help Text Comparator Help Grid Comparator Help Image Comparator Help
Create and run queries to help you manage information in your projects	TestManager	TestManager Help
Create and run reports to help you manage your testing efforts	TestManager	TestManager Help
Test Visual Basic applications	Robot	Chapter 11, *Testing Visual Basic Applications*
Test Oracle Forms applications	Robot	Chapter 12, *Testing Oracle Forms Applications*
Test HTML applications	Robot	Chapter 13, *Testing HTML Applications*

To	Use this component	See
Test Java applications	Robot	Chapter 14, *Testing Java Applets and Applications*
Test PowerBuilder applications	Robot	Chapter 15, *Testing PowerBuilder Applications*
Test PeopleTools applications	Robot	Chapter 16, *Testing PeopleTools Applications*
Test Delphi applications	Robot	Chapter 17, *Testing Delphi Applications*
Create and run manual and external scripts	TestManager	TestManager Help
Manage Internet and intranet Web sites	SiteCheck	Rational SiteCheck Help

▸ ▸ ▸ Part II

Developing and Playing Back GUI Scripts

▸ ▸ ▸ C H A P T E R 2

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

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.

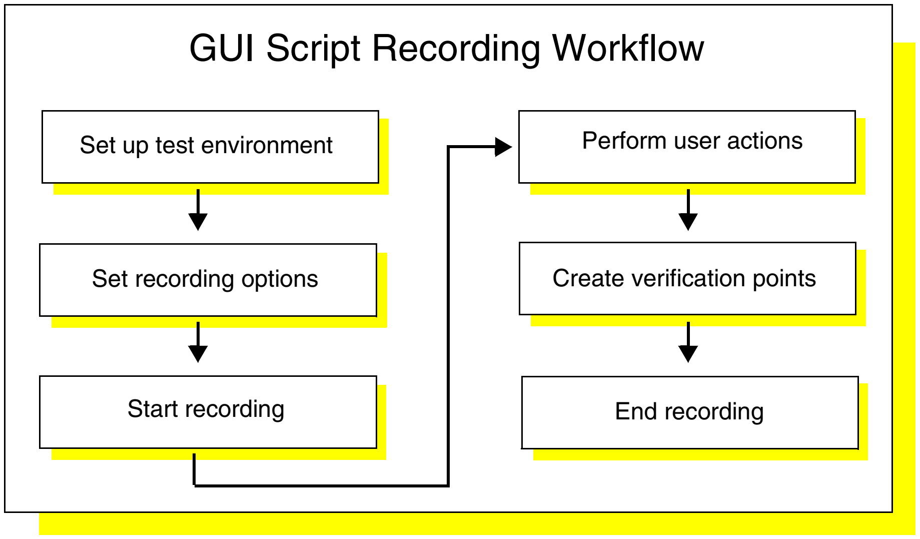
The recorded GUI script establishes the baseline of expected behavior for the application-under-test. When new builds of the application become available, you can play back the script to test the builds against the established baseline in a fraction of the time that it would take to perform the testing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

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 2-5.) 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.

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++, Delphi, 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 12, *Testing Oracle Forms Applications*.
- **HTML** – While recording or editing a script, use the **Start Browser** toolbar button to start Internet Explorer or Netscape Navigator from Robot. This loads the Rational ActiveX Test Control, which lets Robot recognize Web-based objects. For information, see Chapter 13, *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 14, *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.
- **Delphi** – Install the Rational Object Testing Library for Delphi and the Rational Test Delphi Enabler. Run the Enabler, then recompile your project to make it Delphi testable. For information, see Chapter 17, *Testing Delphi Applications*.

You can install the Enablers and the ActiveX Test Control from the Rational Software Setup wizard. For instructions, see the *Installing Rational Testing Products* manual.

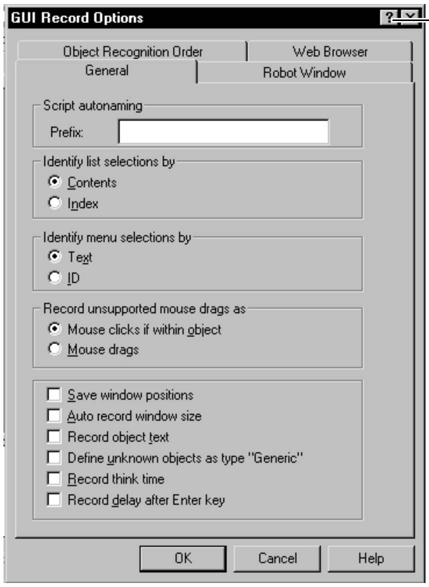
NOTE: You do not need to enable applications created in Visual Basic 5.0 and later, PowerBuilder, or PeopleTools.

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**.


GUI Record Options

Object Recognition Order | Web Browser | General | Robot Window

Script autonaming
Prefix:

Identify list selections by
(●) Contents
( ) Index

Identify menu selections by
(●) Text
( ) ID

Record unsupported mouse drags as
(●) Mouse clicks if within object
( ) Mouse drags

☐ Save window positions
☐ Auto record window size
☐ Record object text
☐ Define unknown objects as type "Generic"
☐ Record think time
☐ Record delay after Enter key

OK | Cancel | Help

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.

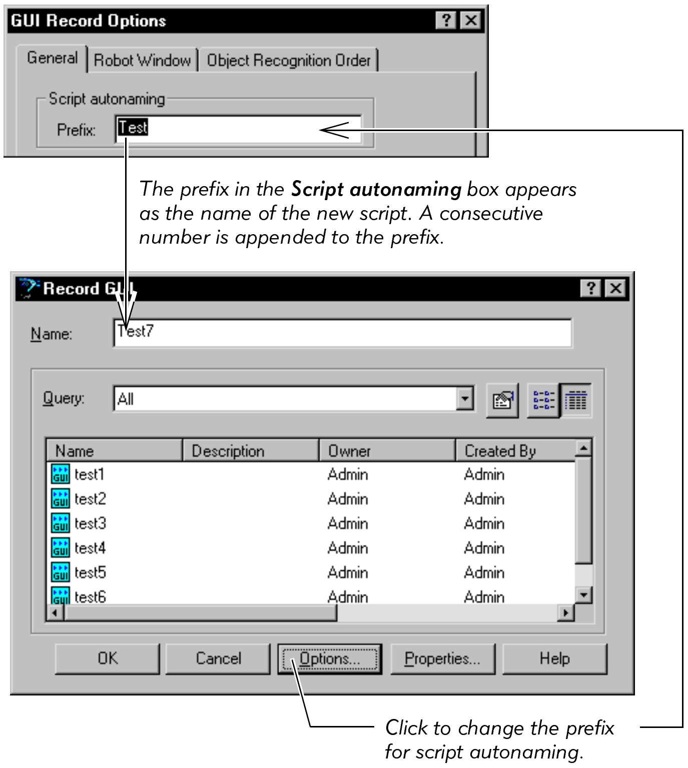
To turn on script autonaming:

1. Open the GUI Record Options dialog box. (See *Setting GUI Recording Options* on page 2-5.)
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, *Test*7 appears in the **Name** box because there are six other scripts that begin with *Test*.



Name	Description	Owner	Created By
test1		Admin	Admin
test2		Admin	Admin
test3		Admin	Admin
test4		Admin	Admin
test5		Admin	Admin
test6		Admin	Admin

*The prefix in the **Script autonaming** box appears as the name of the new script. A consecutive number is appended to the prefix.*

Click to change the prefix for script autonaming.

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 project 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 2-20.

- 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 project 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 2-5.)
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 2-13.

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

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.

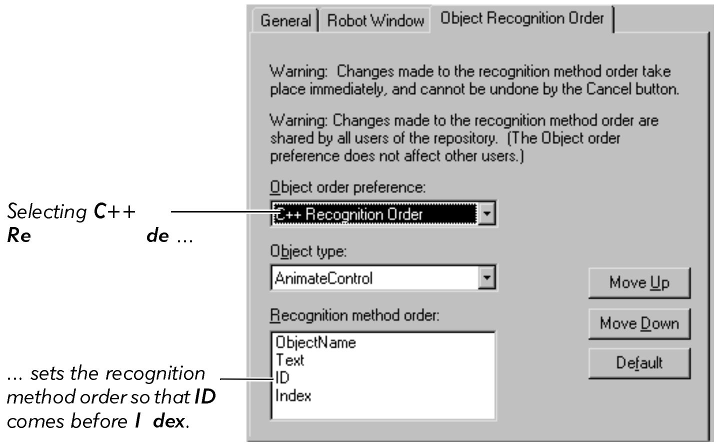
The following table describes the two predefined preferences.

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 2-5.)
2. Click the **Object Recognition Order** tab.
3. Select a preference in the **Object order preference** list.


General
Robot Window
Object Recognition Order
Move Up
Move Down
Default
Selecting ***C++*** ***Re*** ***de*** *...*
... sets the recognition method order so that ***ID*** *comes before* ***I dex****.*

Warning: Changes made to the recognition method order take place immediately, and cannot be undone by the Cancel button.

Warning: Changes made to the recognition method order are shared by all users of the repository. (The Object order preference does not affect other users.)

Object order preference: C++ Recognition Order

Object type: AnimateControl

Recognition method order:
ObjectName
Text
ID
Index

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 2-11.

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.

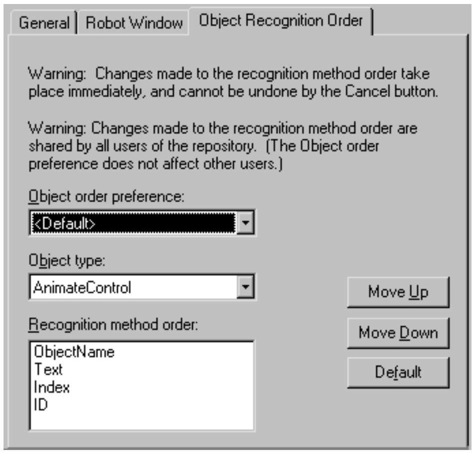
Important Notes

- 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 project. For example, if you change the order for the CheckBox object, the new order is stored in the project and affects all users of that project.
- 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 2-5.)
2. Click the **Object Recognition Order** tab.



Warning: Changes made to the recognition method order take place immediately, and cannot be undone by the Cancel button.

Warning: Changes made to the recognition method order are shared by all users of the repository. (The Object order preference does not affect other users.)

Object order preference: <Default>

Object type: AnimateControl

Recognition method order:
ObjectName
Text
Index
ID

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 project. For example, if you change the order for the CheckBox object, the new order is stored in the project and affects all users of that project.

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 project.
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 2-7, 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 project and uses it to identify the custom object during playback.

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

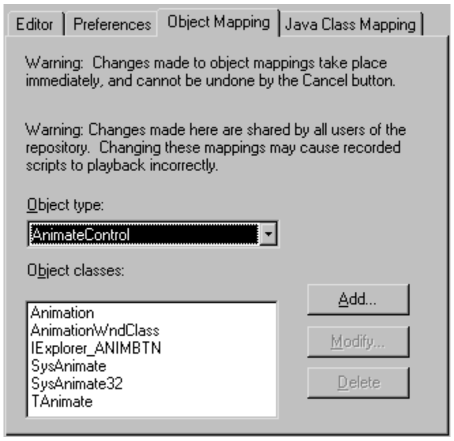
Defining an Object Class Mapping

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**.

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



Warning: Changes made to object mappings take place immediately, and cannot be undone by the Cancel button.

Warning: Changes made here are shared by all users of the repository. Changing these mappings may cause recorded scripts to playback incorrectly.

Object type: AnimateControl

Object classes:

Animation
AnimationWndClass
IExplorer_ANIMBTN
SysAnimate
SysAnimate32
TAnimate

3. 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.

4. Click **Add**.
5. Type the class name you identified in step 1 and click **OK**.
6. 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:

1. Click **Tools** → **General Options**, and then click the **Object Mapping** tab.
2. 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.

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

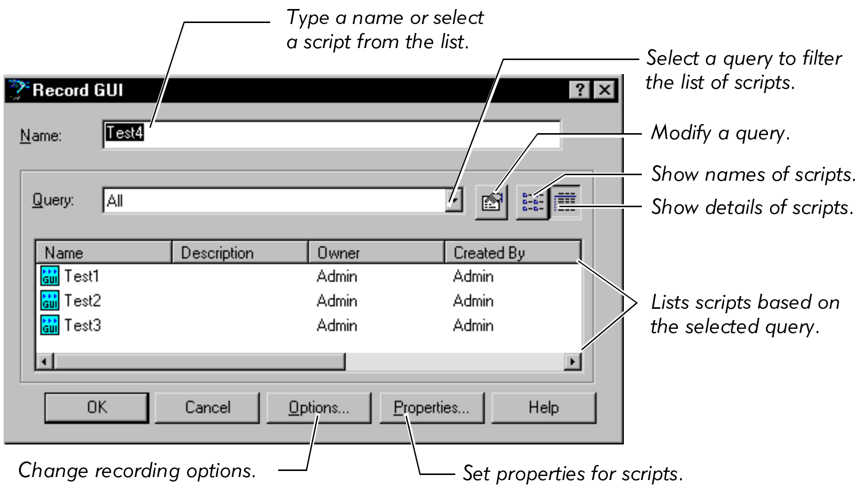
4. Do one of the following:
   - To modify the class name, click **Modify**. Change the name and click **OK**.
   - To delete the object class mapping, click **Delete**. Click **OK** at the confirmation prompt.
5. Click **OK**.

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


Type a name or select a script from the list.
Select a query to filter the list of scripts.
Modify a query.
Show names of scripts.
Show details of scripts.
Lists scripts based on the selected query.
Change recording options.
Set properties for scripts.
Record GUI
OK
Cancel
Options...
Properties...
Help

Name: Test4

Query: All

Name	Description	Owner	Created By
Test1		Admin	Admin
Test2		Admin	Admin
Test3		Admin	Admin

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

   The listed scripts have already 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 displayed list, which is useful in projects with hundreds of scripts. You create queries in TestManager, and you modify queries in TestManager or Robot.

   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 2-6.)

6. To change the recording options, click **Options**. When finished, click **OK**.

7. If you selected a previously 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 2-23.)

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 2-18.)

   – 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 2-19.)

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 an the type of application and how you plan to play it back. For information, see *Starting an Application* on page 3-1.

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 3, *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 2-21.)

    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.

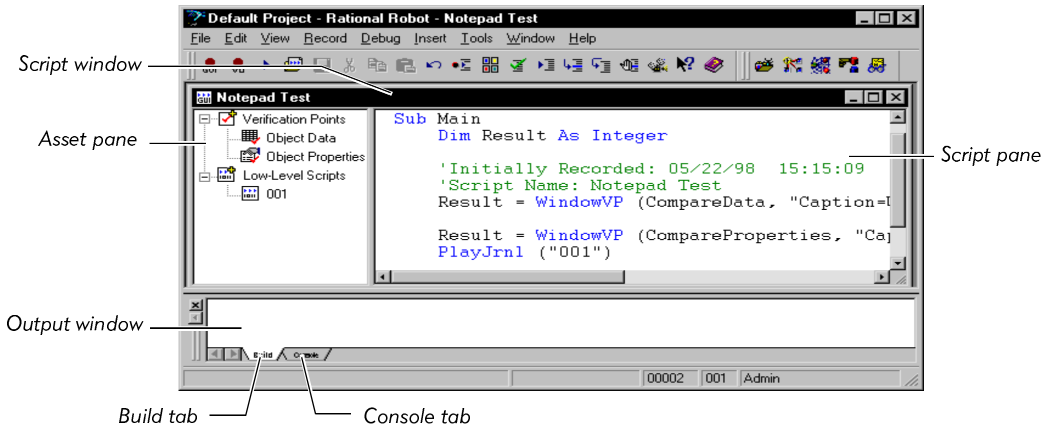
13. When finished, click the **Stop Recording** button on the GUI Record toolbar.

    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.



```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 05/22/98   15:15:09
    'Script Name: Notepad Test
    Result = WindowVP (CompareData, "Caption=

    Result = WindowVP (CompareProperties, "Ca
    PlayJrnl ("001")
```

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 5-7.) 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 2-23.)

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.

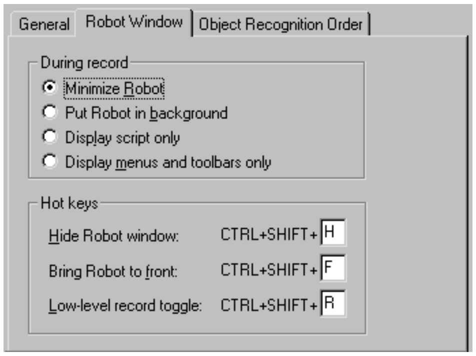
When Robot is minimized or is hidden behind other windows during recording, you can bring it to the foreground in any of the following ways:

- 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 2-5.)
2. Click the **Robot Window** tab.



General | Robot Window | Object Recognition Order

During record
- [✓ Minimize Robot]
- Put Robot in background
- Display script only
- Display menus and toolbars only

Hot keys

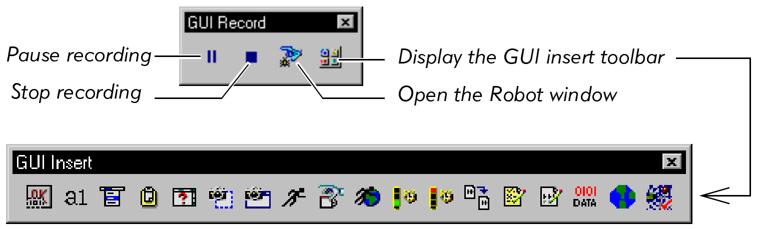
Hide Robot window:	CTRL+SHIFT+ H
Bring Robot to front:	CTRL+SHIFT+ F
Low-level record toggle:	CTRL+SHIFT+ R

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.

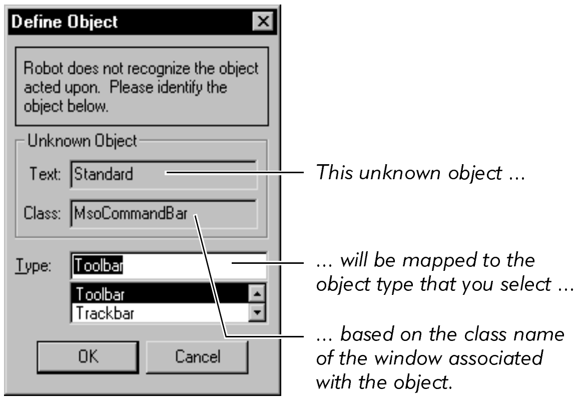
Defining Unknown Objects During Recording

As explained in *Controlling How Robot Responds to Unknown Objects* on page 2-7, 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.



Define Object

Robot does not recognize the object acted upon. Please identify the object below.

Unknown Object

Text: Standard

Class: MsoCommandBar

Type: Toolbar

Toolbar
Trackbar

OK Cancel

This unknown object ...

... will be mapped to the object type that you select ...

... based on the class name of the window associated with the 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 project 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 2-7.)
- 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 project and uses it to identify the custom object during playback. (For more information, see *Mapping Object Types and Classes Before Recording* on page 2-13.)

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:

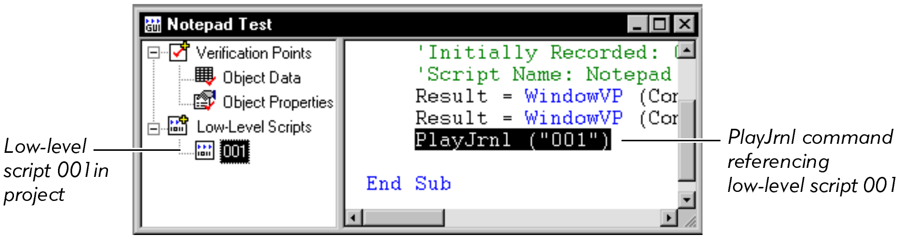
- Press CTRL+SHIFT+R.
- Click the **Open Robot Window** button on the GUI Record toolbar (or press CTRL+SHIFT+F) to bring Robot to the foreground. Click **Record** → **Turn Low-Level Recording On/Off**.

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 project.
- 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**.


Notepad Test
Verification Points
Object Data
Object Properties
Low-Level Scripts
001
Low-level script 001 in project
PlayJrnl command referencing low-level script 001

```
'Initially Recorded:
'Script Name: Notepad
Result = WindowVP (Co
Result = WindowVP (Co
PlayJrnl ("001")

End Sub
```

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 5-3.)

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 is 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

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 define or edit script properties *after* you record the script.

To define script properties:

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 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 Rational TestManager 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*.

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 5, *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 5-9.

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.

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 9-3.
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.

▸ ▸ ▸ C H A P T E R 3

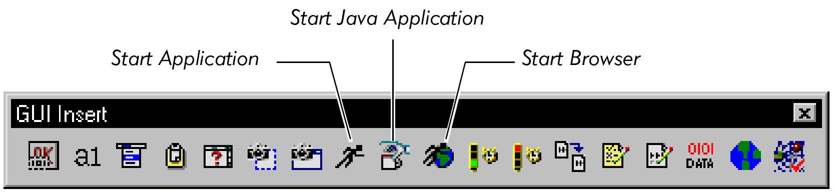
Adding Features to GUI Scripts

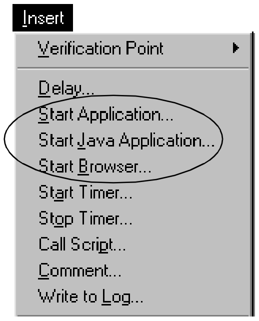
This chapter describes the features that you can add to GUI scripts. It includes the following topics:

- 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, Delphi, 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 2-4.

Starting Applications

The following steps list the basic information you need to know to start an application:

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. You can specify that you want the application to start under Rational Purify, Quantify, or PureCoverage during playback. (For more information, see the next section, *Starting Applications Under the Rational Diagnostic Tools*.)
   – To start a Java application that you want to start under Quantify or PureCoverage during playback, click the **Start Java Application** button. (For more information, see the next section, *Starting Applications Under the Rational Diagnostic Tools*.)
   – To start an HTML application, click the **Start Browser** button. (For more information, see *Enabling HTML Testing in Robot* on page 13-3.)

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

   For information about an 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 on the taskbar) to start an application.

Starting Applications Under the Rational Diagnostic Tools

When you play back a Robot script, you can have any applications specified in the script start under the following Rational diagnostic tools:

Rational Purify – Detects and diagnoses memory access errors and memory leaks. Robot with playback under Purify works with Visual C/C++ applications on Windows NT 4.0 and Windows 2000.

Rational Quantify – Profiles the time spent in each module, function, line, and block of code, and detects performance bottlenecks within an application. Robot with playback under Quantify works with Visual C/C++, Visual Basic, and Java applications on Windows NT 4.0 and Windows 2000.

Rational PureCoverage – 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. Robot with playback under PureCoverage works with Visual C/C++, Visual Basic, and Java applications on Windows NT 4.0 and Windows 2000.

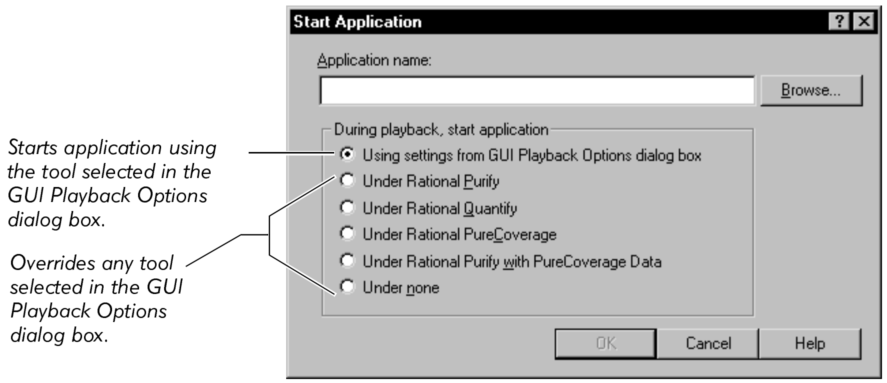
For detailed information about the diagnostic tools and how they work with Robot, see *Setting Diagnostic Tools Options* on page 9-11.

There are two ways to specify the diagnostic tool that an application should start under:

- During recording, in the Start Application or Start Java Application dialog box.
- During playback, in the Diagnostic Tools tab of the GUI Playback Options dialog box.

Specifying the Diagnostic Tool During Recording

During recording, you start an application using the Start Application or Start Java Application dialog box. In the dialog box, you can specify the diagnostic tool that the application should start under during playback.


Starts application using the tool selected in the GUI Playback Options dialog box.
Overrides any tool selected in the GUI Playback Options dialog box.
Browse...
OK
Cancel
Help

Start Application

Application name:

During playback, start application
- [✓ Using settings from GUI Playback Options dialog box]
- Under Rational Purify
- Under Rational Quantify
- Under Rational PureCoverage
- Under Rational Purify with PureCoverage Data
- Under none

It is useful to set the diagnostic tool option during recording if you have several applications in a script and you want to start each application under a different tool during playback.

For example, suppose you have a shell script that calls three scripts. Each script starts one application, and you want to start each one under a different tool. When you started each application during recording, you would select the appropriate tool. When you played back the script, the setting for each application would override the setting in the GUI Playback Options dialog box.

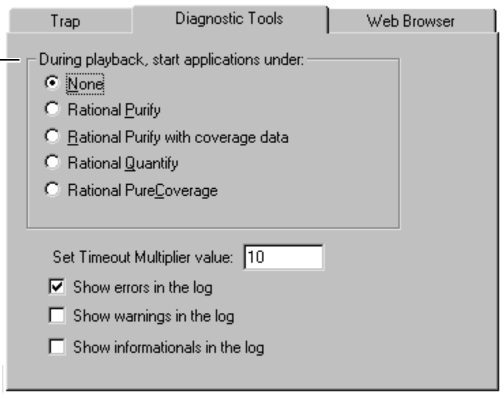
The tools options are enabled in the dialog box if the tools are installed.

When you are ready to play back the script, you need to set some options in the GUI Playback Options dialog box. For information, see *Setting the Diagnostic Tools Options* on page 9-13.

Specifying the Diagnostic Tool During Playback

During playback, you can use the Diagnostic Tools tab of the GUI Playback Options dialog box to specify the diagnostic tool that *all* applications in a script should start under. (For more information, see *Setting Diagnostic Tools Options* on page 9-11.)

Starts application under the selected tool (or None), if ***Us se s f GUI Pl yb k s d l b x*** *was selected when the application was started during recording.*



During playback, start applications under:

- [✓ None]
- Rational Purify
- Rational Purify with coverage data
- Rational Quantify
- Rational PureCoverage

Set Timeout Multiplier value: 10

- [✓] Show errors in the log
- [ ] Show warnings in the log
- [ ] Show informationals in the log

It is useful to set the diagnostic tool option during playback if you want all applications in a script to start under the same tool. This is especially useful if you want to run all the applications under a different tool each time you play back the script.

For example, suppose you have a shell script that calls three scripts. Each script starts one application, and you want to start them all under Purify. When you started each application during recording, you would select **Using settings from GUI Playback Options dialog box**. When you played back the script, you would select **Rational Purify** in the GUI Playback Options dialog box.

The tools options are enabled in the dialog box if the tools are installed.

Robot uses **Set timeout multiplier value** to multiply wait state and delay values during playback. For information, see *Setting the Diagnostic Tools Options* on page 9-13.

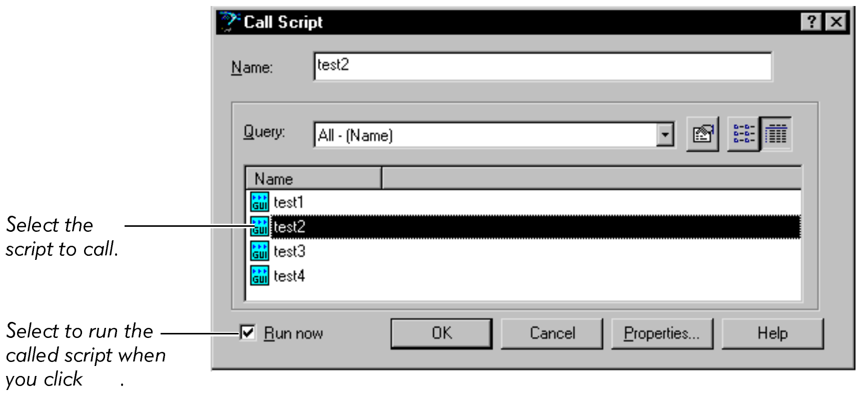
Inserting a Call to Another Script

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.



3. Select a GUI script from the list.
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 2-25.

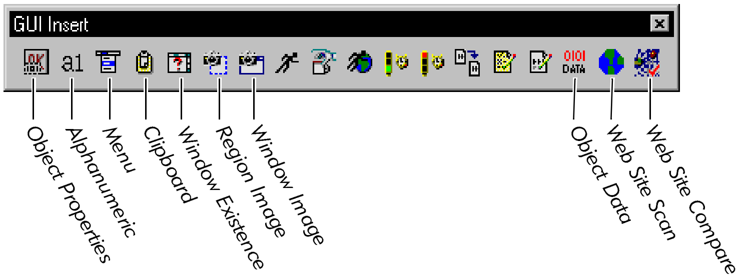
Inserting Verification Points

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 4, *Creating Verification Points in GUI Scripts*.

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

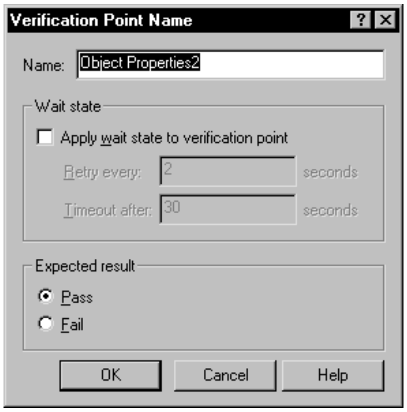
1. Do one of the following:
   - If recording, click the **Display GUI Insert Toolbar** button on the GUI Record toolbar.
   - If editing, position the pointer in the script and click the **Display GUI Insert Toolbar** button on the Standard toolbar.
2. Click a verification point button on the GUI Insert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


Verification Point Name

Name: Object Properties2

Wait state

☐ Apply wait state to verification point

Retry every: 2 seconds

Timeout after: 30 seconds

Expected result

● Pass

○ Fail

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 4-8.)

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 4-9.)

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 3-10.

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 4-8.) 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**.

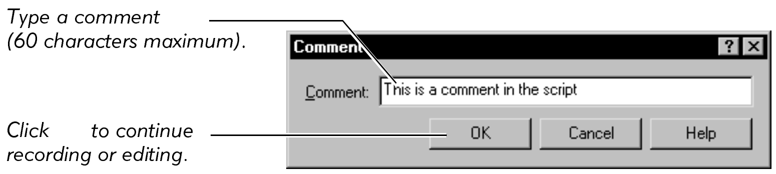
When you play back the script and view the log, the elapsed time is displayed for each Stop Timer event. For more information, see Chapter 9, *Playing Back GUI Scripts*.

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   to continue recording or editing.
Comment
Comment: This is a comment in the script
OK
Cancel
Help

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

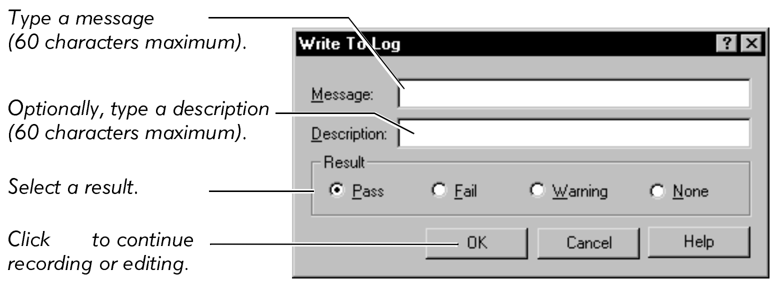
During recording or editing, you can insert a log message, description, and result into a GUI script. During playback, Robot inserts this information into the log. You can use log messages to document your script for the playback process.

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 to continue recording or editing.
Write To Log
Message:
Description:
Result
[✓ Pass] Fail Warning None
OK
Cancel
Help

After playback, you can view logs and messages using TestManager. 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** → **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 9-6.)

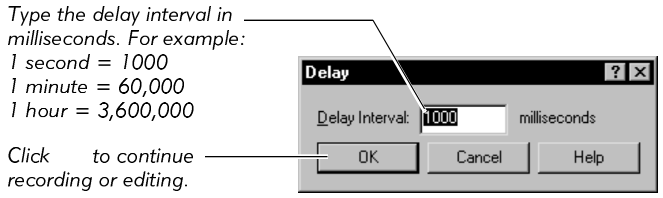
At times during playback, you may need to have Robot pause for a specific amount of time before executing a particular command. For example, an additional delay may be necessary if the application accesses a network server, printer, or other remote system. In these cases, if script playback does not 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 4-8.

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

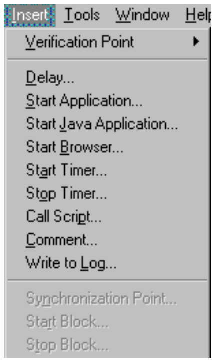
Click to continue recording or editing.

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 project-wide scope, but they do not support verification points. They are stored in the SQABas32 folder of the project.
- **.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 "Mylibrary" (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 5-7.

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 SQABasic header files. They can be accessed by all modules within the project.

SQABasic files are stored in the SQABas32 folder of the project, unless you specify another location. You can specify another location by clicking **Tools** → **General Options**. Click the **Preferences** tab. Use the **Browse** button to find the location. Robot will check this location first. If the file is not there, it will look in the SQABas32 directory.

SQABasic header files have the extension .sbh.

Creating and Editing Header Files

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

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 project-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**.

Adding Declarations to the Global Header File

For your convenience, Robot provides a blank header file called Global.sbh. Global.sbh is a project-wide header file stored in SQABas32 in the project. 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**.

▸▸▸ CHAPTER 4

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

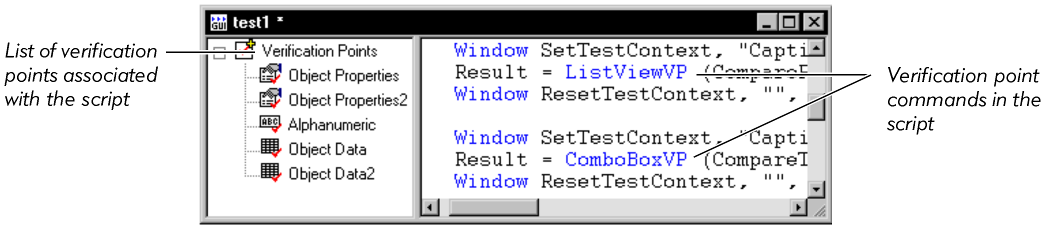
During recording, a verification point captures object information (based on the type of verification point) and stores it in a **baseline data file**. The information in this file becomes the baseline of the expected state of the object during subsequent builds.

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 TestManager log. If a verification point fails (the baseline and actual data do not match), you can select the verification point in the log and click **View** → **Verification Point** 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 project 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.



```
Window SetTestContext, "Capti
Result = ListViewVP (CompareI
Window ResetTestContext, "",

Window SetTestContext, "Capti
Result = ComboBoxVP (CompareI
Window ResetTestContext, "",
```

NOTE: The following verification points are not stored in the project 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.

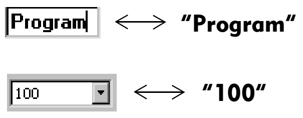
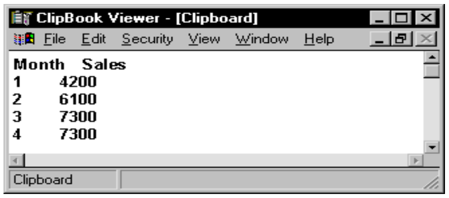
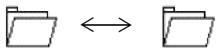
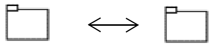
You can easily copy verification points to other scripts if you want to reuse them. For information, see *Copying a Verification Point* on page 4-25.

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 3-14.

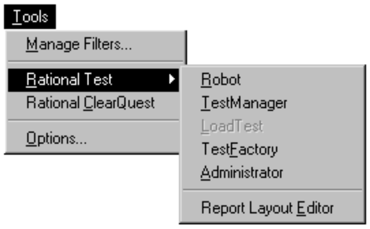
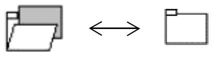
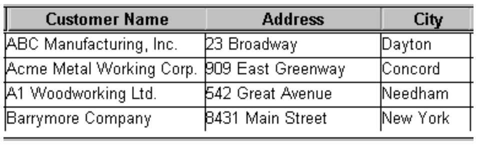
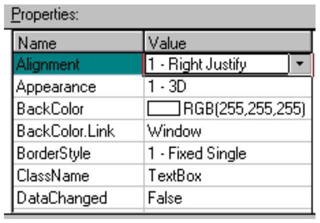
Types of Verification Points

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
Alphanumeric 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.	
Clipboard 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.	 ClipBook Viewer - [Clipboard] File Edit Security View Window Help ClipboardMonth Sales 1 4200 2 6100 3 7300 4 7300
File Comparison 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.	
File Existence 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.	

(Continued)

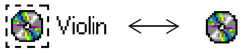
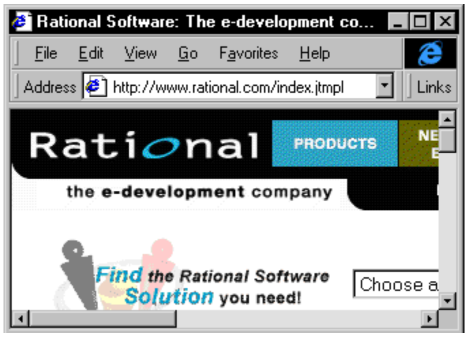
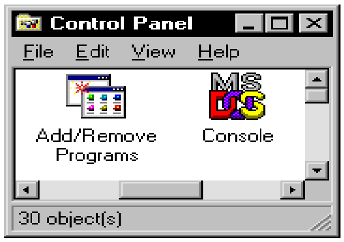
Verification point type	Example
Menu Captures and compares the menu title, menu items, shortcut keys, and the state of selected menus. Robot records information about the top menu and up to five levels of sub-menus. Robot treats menu items as objects within a menu and tests their content, state, and accelerator keys regardless of the menu item's location. (You can also use the Object Data verification point to test a menu.)	 Tools Manage Filters... Rational Test Rational ClearQuest Options... Robot TestManager LoadTest TestFactory Administrator Report Layout Editor
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 A, *Working with Data Tests*.	
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.	

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

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
Region Image Captures a region of the screen as a bitmap. The captured region is a pixel-by-pixel representation that includes colors, height, and width.	
Web Site Compare Captures a baseline of a Web site and compares it to the Web site at another point in time. **Web Site Scan** Checks the contents of a Web site with every revision and ensures that changes have not resulted in defects.	 Rational Software: The e-development co... File Edit View Go Favorites Help Address http://www.rational.com/index.jtmpl Links Rational PRODUCTS the e-development company Find the Rational Software Solution you need! Choose a
Window Existence Verifies the existence and status of a specified window during playback. The status can be normal, minimized, maximized, or hidden.	Control Panel
Window Image Captures a window as a bitmap. The captured window is a pixel-by-pixel representation that includes colors, height, and width.	 Control Panel File Edit View Help Add/Remove Programs Console 30 object(s)

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 logging 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.

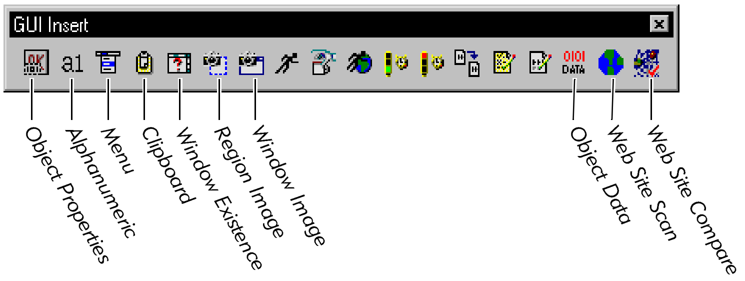
Task	**See**
1. Start to create a verification point.	The next section, *Starting to Create a Verification Point*
2. Set a wait state.	*Setting a Wait State for a Verification Point* on page 4-8
3. Set the expected result.	*Setting the Expected Result for a Verification Point* on page 4-9
4. Select and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

Task	See
6. Select an identification method.	*Selecting an Identification Method* on page 4-15
7. Select the data or properties to test.	*Selecting the Data to Test in a Data Grid* on page 4-19
8. Test column titles or menus (optional).	*Testing Column Titles or Top Menus in a Data Grid* on page 4-20
9. Edit the captured data (optional).	*Editing Captured Data in a Data Grid* on page 4-21

Starting to Create a Verification Point

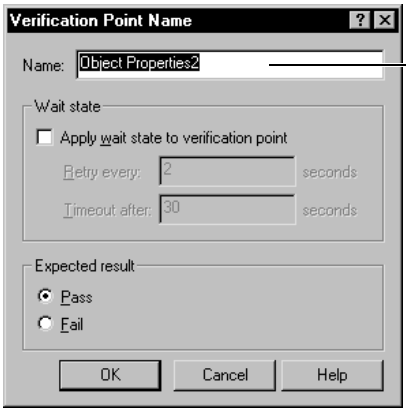
The following is the basic procedure for starting to create a verification point:

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 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 4-9.
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 4-3. 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 4-7.)
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 4-7.)
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 4-8.)

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

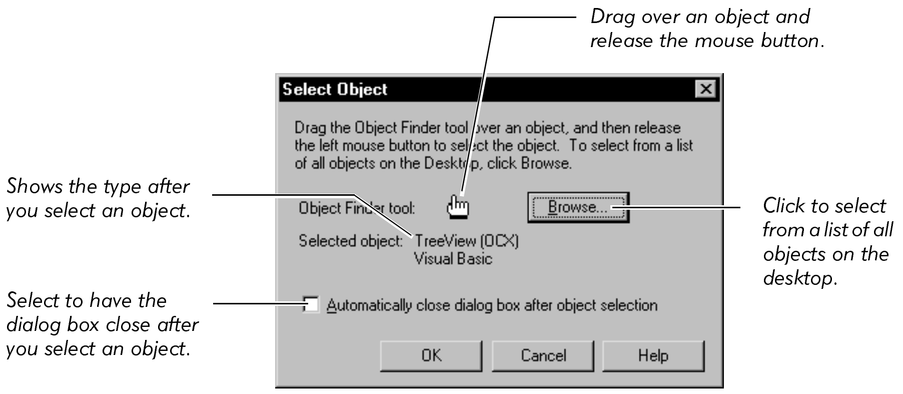
There are two ways to select the object to test:

- 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 4-7.)

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


Drag over an object and release the mouse button.
Shows the type after you select an object.
Select to have the dialog box close after you select an object.
Click to select from a list of all objects on the desktop.
Select Object
Object Finder tool:
Browse...
OK
Cancel
Help

Drag the Object Finder tool over an object, and then release the left mouse button to select the object. To select from a list of all objects on the Desktop, click Browse.

Selected object: TreeView (OCX) Visual Basic

☐ Automatically close dialog box after object selection

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.

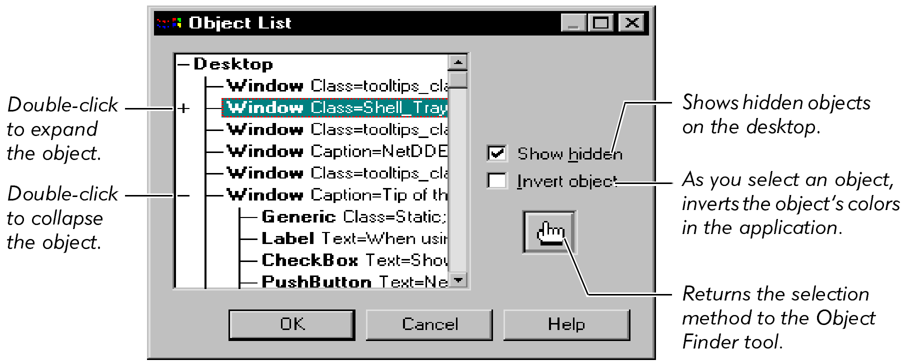
   To select a visible object directly from the application, continue with step 4. To select an object from a list of all objects on the desktop, skip to step 5.

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**.


Object List
Desktop
Window Class=tooltips_cla
Window Class=Shell_Tray
Window Class=tooltips_cla
Window Caption=NetDDE
Window Class=tooltips_cla
Window Caption=Tip of th
Generic Class=Static;
Label Text=When usir
CheckBox Text=Shou
PushButton Text=Ne
Show hidden
Invert object
OK
Cancel
Help
Double-click to expand the object.
Double-click to collapse the object.
Shows hidden objects on the desktop.
As you select an object, inverts the object's colors in the application.
Returns the selection method to the Object Finder tool.

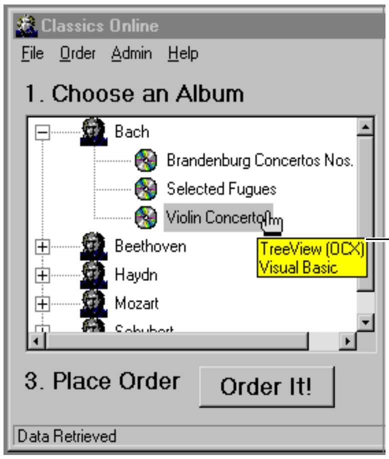
The Object List dialog box includes hidden objects that you cannot point to because they are not visible through the user interface, such as objects with the Visible property set to False and objects with no GUI component. This dialog box also includes objects that are direct children of the desktop,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

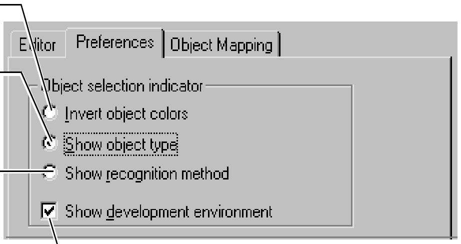
When you point to an object in the application-under-test with the Object Finder tool, Robot displays a TestTip that identifies the object.



TestTip identifies the object type and development environment.

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.



Object selection indicator: Invert object colors ( ) / [✓ Show object type] / Show recognition method ( ); Show development environment [✓]

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*).*

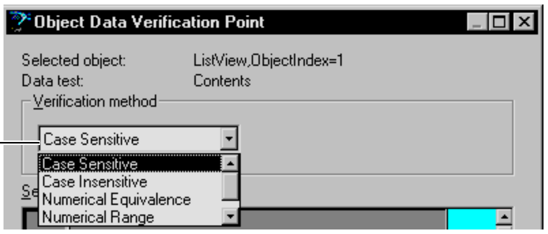
Displays a TestTip that describes both the object type and the object recognition method as you point to an object.

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.



Select a verification method to specify how Robot compares the data.

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.

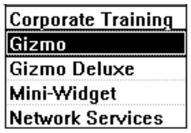
If the data is displayed in a two-dimensional grid, you select two identification methods — one for columns and one for rows. If the data is displayed in a one-dimensional grid, you select only one identification method.

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 4-18.)

By Content

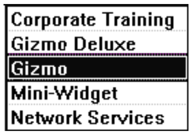
Use this method to verify that the recorded values exist during playback. This method is location-independent. For example, if you record a value of 100, the verification point passes as long as the value 100 exists during playback.

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



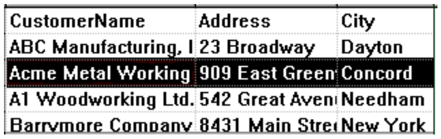
Corporate Training
Gizmo Deluxe
Gizmo
Mini-Widget
Network Services

Playback (Pass)

By Location

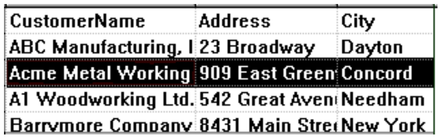
Use this method to verify that the recorded values exist in the same locations during playback. For example, when you test items in a menu, use By Location to verify that the locations of the recorded menu items remain the same during playback. You can also use By Location to verify that the locations 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 Company	8431 Main Stre	New York

Baseline



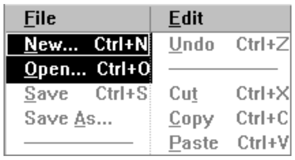
CustomerName	Address	City
ABC Manufacturing, I	23 Broadway	Dayton
Acme Metal Working	909 East Green	Concord
A1 Woodworking Ltd.	542 Great Aven	Needham
Barrymore Company	8431 Main Stre	New York

Playback (Pass)

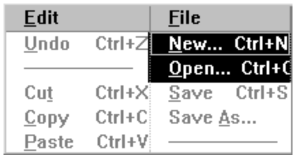
By Title

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.



Baseline



Playback (Pass)

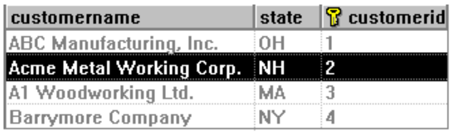
By Key/Value

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.

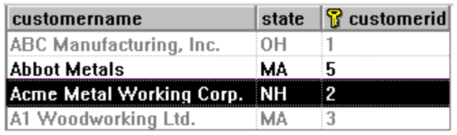
To add or remove a key from a column, position the pointer anywhere in the column and click the right mouse button.

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 the Grid Comparator Help.

List of Identification Methods

The following tables 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.

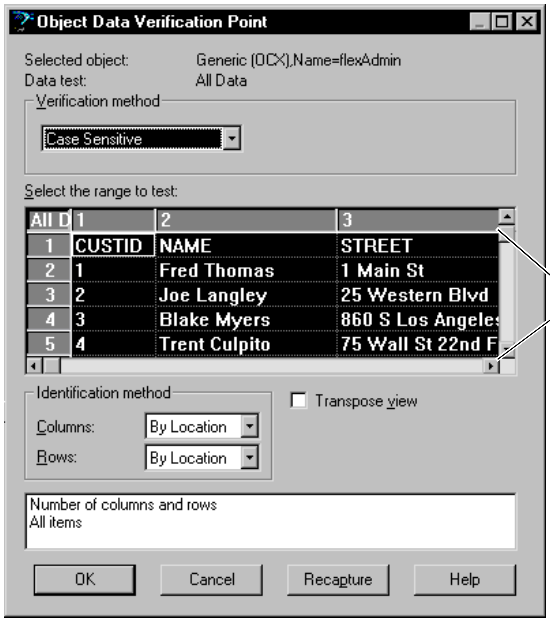
Working with the Data in Data Grids

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.



Selected object: Generic (OCX),Name=flexAdmin
Data test: All Data
Case Sensitive
Select the range to test:

All D	1	2	3
1	CUSTID	NAME	STREET
2	1	Fred Thomas	1 Main St
3	2	Joe Langley	25 Western Blvd
4	3	Blake Myers	860 S Los Angele
5	4	Trent Culpito	75 Wall St 22nd F

Columns: By Location
Rows: By Location

Number of columns and rows
All items

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

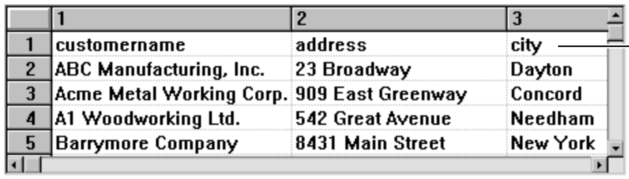
Use any of the following methods to select data in the columns, rows, or cells of the data grid. The selected values become the baseline that Robot uses during playback to test the current build of the application.

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:

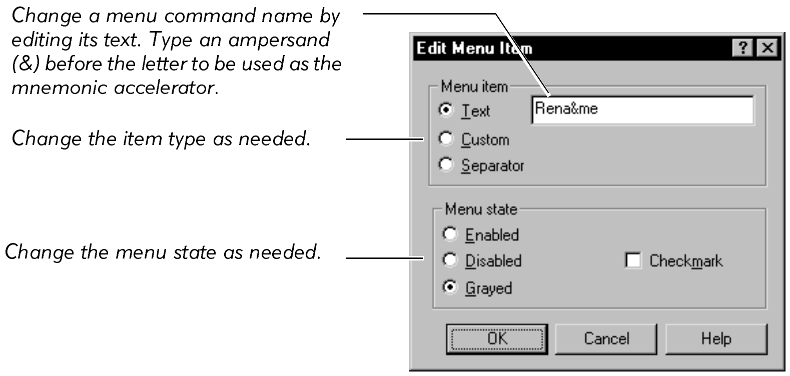
1. Double-click a cell in the data grid. The pointer changes to a text cursor.
2. Edit the data in the cell.
3. To accept the changes, press ENTER. To cancel the changes, press ESC.

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.
Edit Menu Item
Menu item
Text
Rena&me
Custom
Separator
Menu state
Enabled
Disabled
Grayed
Checkmark
OK
Cancel
Help

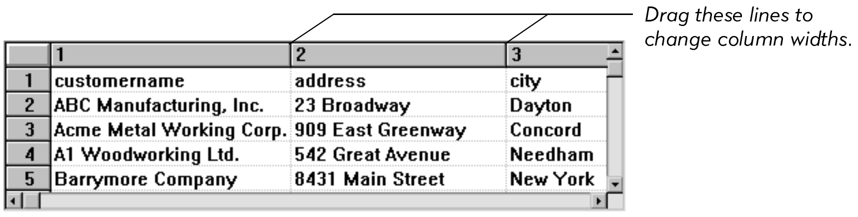
Restrictions on Editing Data

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

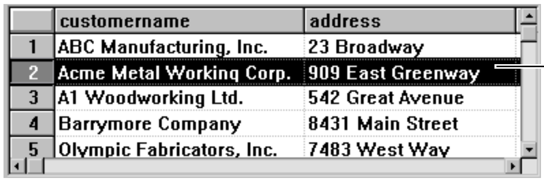
The column widths in the data grid default to fit the longest data string. You can adjust the widths of any of the columns in the grid by dragging the lines between the columns.



	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

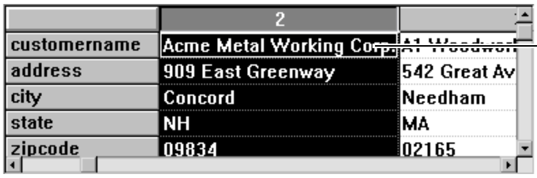
Transposing Columns and Rows in a Data Grid

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 Woodwor
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 project, 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.

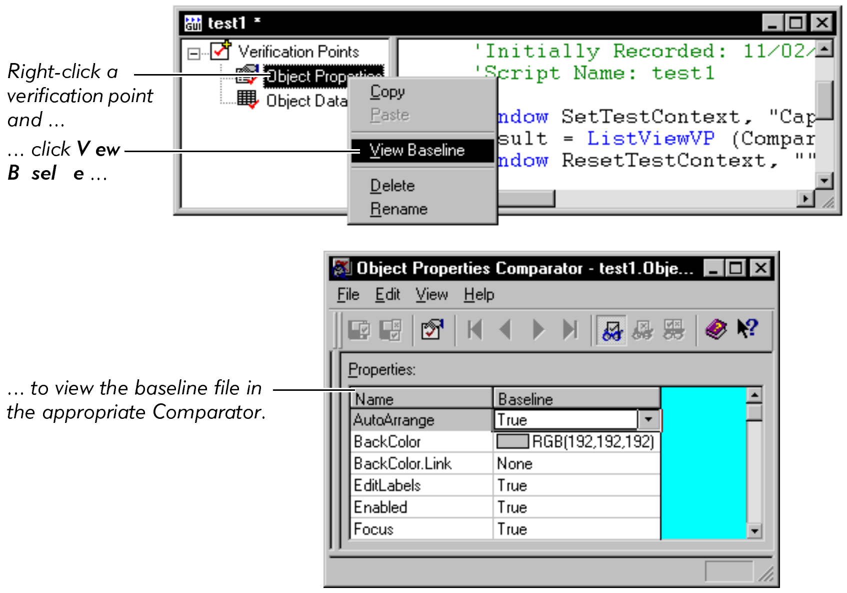
NOTE: The following verification points are not stored in the project and do not appear in the Asset pane in the Script window: File Comparison, File Existence, Module Existence, Window Existence, and Alphanumeric (if the verification method is Numeric Equivalence or Numeric Range). You can rename, copy,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


Right-click a verification point and ...
... click *V* ew *B* sel e ...
... to view the baseline file in the appropriate Comparator.
test1 *
Verification Points
Object Properties
Object Data
Copy
Paste
View Baseline
Delete
Rename
Object Properties Comparator - test1.Obje...
File Edit View Help
Properties:

```
'Initially Recorded: 11/02/
'Script Name: test1

ndow SetTestContext, "Cap
sult = ListViewVP (Compar
ndow ResetTestContext, ""
```

Name	Baseline
AutoArrange	True
BackColor	RGB(192,192,192)
BackColor.Link	None
EditLabels	True
Enabled	True
Focus	True

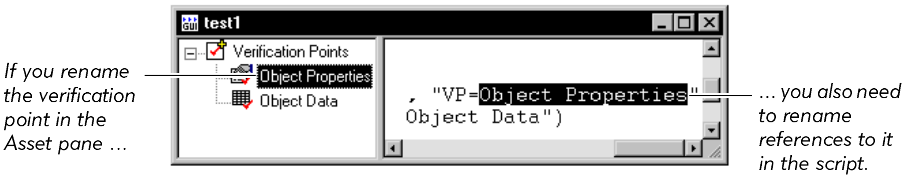
Once the baseline file opens in the appropriate Comparator, you can view and edit the data. Editing data is useful if you want to change the baseline data for a verification point based on a new specification or anticipated changes to the application-under-test. By editing data before playback, you can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

NOTE: To compare the baseline and actual files, you must open the Comparator through the log. For information, see *Analyzing Verification Point Results with the Comparators* on page 9-21.

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 project.
- Renaming all references to that verification point in the script.



If you rename the verification point in the Asset pane …
test1
Verification Points
Object Properties
Object Data
… you also need to rename references to it in the script.

```
, "VP=Object Properties",
Object Data")
```

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 project), 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 project.
- 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 project.

   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 project.
- 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 5

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 3, *Adding Features to GUI Scripts*.

Working with Low-Level Scripts

As indicated in *Switching to Low-Level Recording* on page 2-21, 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 project 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.

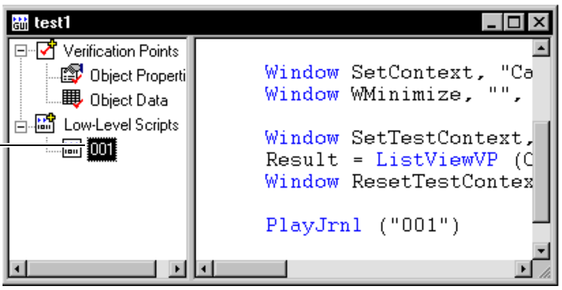
Viewing Low-Level Scripts

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.



```
Window SetContext, "Ca
Window WMinimize, "",

Window SetTestContext,
Result = ListViewVP (C
Window ResetTestContex

PlayJrnl ("001")
```

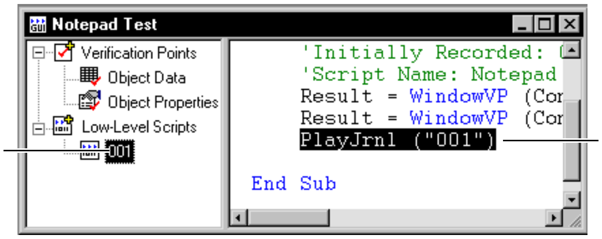
The low-level ASCII file lists the actions that occurred during low-level recording. For information about the contents of this file, see *low-level recording* in the Robot Help Index.

Renaming a Low-Level Script

When you record a low-level script, it is stored in the project. 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 project.
- Renaming all references to that low-level script in the script.

If you rename the low-level script in the Asset pane …



```
'Initially Recorded: (
'Script Name: Notepad
Result = WindowVP (Cor
Result = WindowVP (Cor
PlayJrnl ("001")

End Sub
```

… 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 project), 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 project.
- 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 project.

   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 `PlayJrnl`. 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 project.
- 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 `PlayJrnl`.
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 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

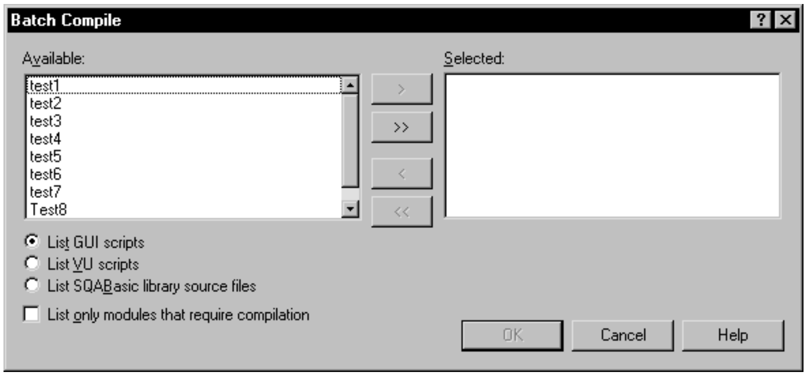
You can compile the active script or file, or you can compile all scripts and files in the current project.

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:

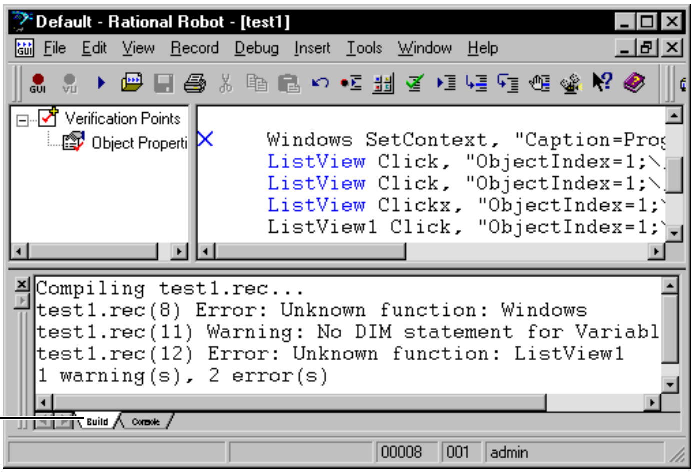
1. Click **File** → **Batch Compile**.



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.


Default - Rational Robot - [test1]
File Edit View Record Debug Insert Tools Window Help
Verification Points
Object Properti
Build
Console
00008
001
admin

```
Windows SetContext, "Caption=Prog
ListView Click, "ObjectIndex=1;\
ListView Click, "ObjectIndex=1;\
ListView Clickx, "ObjectIndex=1;
ListView1 Click, "ObjectIndex=1;
```

```
Compiling test1.rec...
test1.rec(8) Error: Unknown function: Windows
test1.rec(11) Warning: No DIM statement for Variabl
test1.rec(12) Error: Unknown function: ListView1
1 warning(s), 2 error(s)
```

B ld tab shows compilation results.

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
Go	Plays back the currently open script. Executes until either the next breakpoint or the end of the script, whichever comes first.
Go Until Cursor	Plays back the currently open script, stopping at the text cursor position. Executes until either the next breakpoint or the end of the script, whichever comes first.
Animate	Plays back the currently open script, displaying a yellow arrow in the left margin of each line (or highlighting the line) as it executes. Executes until either the next breakpoint or the end of the script, whichever comes first.
Pause	Pauses playback. To resume playback, click **Debug** → **Pause**.
Stop	Stops playback.
Set or Clear Breakpoint	Sets or clears a breakpoint at the cursor position. 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.

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 `Main`.) 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.

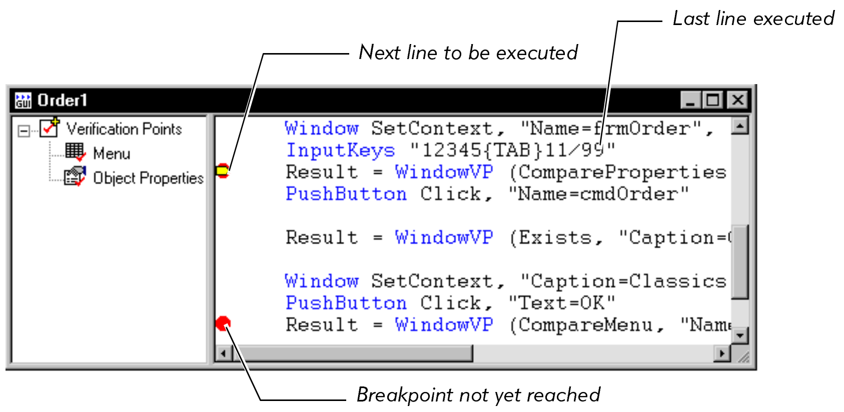
To set and clear breakpoints:

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.



```
Window SetContext, "Name=frmOrder",
InputKeys "12345{TAB}11/99"
Result = WindowVP (CompareProperties
PushButton Click, "Name=cmdOrder"

Result = WindowVP (Exists, "Caption=

Window SetContext, "Caption=Classics
PushButton Click, "Text=OK"
Result = WindowVP (CompareMenu, "Name
```

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**.

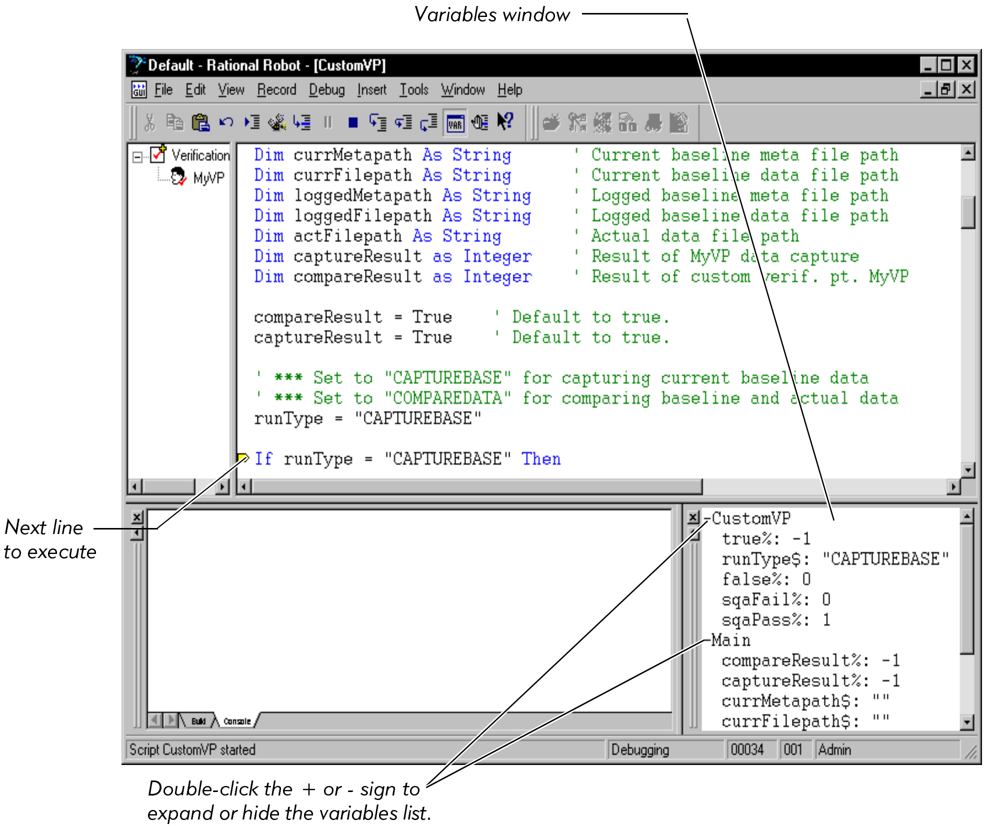
   As Robot plays back the script, it displays a yellow arrow in the left margin of the currently executing line or highlights the line.

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.


Variables window
Default - Rational Robot - [CustomVP]
File Edit View Record Debug Insert Tools Window Help
Verification
MyVP
Next line to execute
Build
Console
Script CustomVP started
Debugging
00034
001
Admin
Double-click the + or - sign to expand or hide the variables list.

```
Dim currMetapath As String      ' Current baseline meta file path
Dim currFilepath As String      ' Current baseline data file path
Dim loggedMetapath As String    ' Logged baseline meta file path
Dim loggedFilepath As String    ' Logged baseline data file path
Dim actFilepath As String       ' Actual data file path
Dim captureResult as Integer    ' Result of MyVP data capture
Dim compareResult as Integer    ' Result of custom verif. pt. MyVP

compareResult = True    ' Default to true.
captureResult = True    ' Default to true.

' *** Set to "CAPTUREBASE" for capturing current baseline data
' *** Set to "COMPAREDATA" for comparing baseline and actual data
runType = "CAPTUREBASE"

If runType = "CAPTUREBASE" Then
```

```
-CustomVP
  true%: -1
  runType$: "CAPTUREBASE"
  false%: 0
  sqaFail%: 0
  sqaPass%: 1
-Main
  compareResult%: -1
  captureResult%: -1
  currMetapath$: ""
  currFilepath$: ""
```

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 CustomVP are module-level variables that are visible to all the sub procedures in the script CustomVP.

Variables and constants that are visible to all modules are listed under the heading *Globals*.

Deleting Scripts

To delete scripts from the project:

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 project 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).

▸ ▸ ▸ Part III

Developing VU Scripts

‣ ‣ ‣ C H A P T E R 6

Setting Recording Options

This chapter describes how to set recording options, manage proxies, and provide login information through the Authentication Datapool. It includes the following topics:

- About session recording
- Setting the recording method
- Setting script generation options
- Setting general recording options
- Defining a client or server computer
- Removing a computer or port
- Authenticating login
- Managing proxies
- About Session Recording

The following steps outline the general process for recording a script:

1. Set the session recording options.

   Recording options tell Robot how to record and generate scripts. You set recording options to specify:

   – The type of recording you want to perform, such as API, network, or proxy. The recording method you choose determines some of the other recording options you need to set.

   – Script generation options, such as specifying whether you want the script to include datapool commands or think time delays, and whether you want to filter out protocols to control the size of the script.

   – General recording options, such as the prefixes to assign to default script and session names.

2. Start the recording session.

   With the API recording method, you must start recording first, at which point Robot prompts you for the name of the client. With the other recording methods, network and proxy, you can start recording before or after you start the client.

3. Start the client application.
4. Record the transactions. While you are recording the transactions, you can split the session into multiple scripts, each representing a logical unit of work.
5. Optionally, insert timers, blocks, comments, and other features into the script during recording.
6. Close the client application.
7. Stop recording
8. Robot automatically generates scripts.

Setting the Recording Method

A **recording method** is the type of performance recording that you want to perform. Your choices are: [double-check table based on new default recording method]

- API recording (the default)
- Network recording
- Proxy recording

In general, we recommend that you select API recording. However, the following table lists certain situations in which a recording method is required or recommended:

Situation	**API**	**Network**	**Proxy**
The client application accesses secure data from a Web server.	Required		
The client application data from a Web server.	Recommended	First alternate	Second alternate
You are testing in a DCOM environment.	Required		
The client application accesses an Oracle 8 database.	Required		

Situation	API	Network	Proxy
The client application accesses an Oracle 7 database. (For network and proxy recording, supply the name of the Oracle database. For more information, see *Providing the Name of an Oracle Database* on page 6-23.)	Recommended	First alternate	Second alternate
The client application is not installed on the Local computer.		Recommended	Alternate
The client application is not running on Windows NT 4 or Windows 2000.		Recommended	Alternate
You want to record traffic from multiple client applications that reside on different computers.		Recommended	Alternate
You want to record traffic between multiple, *specific* client and server computers.			Recommended
Neither the client nor the server computer is on the same network segment as the Local computer.			Required
An Ethernet switch controls network traffic, and neither the client nor the server application are installed on the Local computer.			Required
The client application accesses a TUXEDO server.	Recommended	Alternate	

API Recording

With API recording, Robot records API calls between the specific client application and the server. Recording occurs on the client rather than on the wire, as with network and proxy recording. Therefore, choose API recording if you are accessing secure data from a Web server, because API recording captures the information before it reaches the wire and is encrypted.

API recording is supported only on Windows NT clients. With API recording, you do not have to specify the network names or IP addresses of the client and server as you do with network and proxy recording.

How to Choose API Recording

To use the API recording method:

1. Click **Tools** → **Session Record Options**.
2. Click the **Method** tab, and click **API recording**.

Network Recording

With network recording, Robot records packet-level traffic at the OSI network interface layer using the “promiscuous” mode of your network card. Network recording is media-independent, supporting standards such as Ethernet, Token Ring, and Fiber Distributed Data Interface (FDDI).

Network recording occurs on the wire rather than on the client (as with API recording).

How to Choose Network Recording

To use the network recording method:

1. Click **Tools** → **Session Record Options**.
2. Click the **Method** tab, and click **Network recording**.
3. Optionally, click the **Method:Network** tab, and select the client/server pair that you will record. The default is to record all of the network traffic to and from your computer. (For information, see *Selecting a Client/Server Pair* below.)
4. Optionally, click the **Generator Filtering** tab to specify the network protocols to include in the script that Robot generates. (For information, see *Setting Filtering Options* on page 6-16.)

Selecting a Client/Server Pair

The **Method:Network** tab contains the following lists of computer addresses. Select one item in each list:

Client – The client’s network name (or IP address) and, optionally, the port number.

Server – The server’s network name (or IP address) and, optionally, the port number.

If a computer that you want to specify is not listed, define the computer as described in *Defining a Client or Server Computer* on page 6-28.

You can choose **Any** or **Local machine** instead of a specific computer name:

- If you select **Any** for either the client or the server, Robot records traffic for all clients or all servers on the network.

- If you select **Local machine** for the client (the default), Robot records traffic from the Local computer. Robot determines the computer's network name automatically. You do not have to specify it.

  Local machine records traffic from all of the computer's ports. To record traffic from a particular port, click **Manage Computers** to define the computer network name and port number of interest.

Selecting a Network Card

If you are using network recording and the Local computer has more than one network interface card installed, you must identify the card to use, as follows:

1. Click **Start** → **Settings** → **Control Panel**.
2. Double-click the **Network** icon.
3. Click the **Bindings** tab.
4. Select **all services** in the **Show Bindings for** box.
5. In the list of services, expand the **Rational Test Network Driver** item by clicking the + icon before it.
6. Click the network interface card that you want to use.
7. Click **Move Up** until the selected card is at the top of the list.
8. Click **OK**.
9. Reboot the computer.

Alternatively, instead of moving the name of the interface card that you want to use to the top of the list, you can click **Disable** to disable all of the other interface cards.

Proxy Recording

With proxy recording, the client/server traffic is routed through a proxy computer.

Proxy recording occurs at the OSI application layer and involves receiving and sending socket transactions. With proxy recording, you can record conversations between multiple, *specific* clients and servers (that is, when the **Any** choice in the **Method:Network** tab for either clients or servers would be impractical).

The following examples show when you might need multiple clients and servers:

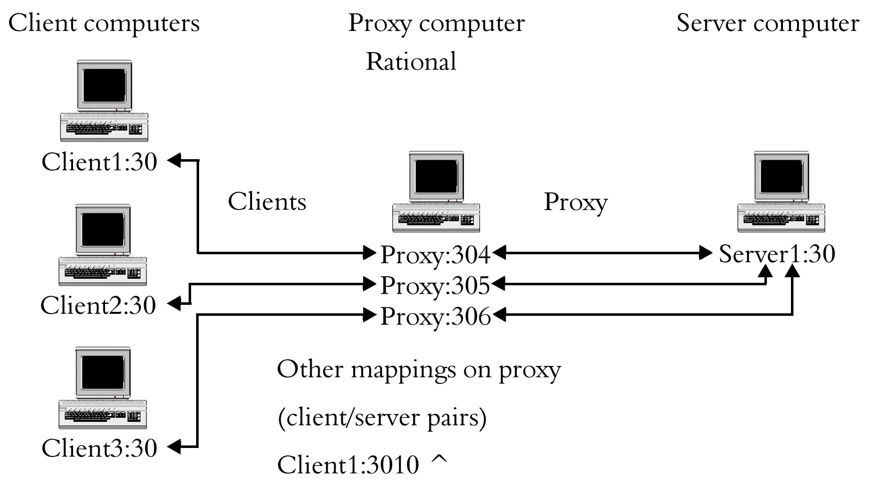
- You need multiple client computers if different user groups (such as order entry clerks and customer service representatives) will issue requests to the server in at the same time during a single recording session.

- You need multiple servers if requests are being sent to different databases (such as an Inventory database and a Human Resources database) located on different computers.

The **proxy computer** intercepts requests from clients and relays them to the server. None of the client computers issuing requests to the servers need to have the Robot software installed. Robot is required only on the proxy computer.

Note that this document uses the word "proxy" to refer to the computer that performs proxy recording. It does not refer to a Web proxy server.

The following figure illustrates a proxy recording setup with multiple client computers and one server. Each computer's network name indicates its role in the client/server traffic. Network names are followed by the computer's port number:



NOTE: The proxy can run on one of the client computers. To have one computer serve as both the proxy and a client, assign different port numbers to the proxy and client.

How to Choose Proxy Recording

To use the proxy recording method:

1. Click **Tools** → **Session Record Options**.
2. Click the **Method** tab, and click **Proxy recording**.

3. Click the **Method:Proxy** tab to:
   - Create a proxy computer.
   - Identify client/server pairs that will communicate through the proxy.

After you set up your system for proxy recording, you should record a trial script to make sure the proxy recording yields the results you expect.

NOTE: If you are proxy recording against an Oracle database, the server should not be set up to redirect. Consult your Oracle documentation for information.

Proxy Recording Overview

To set up and use proxy recording:

1. Start Robot on the proxy computer.
2. In the Proxy Administration dialog box, match up the proxy computer and port with each server to be used in the test.

   For details, see *Creating a Proxy Computer* on page 6-8.
3. In the **Method:Proxy** tab of the Session Record Options dialog box, match up each client with the server it will send requests to. Be sure to specify the actual server and not the proxy computer.

   For details, see *Identifying Client/Server Pairs* on page 6-8.
4. Configure each client to send requests to the proxy computer, not to the server. For example, if the client will be sending requests to an Oracle database, use the Oracle client configuration software to specify the proxy computer's address and port number, not the server's.
5. On each client computer, a tester should start the client application and navigate to the point where recording will begin.
6. On the proxy computer, enable recording (**File** → **Record Session**).
7. With recording enabled, each tester at each client computer performs the transactions to record.
8. When all transactions are complete, stop recording on the proxy computer.

Creating a Proxy Computer

You create a proxy computer by mapping the proxy computer's address to the address of one or more servers.

Before you create a proxy computer, be sure that:

- The server's network name (or IP address) and port number are defined. If they are not defined, click **Manage Computers** to display the Manage Computers dialog box. For information about how to define the server's network name and port number in this dialog box, see *Defining a Client or Server Computer* on page 6-28.
- Proxy service is running. For more information, see *Starting and Stopping Proxy Service* on page 6-33.

To create a proxy computer:

1. Click **Tools** → **Session Record Options**.
2. Click the **Method** tab and make sure that **Proxy recording** is selected.
3. Click the **Method:Proxy** tab.
4. Click **Proxy Admin**.
5. In **Proxy:Port**, specify the proxy computer's port number. Note that Robot has already detected and specified the proxy computer's name.

   You can specify any available port number. Avoid the "well-known" ports (those below 1024). If you specify a port number that is unavailable, Robot prompts you for a new port number.
6. In the **Server:Port** list, select a server involved in the test.
7. Click **Create Proxy**.

The proxy computer is added to the **Existing Proxies** list.

Identifying Client/Server Pairs

Clients and servers communicate through the proxy. A client and server that communicates through a proxy is called a **client/server pair**.

Before you identify a client/server pair, be sure that the network names or IP addresses of all the clients and servers in the test appear in the **Method:Proxy** tab. If any are not listed, click **Manage Computers** to define them. For information about how to define a client or server computer, see *Defining a Client or Server Computer* on page 6-28.

To associate each client in the test with the server it will communicate with:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method** tab and make sure that **Proxy recording** is selected.
3. Click the **Method:Proxy** tab.
4. Select a client in the **Client [:Port]** list.

   The client port is optional.
5. Select the client's server in the **Server:Port** list.

   The server port is required.
6. Click **Add**.

The client/server pair that you have identified appears in the **Client/Server pairs for recording** list.

Setting Script Generation Options

Although you should set script generation options before you record a session, you can also change these options after you record a session. After recording, you can regenerate the script with new options, without recording the session again. The script generation options enable you to:

- Modify the contents of the script—for example, by specifying whether the script will include datapool commands or think time delays
- Set filtering options to control the size of the script—for example, by selecting certain protocols to be included, and excluding the other protocols.
- Modify a script that contains specific protocol requests—for example, controlling settings for HTTP, Oracle, Tuxedo, or IIOP.

Modifying the Contents of a Script

To modify the contents of a script:

1. Click **Tools** → **Session Record Options.**
2. Click the **Generator** tab.

This tab lets you specify whether the script will:

- Use datapools.
- Contain prefixes for command IDs.
- Contain the data retrieved from the server (as comments).
- Contain the number of rows or bytes affected by SQL statements or Web requests.
- Contain the return codes for SQL statements.
- Bind output parameters to VU variables
- Include think time delays.
- Include both think time and CPU delays.
- Have a maximum think time delay.

The following sections describe these options.

Use Datapools

Select this check box if you want Robot to generate datapool commands in the script. Datapool commands allow the script to access data in a datapool.

A **datapool** allows each virtual tester running a script to send realistic data to the server. If you do not use a datapool, each virtual tester sends the same data to the server (the data that you sent to the server when you recorded the script).

Command ID Prefix

This option specifies an optional prefix for emulation command IDs.

Emulation commands include commands for emulating database clients as well as commands for performing communication and timing functions. An *emulation command ID* appears in brackets after the command name. It uniquely identifies the emulation command in TestManager reports.

Emulation command IDs consist of a prefix and a three-digit numeric suffix. For example, if you specify `task` as the command ID prefix, you might see the following emulation command name and ID:

```
sqlnrecv ["task001"] 1355;
```

Robot automatically increments the numeric suffix by a value of 1 with each emulation command.

The maximum length of the command ID prefix is seven characters. If you do not specify a prefix, Robot uses the script name as the prefix (up to seven characters).

You cannot use this option to define multiple command ID prefixes in a recording session. If you want your script to contain multiple command ID prefixes, use blocks to define these prefixes.

With TUXEDO commands, any prefix that you define in **Command ID prefix** is ignored if you specify a predefined TUXEDO emulation command ID prefix. For more information, see *Assigning a Prefix to TUXEDO Command IDs* on page 6-23.

Truncating a Command ID Prefix

Robot uses a tilde (~) to indicate that a command ID prefix that exceeds seven characters has been truncated.

The command ID prefix format is slightly different for truncated prefixes that appear in single-script sessions and multi-script sessions. For example, if you define a command ID prefix of `EmulCmdID`, Robot truncates the prefix as follows:

- Truncation format for single-script sessions:

  ```
  sqlexec ["EmulCmd~024"] "select * from table";
  ```

  ```
  sqlnrecv ["EmulCmd~025"] ALL-ROWS;
  ```

- Truncation format for multi-script sessions:

  ```
  sqlexec ["EmulCmd~3.024"] "select * from table";
  ```

  ```
  sqlnrecv ["EmulCmd~3.025"] ALL-ROWS;
  ```

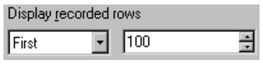
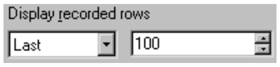
  The 3 after the tilde shows that the command is in the third script in the session.

Display Recorded Rows

This option specifies whether you want some or all of the data retrieved from the server to be inserted into the script.

The inserted data is for informational purposes only. Data is inserted in the form of comments and does not affect playback.

Select one of the following values from the **Display recorded rows** list:

Value	Meaning
All	Insert all retrieved data into the script.
First	Insert a specified number of bytes (HTTP and socket protocols) or rows (other protocols) from the beginning of the data retrieved from the server. If you select **First**, enter the number of bytes or rows to retrieve in the box that appears to the right of the **Display recorded rows** list:  Display recorded rows First 100
Last	Insert a specified number of rows from the end of the set of records retrieved from the server. You cannot use **Last** with HTTP or socket protocols. If you select **Last**, enter the number of rows to retrieve in the box that appears to the right of the **Display recorded rows** list:  Display recorded rows Last 100
None	Do not insert any retrieved data into the script.

Display recorded rows is supported only for Sybase, Microsoft SQL Server, ODBC, HTTP, Tuxedo, and socket protocols.

Verify Playback Row Counts

This option specifies whether the number of rows that were affected by a SQL statement, or the number of bytes that were affected by a request to a Web server, are inserted into the script.

This information helps you determine whether the statement or request executed during playback behaves as it did during recording. If a different number is returned during playback, the Test Log window notes the discrepancy.

Verify playback row counts has the following meanings:

State of check box	Meaning
The check box is selected.	Insert into the script the number of rows or bytes affected by a SQL statement or Web request.
The check box is cleared.	Do not insert the number of affected rows or bytes into the script.

For an example of the effect that this check box has on a generated script, see *Example of the Verify Playback Check Boxes* on page 6-13.

Verify Playback Return Codes

This option specifies whether you want the return code for an executed SQL statement to be inserted into the script.

During playback, TestManager checks whether the return code for a SQL statement executed during playback matches the return code for the same statement executed during recording. If the SQL statement returns a different code during playback, the Test Log window notes the discrepancy.

The **Verify playback return codes** check box has the following meanings:

State of check box	Meaning
The check box is selected.	Report SQL return codes for each SQL statement executed in the script.
The check box is cleared.	Do not insert any SQL return codes into the script.

A single SQL statement can return multiple error codes.

Example of the Verify Playback Check Boxes

Suppose you want the script to execute the following SQL statement:

```
INSERT INTO mytable VALUES ("value1", "value2")
```

Depending on your **Verify playback row counts** and **Verify playback return codes** selections, Robot generates the SQL statement in one of the following ways:

Check box selected	**VU command and meaning**
Neither	`sqlexec ["x001"] "INSERT INTO mytable VALUES ('value1', 'value2')";` During recording, Robot does not report the data it collects from the execution of the SQL statement. During playback, TestManager assumes that any non-zero return code is an error. It pays no attention to the number of affected rows.
Verify playback return codes	`sqlexec ["x001"] EXPECT_ERROR {-212}, "INSERT INTO mytable VALUES ('value1', 'value2')";` Robot records that the error code -212 was returned from the SQL statement. During playback, TestManager expects the SQL statement to return the error code -212. If the SQL statement returns a different code during playback, the Test Log window notes the discrepancy.
Verify playback row counts	`sqlexec ["x001"] EXPECT_ROWS 1, "INSERT INTO mytable VALUES ('value1', 'value2')";` Robot records that one row was affected by the SQL statement. During playback, TestManager expects the SQL statement to affect one row. If the SQL statement returns a different count during playback, the Test Log window notes the discrepancy.
Both	`sqlexec ["x001"] EXPECT_ERROR{-212}, EXPECT_ROWS 0, "INSERT INTO mytable VALUES ('value1', 'value2');` Robot records that the SQL statement returned the error code -212, and that no rows were affected. During playback, TestManager expects that the SQL statement will return error -212 and that no rows will be affected. If the SQL statement returns different results during playback, the Test Log window notes the discrepancy.

Because one SQL statement can return multiple error messages (for example, as a result of stored procedure execution), `EXPECT_ERROR` is an array. During playback, if an error code is returned that is not one of the values specified in the array, TestManager generates an error.

Bind Output Parameters to VU Variables

Select this check box to automatically script the VU expressions needed to contain the return values of output parameters. This applies only to emulation commands that support output parameter binding (currently the `IIOP_invoke` command). Clearing this box will shorten VU scripts, but you will have to manually script output parameter binding expressions and binding variable declarations for any output parameters of interest.

Playback Pacing

Controls the script's playback speed by including or excluding think time delays in the script. A **think time** delay includes both the time required for the user to think about and key in a request and the time required for the client to receive a response to the request.

Choose one of the following **Playback Pacing** settings:

Pacing setting	Meaning
per command	Plays back the script at a rate based on the actual time required to record and process each emulation command. For example, if the think time delay for a VU emulation command is 16,703 ms during recording, Robot adds the following line before that emulation command: `push Think_avg 16703;` This setting provides a realistic rate of playback on a per-command basis, reproducing delays in the same script locations as they occurred during recording. However, this setting adds more commands to the script than the **per script** setting does.
per script	Plays back the script at a rate based on the average time it took to record and process all emulation commands. All emulation commands use the same (average) think time delay. This setting and the **per command** setting both run a script in roughly the same amount of time. While playback timing is not as accurate on a per-command basis with the **per script** setting, it requires fewer commands to be inserted into the script. As a result, you can modify the scriptscript's average think time by editing one "think average" environment variable (VU `Think_avg` or Visual Basic `EVAR_Think_avg`).
none	Plays back the script on a **per script** basis, using the most recently-set value for VU `Think_avg` (or Visual Basic `EVAR_Think_avg`). The default value is 5000 ms. No think time commands are added to the script with this setting.

Pacing settings of **per command** and **per script** use a combination of think time and response time environment variables. For more information, see the *VU Language Reference*.

NOTE: If you set **Playback Pacing** to **none**, the **CPU/User threshold (ms)** and **Think maximum (ms)** options are disabled.

CPU/User Threshold (ms)

Specifies the dividing point, in milliseconds (ms), between CPU processing delays and delays due to user think time. In TestManager reports, delays that fall below the threshold you specify are considered CPU processing delays.

For example, an actual user might pause to think before selecting a student name from a SQL database. This delay is recorded as user think time. Once the user clicks on the student name, the time spent generating the SQL command and accessing the database is a CPU delay.

Typical thresholds range from 0 through 10,000 ms (10 seconds). Valid thresholds range from 0 through 2,000,000,000 ms (just over 23 days).

If you clear the **CPU/User Threshold (ms)** check box, all delays are considered think time delays.

Think maximum (ms)

Specifies the maximum think time delay to allow in a script. If you specify a maximum think time delay, no think time delay in the script will exceed it.

You can type a maximum think time or select one from the list. The valid range is 0 through 2,000,000,000 ms (just over 23 days).

If you clear the **Think maximum (ms)** check box, there is no limit to the length of a think time delay.

Setting Filtering Options

When your record a session, the session might include requests associated with a variety of protocols—for example, Oracle, Microsoft SQL Server, and HTTP. After recording, you can generate scripts that include requests for all of the recorded protocols or just some of them.

Typically, you filter protocols only with network and proxy recording, because you record all traffic to or from an IP address. API recording targets a client application on a specific computer, so you probably do not need to filter protocols if you are using this method. However, you can filter protocols with API recording if you notice that Robot is capturing raw socket information as well as other protocols (such as non-HTTP requests from a Web browser).

To see a list of the protocols that Robot records, select the **Manual Filtering** check box, record a script, and then view the list in the Manual Filtering dialog box at the end of the recording session.

How to Filter Protocols

To filter protocols from a recorded script:

1. Click **Tools** → **Session Record Options**.
2. Click the **Generator Filtering** tab.

NOTE: If you are recording DCOM protocol, Visual Basic version 6 must be installed on your Local computer, and the path to VB6.exe and RC.exe must be set.

Automatic and Manual Filtering

At script generation time, at the end of the recording session, Robot can automatically filter the protocols to generate (based on the protocols listed in the **Selected protocols** box in this tab), or you can specify the protocols that Robot should generate, depending on how you set the following check boxes:

Check box	Meaning
Auto Filtering	If selected, Robot generates scripts containing requests for the protocols listed in **Selected protocols**. If cleared, Robot generates scripts containing requests for all scriptable protocols it records.
Manual Filtering	If selected, Robot displays a list of connections, each consisting of a client, a server, and the protocol used in the client/server traffic. This list appears immediately after recording, just before Robot generates the script. When the list appears, select one or more connections to include in the script. If cleared, Robot does not display a list of connections.

NOTE: If you select **Manual Filtering**, Robot attempts to detect the computer name of each client and server it records before it generates the script. This can increase script generation time.

If you select both **Auto Filtering** and **Manual Filtering**, the list of connections includes all captured protocols:

- Protocols that you listed in the **Selected protocols** box are marked for inclusion in the script.
- Protocols that you listed in the **Available protocols** box are marked for exclusion from the script.

You can change these inclusion/exclusion designations in the Manual Filtering dialog box, so that requests for any captured protocol can be included in or excluded from the script. For information about using this dialog box, see *Choosing the Protocols to Include in a Script* on page 7-10.

Protocol Lists

When the **Auto Filtering** check box is selected, the following lists are enabled:

Available protocols – Protocols that are available for capture, but that you want to exclude from the script that Robot generates.

Selected protocols – Protocols to include in the script that Robot generates.

Jolt, Socket, and TUXEDO Protocols

If you click the **Advanced** button, the Advanced Protocol Filtering dialog box appears. This dialog box lets you define more detailed information about the Jolt, socket, and TUXEDO protocols specified in the **Selected protocols** list:

- **Jolt protocol.** By default, TestManager plays back all Jolt sessions recorded, regardless of which Jolt servers on the network they may have connected to. To limit playback to Jolt sessions connected to a specific Jolt server, specify values for both of the following filters:

  – **JSL Host** – Filter out all sessions not connected to a Jolt Server Listener at this host address.

  – **JSL Port** – Filter out all sessions not connected to a Jolt Server Listener at this TCP port number.

  To play back specific Jolt sessions, specify values for either or both of the following filters:

  - **UserName** – Filter out all sessions that do not have this value as the userName parameter of the client's JoltSession object constructor invocation.
  - **UserRole** – Filter out all sessions that do not have this value as the userRole parameter of the client's JoltSession object constructor invocation.

- **Socket protocols**. All Robot recording captures socket protocols. Socket protocols are at a level below the other protocols that Robot captures. The following check boxes define the socket protocols you can include in a script:
  - **Well-known protocols (FTP, Telnet, ...)** – Select this check box to include common socket protocols in the generated scripts. These protocols typically use port numbers 1 through 1,023 (for example, FTP uses port 21).
  - **Unrecognized protocols** – Select this check box to include other socket protocols in the generated scripts. For example, select this box to capture requests from a Java applet that communicates with a server via sockets.

- **TUXEDO protocol**. TestManager can play back the traffic from one TUXEDO connection at a time. If you record traffic from multiple TUXEDO connections, you can specify which conversation to generate in the following **WorkStation Listener** (WSL) boxes:
  - **WSL Host** – The name of the workstation listener host.
  - **WSL Port** – The TCP/IP port number for the host.

  If you provide workstation listener information, you must supply values for both boxes.

  In addition, you can specify the conversation to generate by supplying values for the following user-defined fields of the TPINIT request message. These fields are set by the client in the TPINIT typed buffer that is passed as an argument to the TUXEDO API function `tpinit()`. You can define either or both of the following fields. If you define both, the request message must contain both field values (that is, a logical AND operation):
  - **Usrname** – The client name.
  - **Cltname** – The user name.

  When specifying a TUXEDO connection, you can use either the WSL or TPINIT method, or you can use both methods.

NOTE: The TPINIT method is an advanced method for specifying a TUXEDO connection. Typically, it is used only if the WSL method does not produce satisfactory results.

Providing Protocol-Specific Information

If you are recording HTTP, Oracle, TUXEDO, or IIOP requests, you need to supply Robot with certain information.

Controlling the Values Accepted When an HTTP Script Is Played Back

You can set recording options that control which status values are acceptable when a script that accesses a Web server is played back. If you do not set any recording options, the script plays back successfully only if the playback conditions *exactly* match the conditions during recording. However, you can set recording options so that a script plays back successfully even if:

- The server responds with partial or full page data during record or playback.
- The response was cached during record or playback.
- The script was redirected to another http server during playback.
- You are recording a number of HTTP scripts and plan to play them back in a different order.

To expand the conditions under which a script will play back successfully:

1. Click **Tools** → **Session Record Options**.
2. Click the **Generator per protocol** tab.
3. Select **HTTP** at the **Protocol** section, and then select one or more of the following:

   a. **Allow partial responses**

      Select this option to enable a script to play back successfully if the HTTP server responds with partial data during playback. This generates a script that sets the TSS environment variable **Http_control** to `HTTP_PARTIAL_OK`. Leaving this box cleared enforces strict interpretation of recorded responses during playback.

   b. **Allow cache responses**

      Select this option to enable a script to play back successfully if a response is cached differently during playback. This generates a script that sets the TSS environment variable **Http_control** to `HTTP_CACHE_OK`. Leaving this box cleared enforces strict interpretation of recorded cache responses during playback.

   c. **Allow redirects**

Select this option to enable a script to play back successfully the script was directed to another HTTP server during playback or recording. This generates a script that sets the TSS environment variable **Http_control** to `HTTP_REDIRECT_OK`. Leaving this box cleared enforces strict interpretation of recorded redirects during playback.

d. Use HTTP keep-alives for connections in a session with multiple scripts.You should generally leave this box cleared.

Selecting this option provides more accurate representation of keep-alive behavior, but at a cost—if you loop scripts or play them back in a different order, you will have to manually edit your scripts to achieve successful playback.

Therefore, you should select this option only if:

– You will not loop scripts or play them back in a different order (or, if you do, you do not mind editing the scripts).

– You want to preserve the browser's connection keep-alive behavior that is in the recorded session.

The default behavior for multiple script recordings is to not use keep-alives so that you don't have to be aware of persistent http connections that span script boundaries when you loop or change script ordering. However, the default behavior may result in increased HTTP server overhead due to the absence of keep-alives.

Supplying Variable Data Values to an HTTP Script

Dynamic data correlation is a technique to supply variable data values to a script when the transactions in a script depend on values supplied from the server.

When you record an http script, the Web server may send back a unique string, or session ID, to your browser. The next time your browser makes a request, it must send back the same session ID to authenticate itself with the server.

The session ID can be stored in three places:

- In the Cookie field of the HTTP header.
- In an arbitrarily named field of the HTTP header.
- In an arbitrary hidden field in an actual HTML page.

TestManager finds the session IDs (and other correlated variables) and, when you run the suite, automatically generates the proper script commands to extract their actual values.

Before you record a script, you can tell TestManager to correlate all possible values (the default), not to correlate any values, or to correlate only a specific list of variables that you provide.

To specify the level of data correlation:

1. Click **Tools** → **Session Record Options.**
2. Click the **Generator per protocol** tab.
3. At **Correlate variables in response**, select one of the following:
   - **a** **All** – All variables are correlated. You should generally select this option. Select another option only if you encounter problems when you play back the script.
   - **b** **Specific** – Only the variables that you select are correlated.
   - **c** **None** – No variables are correlated.

If you have selected **All** or **Specific**, your generated VU script will contain the function `http_find_values`. This function finds the value of items that the server returns and the user does not change. It then correlates these values and places them in a system-defined variable in the form `SgenRes_00`*n*.

Examine the script to determine whether the proper values are being correlated. If you want fewer values to be correlated, change the **Correlate variables in response** option to **Specific**, and then use the **Add** and **Remove** buttons to select only the names that you want correlated.

For example, assume you enter data in a form during recording, and the form has a field that you cannot modify—for example, `UNITED STATES`. In the generated script, the `http_response` emulation command will show the form as follows:

```
"<form name = \…
…
"\t<input type=\"hidden\" name=\"Country\" value=\"UNITED
STATES\">\r\n
```

TestManager can determine that this is an item for correlation, and adds an `http_find_values` function to your script. This function puts the `UNITED STATES` value in a variable. Your script will also contain a line that looks like this

```
{
string SgenRes_002[];
SgenRes_002 = http_find_values("Country", HTTP_FORM_DATA, 1);
#if 0
<UNITED STATES>
#endif
}
```

If you do not want UNITED STATES to be correlated, choose **Specific** but do not select the Country name from the list. You do not have to re-record the script; you can simply regenerate it from the session.

Providing the Name of an Oracle Database

If you are using the network or proxy methods to record Oracle requests, you must provide the name that the client application uses to connect to the Oracle database. This name is in Oracle's tnsnames.ora file. You can later play back the script against another Oracle database by changing this name.

To provide this name:

1. Click **Tools** → **Session Record Options.**
2. Click the **Generator per protocol** tab.
3. Select Oracle at the **Protocol** section.
4. Enter the name that the client application uses in the **Database name** box.

Assigning a Prefix to TUXEDO Command IDs

If you are recording TUXEDO requests, you can have Robot automatically assign an identifying prefix to TUXEDO emulation command IDs. The prefix allows TestManager reports to be filtered by emulation command type. The prefix that you set here overrides the **Command ID prefix** in the **Generator** tab.

For example, if you select the **Use command type to prefix emulation commands** check box, the third tux_tpcall emulation command in the script has the following command ID:

```
tux_tpcall ["tcal003"]
```

Robot assigns the following command ID prefixes for particular TUXEDO commands:

Command	**Prefix**	**Command**	**Prefix**
tux_bq	tbq	tux_tpinit	tini
tux_tpabort	tabo	tux_tpnotify	tnot
tux_tpacall*	tacaR	tux_tppost	tpos

* tacaR applies when tux_tpacall involves a request only. When tux_tpacall involves both a request and the reply (request/reply turnaround), the prefix is tacaT. This is used for asynchronous request timing only and is never assigned to an emulation command.

Command	Prefix	Command	Prefix
tux_tpbroadcast	tbro	tux_tprecv	trec
tux_tpcall	tcal	tux_tpresume	tres
tux_tpcommit	tcom	tux_tpsend	tsen
tux_tpconnect	tcon	tux_tpsubscribe	tsub
tux_tpdequeue	tdeq	tux_tpsuspend	tsus
tux_tpdisconnect	tdis	tux_tpterm	tter
tux_tpenqueue	tenq	tux_tpunsubscribe	tuns
tux_tpgetrply	tget		

* tacaR applies when tux_tpacall involves a request only. When tux_tpacall involves both a request and the reply (request/reply turnaround), the prefix is tacaT. This is used for asynchronous request timing only and is never assigned to an emulation command.

Assigning a Prefix to IIOP Command IDs and Including IORs in IIOP_bind

If you are recording IIOP requests, you can have Robot automatically assign an identifying prefix to IIOP emulation command IDs. You can also include the original IORs in iiop_bind commands.

To assign an identifying prefix or to include the original IORs in iiop_bind commands:

1. Click **Tools** → **Session Record Options**.
2. Click the **Generator per protocol** tab.
3. Select **IIOP** at the **Protocol** section, and then select one or more of the following:
   – **Use operation name to prefix emulation command IDs**

     Select this check box to have Robot automatically prefix the command IDs with the operation being invoked. This applies to the iiop_invoke command. The prefix allows TestManager reports to be filtered by operation. The prefix that you set here overrides the **Command ID prefix** in the **Generator** tab.

   – **Include original IORs in iiop_bind commands**

Select this check box to make the `ior` argument of every scripted `iiop_bind` emulation command contain the stringified form of the IOR originally recorded for that object reference. The `ior` argument is required by the IOR bind modus. Clearing this box will shorten scripts, but you will have to manually enter the `ior` argument value when using the IOR bind modus.

Assigning a Prefix to DCOM Command IDs

If you are recording DCOM requests, you can have Robot automatically assign an identifying prefix to DCOM emulation command IDs. Emulation command IDs consist of a prefix and a three-digit numeric suffix. Robot automatically increments the numeric suffix by a value of 1, which allows each command to be uniquely identified.

To assign an identifying prefix :

1. Click **Tools** → **Session Record Options.**
2. Click the **Generator per protocol** tab.
3. Select **DCOM** at the **Protocol** section, and then select an event label. The label you select determines the prefix of the emulation command IDs in your script.
   – **User Generated**. Lets you generate a command ID prefix. After you select this option, click the **Command ID prefix** in the **Generator** tab, and then enter the prefix that you want generated. If you do not enter a prefix, the command ID follows the name of your script. For example, if you do not enter a prefix and the your script is named Accounting, the command IDs are `[ACCOUNT~001]`, `[ACCOUNT~002]`, and so forth.
   – **SCRIPT NAME**. The command ID prefix is the script name. If your script is named Accounting, the command IDs are `[ACCOUNTING001]`, `[ACCOUNTING002]`, and so forth.
   – **Method**. The command ID prefixes are the method names in your program.
   – **Class**. The command ID prefixes are the class names in your program. Any commands within a class follow the class name.
   – **Class.Method**. The command ID prefixes are in the format *class-name.method-name*.
   – **Library.Class.Method**. The command ID prefixes are in the format *library-name.class-name.method-name*.
   – **None**. Command IDs are omitted from your script.

Setting General Recording Options

Robot lets you set general recording options, which apply to all session recording methods. To set general recording options:

1. Click **Tools** → **Session Record Options**.
2. Click the **General** tab.

This tab lets you set:

- Automatic prefixes for scripts and sessions
- Whether Robot should prompt you to start an application after you start network or proxy recording
- Various settings for the Session Recorder window

The following sections describe these options.

Autonaming Prefixes

In the **Autonaming Prefixes** group box, you can define prefixes for default script and session names. If you define a prefix, Robot appends a consecutive number to the prefix, and uses the unique prefix and number combination as the name it suggests each time it prompts you to define a script or session name during recording.

For example, if your prefix for a script is `Script`, `Script1` is the default script name for the first script you record, `Script2` is the default script name for the second script you record, and so on.

During recording, when you are prompted to define a script or session name, you can accept the default name, modify it, or change it completely.

Optionally, you can leave either or both of these boxes blank, so that session and script names consist of the names that you define during recording.

Names can have a maximum of 40 characters. Consequently, the maximum length of a prefix is 40 characters less the number of digits in the numeric suffix of the last script or session that you recorded.

Start Application

To have Robot prompt you to start the client after you begin recording, select **Prompt for application name on start recording**.

With *API* recording, you must start the client after you begin the recording process, when Robot prompts you to do so. As a result, the **Prompt for application name on start recording** check box is disabled with API recording.

With *network* or *proxy* recording, you can start the client before or after you begin recording. However, connection information is sometimes lost if you start the application before you begin recording. We recommend that you start the application after you begin recording.

If you start the client after you begin recording, you can have Robot prompt you to start the client, or you can start it without being prompted (for example, if the client is running on a different computer than Robot).

Setting the Recorder Window

The Session Recorder window displays information about client requests and server responses as they occur during a recording session. This window appears automatically, in either a normal or minimized state, when you begin recording.

The following check boxes in the **Recording Window** group box set a variety of options for the Session Recorder window:

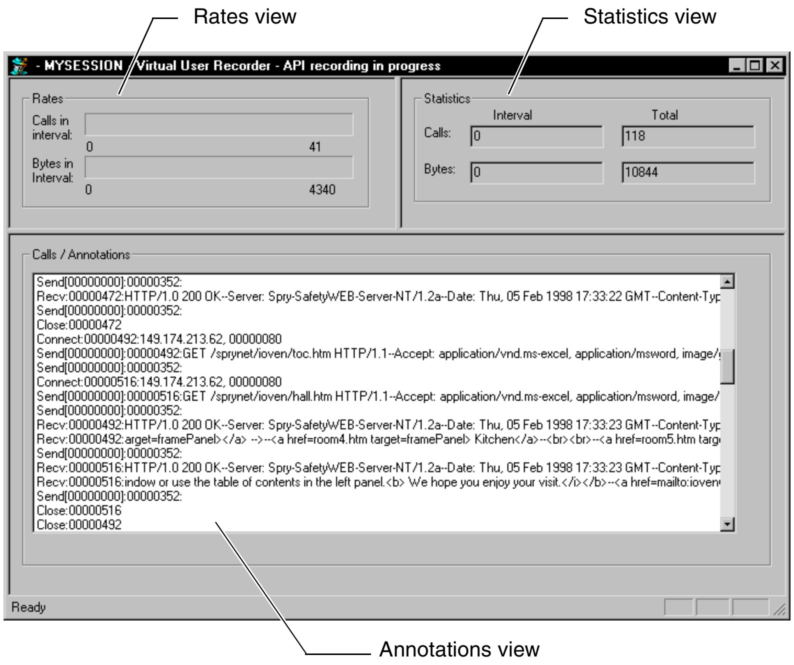
Minimize on start recording – Select this check box to minimize the Session Recorder window when you start recording. Clear this check box to display the window in a normal state.

During recording, the Session Recorder icon is displayed in the taskbar, to the left of the clock. The icon blinks when Robot is capturing a request or response. This icon serves as a visual cue that Robot is recording, even when the Session Recorder window is minimized.

Show Rates view – Select this check box to display the number of calls and bytes in the current three-second interval.

Show Statistics view – Select this check box to display the total number of calls and bytes in the recording session.

Show Annotations view – Select this check box to display the comments, start/stop blocks, timers, or synchronization points that you insert into the script during recording.



```
Send[00000000]:00000352:
Recv:00000472:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:22 GMT--Content-Typ
Send[00000000]:00000352:
Close:00000472
Connect:00000492:149.174.213.62, 00000080
Send[00000000]:00000492:GET /sprynet/ioven/toc.htm HTTP/1.1--Accept: application/vnd.ms-excel, application/msword, image/
Send[00000000]:00000352:
Connect:00000516:149.174.213.62, 00000080
Send[00000000]:00000516:GET /sprynet/ioven/hall.htm HTTP/1.1--Accept: application/vnd.ms-excel, application/msword, image/
Send[00000000]:00000352:
Recv:00000492:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:23 GMT--Content-Typ
Recv:00000492:arget=framePanel></a> -->--<a href=room4.htm target=framePanel> Kitchen</a>--<br><br>--<a href=room5.htm targ
Send[00000000]:00000352:
Recv:00000516:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:23 GMT--Content-Typ
Recv:00000516:indow or use the table of contents in the left panel.<b> We hope you enjoy your visit.</i></b>--<a href=mailto:ioven
Send[00000000]:00000352:
Close:00000516
Close:00000492
```

Defining a Client or Server Computer

If you are using network or proxy recording, and the computer that you want to use is not listed in the Method Network and Method Proxy tabs, you can add it to the list.

To add a client or server computer:

1. Click **Tools** → **Session Record Options**.
2. Click the **Method:Network** tab or **Method:Proxy** tab, depending on whether you are adding a computer for network or proxy recording.
3. Click **Manage Computers**.
4. In the **Name** box of the **Computers** group, type a name to associate with the network name of the computer that you are adding. You can assign any name up to 40 characters.
5. Type the computer's network name.

You can find the computer's network name in the **Identification** tab of the Windows NT Network dialog box (**Start** → **Settings** → **Control Panel** → **Network**). In this tab, the network name is labeled **Computer Name**.

Alternatively, you can type the computer's IP address associated with the network name. However, because DHCP-provided IP addresses can change, you should type a network name.

6. Optionally, click **Ping** to make sure that the network name you just typed is correct. If it is correct, "Successful Ping of *network name*" appears in the status bar.
7. Select **Client System** to list this computer as a client. Select **Server System** to list this computer as a server. You can select both choices.
8. Click **Add**.
9. Take the following steps to use a port number with the network name. A port number is required for servers used in proxy recording:
   a. In the **Ports** group, type a name to associate with the port number that you are adding. You can assign any name up to 40 characters.
   b. Type the port number to use with the computer's network name.
   c. Click **Add**.
10. Click **Close**.

The computer is now added to the Rational project and appears in the list of computers.

NOTE: You can also define a computer with Rational Administrator. The Administrator lets you define additional information about the computer, such as what type of operating system the computer runs on.

Removing a Computer or Port

To remove a client or server computer from the **Method:Network** tab:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method:Network** tab.
3. Click **Manage Computers**.
4. In the **Name** box of the **Computers** group, select the computer name to remove from the list.
5. Click **Remove**.

6. Click **Close**.

To remove a port name and number from a computer's address:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method:Network** tab.
3. Click **Manage Computers**.
4. In the **Name** box of the **Computers** group, select the computer name associated with the port that you are removing.
5. Under **Ports**, select the port name to remove.
6. Click **Remove**.
7. Click **Close**.

Authenticating Login

If you are running tests against a database that requires a user ID and password, you must provide them when the script is recorded and when the script is played back.

During recording, Robot attempts to detect the user ID, password, and other login information. When successful, Robot stores this information in an Authentication Datapool. In addition, Robot adds the user ID, but not the password, to the script.

During playback, when a user ID and password are required for a database, Robot finds the user ID in the script, and then attempts to find a password in the Authentication Datapool that is associated with the user ID. Robot uses the first active password that it finds for the user ID.

When to Modify the Authentication Datapool

If Robot detects the user ID, password, and other login information provided during recording, it updates the Authentication Datapool automatically. If your login information does not subsequently change, you never need to modify the Authentication Datapool.

However, there are times when modifying the Authentication Datapool is necessary:

- If Robot cannot detect the password during recording. For example, Oracle passwords are almost always transmitted in encrypted form. As a result, you typically need to enter Oracle passwords into the Authentication Datapool.
- If you change your password after the password is recorded and stored in the Authentication Datapool.

- If the server does not allow a user to log into the database multiple times simultaneously.

  In other words, suppose Robot detects your user ID and password when you record a script. Robot writes the information to the Authentication Datapool. During playback, TestManager retrieves your user ID and password from the Authentication Datapool and uses the information to log the virtual tester into the database.

Modifying the Authentication Datapool with TestManager

To modify the Authentication Datapool with TestManager:

1. In TestManager, click **Tools** → **Manage** → **Datapools**.
2. Click **RTAuthentication** (the Authentication Datapool).
3. Click **Edit**.
4. Click **Edit Datapool Data** in the Datapool Properties dialog box.
5. Each row in the Edit Datapool dialog box contains the following information:

Datapool column	Meaning
State	Whether the password in this row is ACTIVE or INACTIVE. Select one of these choices from the list box. If a user provides a password for a particular service, and the Authentication Datapool already contains a password for that user and service, Robot automatically makes the currently provided password active and the earlier password inactive. If there is more than one active password, Robot uses the first active password that it finds in the Authentication Datapool.
Class	The class is always **SQL** in this release.
Subclass	One of the following values: ▪ **Oracle** ▪ **Sybase** ▪ **SQL Server**
Service	The name of the database server as it is defined in the database environment. Do not use a computer name for the name of Service. This is the same name that Robot inserts into the *`server`* argument of the `sqlconnect` command during recording.
Login	The user ID.

Datapool column	Meaning
Password	The password for this user ID.

6. Repeat the last step for each user ID and password that you need to enter.
7. Click **Save**, and then click **Close**.
8. Click **OK** to close the Datapool Properties dialog box, and then close the Manage Datapools dialog box.

Modifying the Authentication Datapool During Recording

If you need to insert many rows of login information into the Authentication Datapool, it is best to do so through TestManager.

But if you need to add just a few rows of login information, you should do so during recording, when Robot prompts you for this information.

Unique Features of the Authentication Datapool

The Authentication Datapool is similar to other datapools that you edit with TestManager. However, there are differences:

- An empty Authentication Datapool is included with the Rational Test software.
- The Authentication Datapool is used strictly for login information. You cannot assign any standard or user-defined data types to the columns in an Authentication Datapool.
- Do not delete or rename the Authentication Datapool.
- You should not add to or remove the columns in the Authentication Datapool.
- The Authentication Datapool is not associated with the `DATAPOOL_CONFIG` statement or any datapool commands.

Managing Proxies

If you are using the proxy recording method, you need to create a proxy and identify client/server pairs that will communicate through the proxy.

After you have defined a proxy relationship, you can manage your proxies as follows:

- Starting and stopping proxy service.
- Monitoring proxy activity.

- Deleting a client/server pair.
- Deleting a proxy.
- Reassociating a proxy with a client/server pair.

The following sections describe these functions.

Starting and Stopping Proxy Service

Proxy service is a system service that lets you use the proxy recording method. Proxy service starts automatically when you:

- Install TestStudio.
- Start your system.
- Open the Session Record Options dialog box and click the **Method:Proxy** tab.

Proxy service stops automatically when you shut down Windows.

Explicitly Starting or Stopping Proxy Service

Typically, you will want to keep proxy service running, even when you shut down Robot. But if you need to explicitly stop proxy service, or start it up again after stopping it, follow these steps:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method** tab and make sure that **Proxy recording** is selected.
3. Click the **Method:Proxy** tab.
4. Click **Proxy Admin**. The Proxy Administration dialog box appears.

   The **Status** box shows whether proxy service is Running or Stopped:

   – If proxy service is Stopped, click **Start** to run it.

   – If proxy service is Running, click **Stop** to stop it.

Alternatively, you can start and stop proxy service as follows:

1. Click **Start** → **Settings** → **Control Panel**.
2. Double-click **Services**.
3. Select **ProxyServer Service**.
4. Click either **Start** or **Stop**.

Recreating Proxies After Proxy Service Is Stopped

The proxy computers are listed in the **Existing Proxies** grid of the Proxy Administration dialog box.

When proxy service stops, either explicitly or during Windows shutdown, all proxy computers are deleted. Therefore, you must create new proxy computers before you start proxy recording. For information about creating a proxy computer, see *Creating a Proxy Computer* on page 6-8.

Proxy service is *not* automatically shut down (and therefore, proxy computers are not deleted) if you:

- Exit Robot, but do not exit Windows.
- Log off of your NT session, but do not exit Windows.

Monitoring Proxy Activities

You can view information about existing proxies in the **Existing Properties** grid of the Proxy Administration dialog box. The grid has the following columns:

Column in grid	**Meaning**
Proxy:Port	The name and port number of the proxy computer.
Server:Port	The name and port number of the server computer. Client requests to the server are routed through the proxy.
Connections	The current number of connections to the server.

Column in grid	Meaning
State	The state of the proxy: ACTIVE – The proxy is available for recording. RECORD – The proxy is recording. CLOSE_WAIT – A request has been issued to delete the proxy. The proxy is deleted as soon as it is no longer in use. If the proxy is in use, new connect requests are accepted. CLOSE_WAIT_NOCONN – A request has been issued to delete the proxy. The proxy is deleted as soon as it is no longer in use. If the proxy is in use, new connect requests are not accepted.

Click **Refresh** to update the information in the grid.

Deleting Client/Server Pairs

You should remove client/server pairs that are listed in the **Method:Proxy** tab but are not involved in the session you that want to record. To do so:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method** tab and make sure that **Proxy recording** is selected.
3. Click the **Method:Proxy** tab.
4. Select the client/server pair to delete from the **Client/Server pairs for recording** list.
5. Click **Remove**.
6. Click **OK**.

Deleting a Proxy

To delete the proxy relationship between a server and its proxy:

1. Click **Tools** → **Session Record Options.**
2. Click the **Method** tab and make sure that **Proxy recording** is selected.
3. Click the **Method:Proxy** tab.
4. Click **Proxy Admin**. The Proxy Administration dialog box appears.
5. In the grid of existing proxies, select the proxy to delete.
6. Click **Delete Proxy**.
7. In the Delete Proxy dialog box, click one of the following buttons:

- **Wait for all connections to close, accept new connections**

  Delete the proxy as soon as it is no longer in use. If the proxy is currently in use, allow new connections. This selection is associated with the proxy state CLOSE_WAIT.

- **Wait for all connections to close, do not accept new connections**

  Delete the proxy as soon as it is no longer in use. If the proxy is currently in use, do not allow new connections. This selection is associated with the proxy state CLOSE_WAIT_NOCONN.

- **Immediately close all connections**

  Delete the proxy immediately. If the proxy is currently in use, close the connections.

8. Click **Delete**.

In the Proxy Administration dialog box, the proxy that you deleted has either been removed, or it is still present but has a modified state, depending on your choice in step 7.

Re-Creating Proxies that Have Been Removed

If a proxy is removed, any client/server pairs that communicated through that proxy can no longer do so. To use these client/server pairs in proxy recording again, you must first re-create the proxy. You re-create a proxy by reassociating it with the server.

When you click the **Method:Proxy** tab of the Session Record Options dialog box, any client/server pairs that are no longer associated with a proxy are displayed in the Delete Pairs Without Proxies dialog box. You can either:

- Click **OK** to delete all of the client/server pairs.
- Click **Cancel** to re-create one or more proxies.

To re-create a proxy:

1. Click **Proxy Admin** in the **Method:Proxy** tab.
2. In **Proxy:Port**, specify the proxy computer's port number. Note that Robot has already detected and specified the proxy computer's name.
3. You can specify any available port number. Avoid the "well-known" port numbers (those below 1024). If you specify a port number that is unavailable, Robot prompts you for a new port number.
4. In the **Server:Port** list, select the server to be reassociated with the proxy.
5. Click **Create Proxy**.

The proxy that you just created appears in the **Existing Proxies** list.

▸ ▸ ▸ C H A P T E R 7

Recording Sessions

This chapter describes how to record sessions and generate scripts. The chapter includes the following topics:

- Recording a session
- Recording a single script in a session
- Getting feedback during recording
- Canceling scripts during recording
- Choosing the protocols to include in a script
- Playing back a script quickly
- Working with sessions
- Coding a script manually
- Defining script properties
- Managing scripts and sessions
- Recording a Session

You do not record scripts for performance testing directly as you do GUI scripts. Instead, you record a *session*. After recording, Robot generates one or more scripts from the session.

A Robot recording *session* contains all of the client requests and server responses issued from the time you begin recording until the time you stop recording. Robot stores all of the requests and responses recorded during the session in a session file (.wch). The session file is sometimes called the watch file.

What You Can Record in a Session

Robot gives you considerable recording flexibility. You can record:

- Multiple transactions. For example, you can record a data entry transaction and a query transaction in the same recording session, one after the other.
- Transactions against the same server or different servers. For example, you can record one transaction against one Web server, and then record another transaction against a different Web server.
- Different types of requests in the same session. For example, you can record Oracle, Microsoft SQL Server, HTTP, and TUXEDO requests in a session.

Where Files Are Stored

By default, scripts (.s) are stored in the TMS_Scripts directory of your current project datastore. For example, if the current project is MyProject, here is what the directory hierarchy looks like:

c:\MyProject\TestDatastore\DefaultTestScriptDatastore\TMS_Scripts

Session files (.wch) are stored in the TMS_Sessions directory—for example:

c:\MyProject\TestDatastore\DefaultTestScriptDatastore\TMS_Sessions

Restoring Robot During Recording

When you begin recording, Robot becomes minimized by default, allowing you unobstructed access to the client application.

At any time during recording, you can restore the Robot window without affecting the client/server traffic you are recording. For example, you might want to restore the Robot window to:

- Reset recording options, such as the script generation options in the **Generator** tab.
- Insert features such as timers, blocks, and synchronization points through the Robot **Insert** menu rather than through the floating toolbar.

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

- Click the **Open Robot Window** button on the Session Record floating toolbar.
- Click the **Robot** icon on the Windows taskbar.

You can also use the standard Windows ALT+TAB key combination.

Recording a Single Script in a Session

Use the following procedure to record a script. For information about splitting a recording session into multiple scripts, see *Splitting a Session into Multiple Scripts* on page 7-16.

To record a script:

1. In Robot, click the **Record Session** button.

   Alternatively, click **File** → **Record Session**, or press CTRL+SHIFT+R.

2. Type the *session* name (40 characters maximum), or accept the default name. You will specify the *script* name when you finish recording the script.

   If you have not yet set your session recording options, do so now by clicking **Options**.

3. Click **OK** in the Record Session - Enter Session Name dialog box. The following events occur:

   – Robot is minimized (default behavior).

   – The Session Record floating toolbar appears (default behavior). You can use this toolbar to stop recording, redisplay Robot, split a script, and insert features into a script. (See *Using the Floating Toolbars* on page 7-5.)

   – The Session Recorder icon appears on the taskbar. The icon blinks as Robot captures requests and responses.

   – If the client application is already running, the scripts Session Recorder window appears in a normal or minimized state. During recording, this window displays statistics about the recorded client/server conversation as it occurs. (See *Getting Feedback During Recording* on page 7-7.)

   – If the client application is not running, the Start Application dialog box appears before the Session Recorder window appears.

4. If the Start Application dialog box is displayed, provide the following information, and then click **OK**:

   – The path of the executable file for the browser or database application.

   – Optionally, the working directory for any components (such as DLLs) that the client application needs at runtime.

   – Optionally, any arguments that you want to pass to the client application.

The Start Application dialog box appears only if you are performing API recording, or if you are performing network or proxy recording and selected **Prompt for application name on start recording** in the **General** tab of the Session Record Options dialog box.

5. Perform the transactions that you want to record.

   As the application sends requests to the server, notice the activity in the Session Recorder window. Progress bars and request statistics appear in the top of the window.

   If there is no activity in the Session Recorder window (or if the Session Recorder icon never blinks), there is a problem with the recording. Stop recording and try to find the cause of the problem.

6. Optionally, insert features such as blocks and timers through the Session Insert floating toolbar or through the Robot **Insert** menu.
7. Optionally, when you finish recording transactions, close the client application.
8. Click the **Stop Recording** button on the Session Record floating toolbar.
9. In the **Name of the just-recorded script** box, type or select a name for the script that you just finished recording, or accept the default name.

   Alternatively, to cancel the requests you made since you began recording, click **Ignore just-recorded information**. For more information, see *Cancelling Scripts During Recording* on page 7-9.

10. Click **OK**.

    The Generating Scripts dialog box appears. This dialog box reflects the progress of the automatic script generation. After a few seconds (or longer, depending on the length of the session), script generation ends, the message *Completed successfully* appears in the status bar, and the **OK** button is enabled.

    During script generation, you might see:

    – The Missing Password dialog box. For more information about this dialog box, see *Providing a Missing Password* on page 7-5.

    – The Manual Filtering dialog box. For information about this dialog box, see *Choosing the Protocols to Include in a Script* on page 7-10.

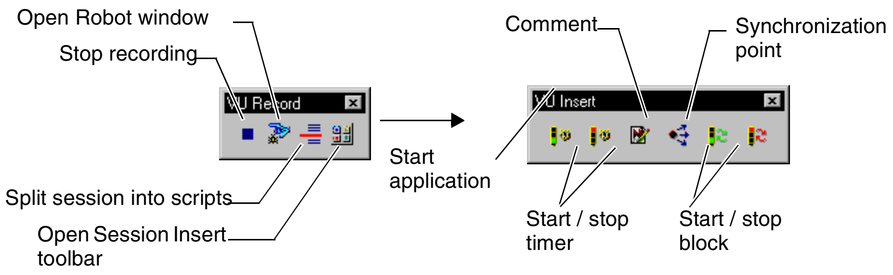
11. Click **OK** in the Generating Scripts dialog box. The script that you recorded appears in the Robot window.

Using the Floating Toolbars

When you begin to record a script, Robot displays a floating toolbar by default. The Session Record toolbar gives you access to activities you might want to perform while Robot is hidden from view during recording.

If you click the rightmost button on the Session Record toolbar, you display the Session Insert toolbar. This toolbar lets you insert features into the script and start another application during recording.

The following figure shows the Session Record toolbar and the Session Insert toolbar:



If Problems Occur During Script Generation

If problems occur during script generation, the following message appears in the status bar of the Generating Scripts dialog box:

```
Completed with warnings and/or errors.
```

To see the list of errors, click **Details**. If the text of an error is truncated, you can either:

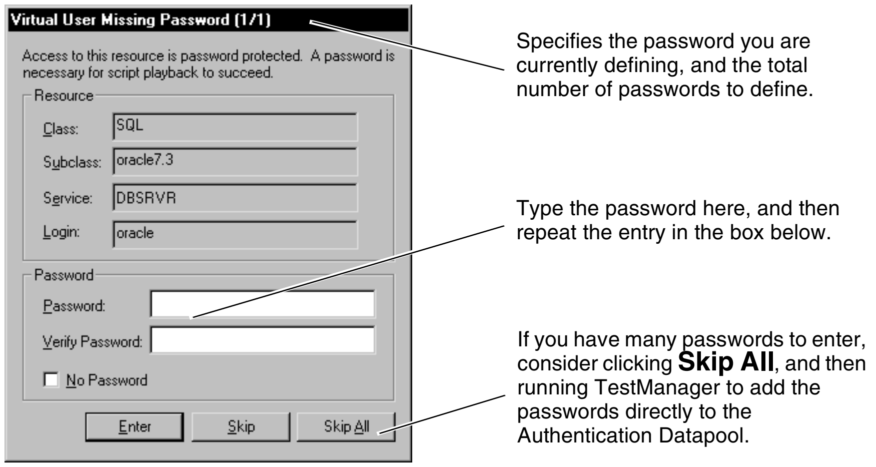
- Double-click the text to see the entire message.
- Press CTRL+C to copy the text to the Clipboard.

Providing a Missing Password

During recording, if you provide a user ID and password required to access the database, Robot attempts to detect this login information. If Robot can detect all of this information, it writes the information to the Authentication Datapool. (During script playback, TestManager checks the Authentication Datapool whenever an emulated user needs to provide a valid user ID and password when accessing the database.)

If Robot cannot detect a password that you provided during recording, and it cannot find a valid password for the associated user ID in the Authentication Datapool, Robot prompts you to provide the password just before generating the scripts you recorded.

Robot prompts you to provide each password that it could not detect, one by one, in the Missing Password dialog box. In the title bar, Robot displays the total number of passwords that it needs to prompt you for, and the number of the password that you are currently providing. For example, if **(1 of 3)** appears in the title bar, Robot is currently prompting you for the first of three passwords.


Virtual User Missing Password (1/1)
Access to this resource is password protected. A password is necessary for script playback to succeed.
Resource
Class: SQL
Subclass: oracle7.3
Service: DBSRVR
Login: oracle
Password
Password:
Verify Password:
No Password
Enter
Skip
Skip All
Specifies the password you are currently defining, and the total number of passwords to define.
Type the password here, and then repeat the entry in the box below.
If you have many passwords to enter, consider clicking Skip All, and then running TestManager to add the passwords directly to the Authentication Datapool.

To Provide a Password

To add a password for the user ID displayed in **Login**:

1. Type the password in **Password**, and type it again in **Verify Password**. An asterisk (*) represents each character that you type.

   Alternatively, if no password is needed for this user ID, select **No Password**.

2. Click **Enter**.

Robot automatically closes the dialog box when you finish providing passwords.

To Skip One or More Passwords

If you do not know a password for a particular user ID, click **Skip**. You will need to provide the password later (for example, by editing the Authentication Datapool).

If you prefer to provide passwords for all the user IDs at a later time, click **Skip All**. You may prefer to do this if there are many passwords to provide.

Getting Feedback During Recording

When you begin to record a session, the Session Recorder appears, either in a normal or minimized state. You can use this window to monitor client activity during the recording session.

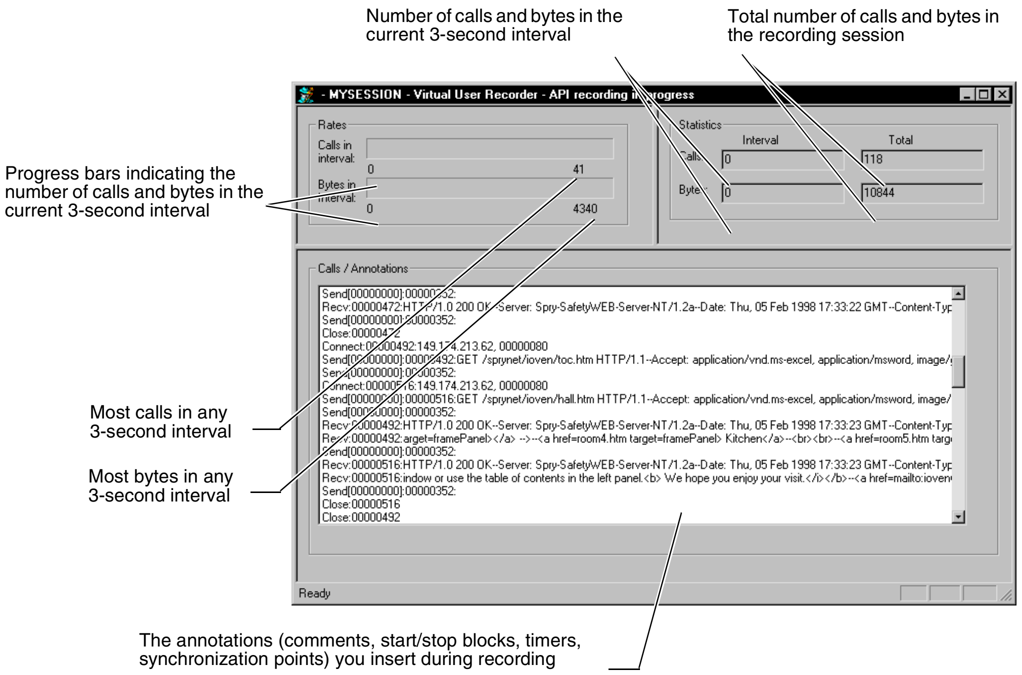
The Session Recorder tracks a variety of statistics about the client/server conversation, such as the number of bytes the client sends or receives in a call.

To help you gauge the rate at which client/server activity occurs, the Session Recorder displays its data at three-second intervals. Information in the Session Recorder window is continuously updated as the client/server conversation progresses.

If you do not see any activity in this window as you record, Robot is not capturing client/server traffic. Stop recording and try to determine the cause of the problem.

The Session Recorder During Recording

The following figure shows the Session Recorder window as it might appear during recording, and the type of information that it displays:


Number of calls and bytes in the current 3-second interval
Total number of calls and bytes in the recording session
- MYSESSION - Virtual User Recorder - API recording in progress
Rates
Calls in interval:
0
41
Bytes in interval:
0
4340
Statistics
Interval
Total
Calls
Bytes
Progress bars indicating the number of calls and bytes in the current 3-second interval
Most calls in any 3-second interval
Most bytes in any 3-second interval
Calls / Annotations
Ready
The annotations (comments, start/stop blocks, timers, synchronization points) you insert during recording

	Interval	Total
Calls	0	118
Bytes	0	10844

```
Send[00000000]:00000352:
Recv:00000472:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:22 GMT--Content-Typ
Send[00000000]:00000352:
Close:00000472
Connect:00000492:149.174.213.62, 00000080
Send[00000000]:00000492:GET /sprynet/ioven/toc.htm HTTP/1.1--Accept: application/vnd.ms-excel, application/msword, image/
Send[00000000]:00000352:
Connect:00000516:149.174.213.62, 00000080
Send[00000000]:00000516:GET /sprynet/ioven/hall.htm HTTP/1.1--Accept: application/vnd.ms-excel, application/msword, image/
Send[00000000]:00000352:
Recv:00000492:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:23 GMT--Content-Typ
Recv:00000492:arget=framePanel></a> -->--<a href=room4.htm target=framePanel> Kitchen</a>--<br><br>--<a href=room5.htm targ
Send[00000000]:00000352:
Recv:00000516:HTTP/1.0 200 OK--Server: Spry-SafetyWEB-Server-NT/1.2a--Date: Thu, 05 Feb 1998 17:33:23 GMT--Content-Typ
Recv:00000516:indow or use the table of contents in the left panel.<b> We hope you enjoy your visit.</i></b>--<a href=mailto:iovem
Send[00000000]:00000352:
Close:00000516
Close:00000492
```

The activity that the window displays varies, depending on your recording method.

- With API recording, the window displays the number of API calls and bytes sent from your computer.
- With network recording, the window displays the number of IP packets and the bytes in these packets. However, the information may not be from your computer only. For example, if you are recording the activities of any client, in the **Client** list, the Session Recorder window reports the activity of *all* clients on the network, not just the activity of your computer.
- With proxy recording, the window displays the number of IP packets and the bytes in these packets.

The Session Recorder Icon

During recording, the icon associated with the Session Recorder window is displayed on the taskbar. The icon blinks whenever Robot is capturing a request or response. This icon serves as a visual cue that Robot is recording, even when the Session Recorder window is minimized.

The Session Recorder After Recording

The Session Recorder captures "raw" client API calls or network IP packets — in other words, the calls or packets as they exist before Robot converts them into appropriate scripting language commands.

When recording ends, Robot stores the calls or packets as follows:

- Robot stores the calls or packets in a session file (.wch) in the same raw form that the packets have when captured.
- Robot translates the calls or packets into VU commands and stores them in one or more .s script files.

NOTE: At any time, you can regenerate new script files from the stored .wch file. For more information, see *Regenerating Scripts from a Session* on page 7-17.

Cancelling Scripts During Recording

During session recording, you can cancel scripts that you have generated. The scripts are deleted.

This feature is useful if you make errors while recording a session, or if you want to exclude non-essential or preliminary activity (such as logging in or navigating to the Web site that you want to test). For example, if you split a script at the point where you want to send a query, you can ignore the login and other preliminary requests you needed to make to get to the query's starting point.

Canceling a Script in a Single-Script Session

If you have not split the session into multiple scripts, you can cancel both the script and the session, and then stop recording as follows:

1. During recording, click the **Stop** button on the Session Record floating toolbar.
2. In the Stop Recording dialog box, click **Ignore just-recorded information**.
3. Click **OK** in the Stop Recording dialog box.
4. Click **OK** to acknowledge that the session is being deleted.

Canceling the Current Script in a Multi-Script Session

When you record a session, you click the **Split Script** button to create multiple scripts.

To cancel the current script, keep the other scripts that you recorded in this session, and then continue recording:

1. During recording, click the **Split Script** button on the Session Record floating toolbar.
2. In the Split Script dialog box, click **Ignore just-recorded information**.
3. Click **OK**.

You can now begin recording a new script.

To cancel the current script, keep the other scripts that you recorded in this session, and then stop recording:

1. During recording, click the **Stop** button on the Session Record floating toolbar.
2. In the Stop Recording dialog box, click **Ignore just-recorded information**.
3. Click **OK**.

4. Click **OK** in the Generating Scripts dialog box (after Robot finishes generating the script).

Canceling All Scripts in a Multi-Script Session

To cancel all of the scripts in a session and stop recording:

1. During recording, click the **Stop** button on the Session Record floating toolbar.
2. Click **OK** in the Stop Recording dialog box.
3. Immediately click **Cancel** in the Generating Scripts dialog box.
4. Optionally, delete the session and empty scripts that Robot generated.

   You probably want to keep a script if you have planned a script in TestManager and defined properties for it. You can later record over the script and retain the properties that you have defined.

For information about deleting scripts and sessions, see *Deleting Scripts and Sessions* on page 7-25.

Choosing the Protocols to Include in a Script

During network and proxy recording (and to a lesser extent, during API recording), Robot might capture requests for protocols that you do not want to include in a script. You can specify the protocols to include in either of the following ways:

- Automatically, by selecting **Auto Filtering** in the **Generator Filtering** tab of the Session Record Options dialog box.
- Manually, by selecting **Manual Filtering** in the **Generator Filtering** tab. If you select this check box, Robot displays the Manual Filtering dialog box during script generation, immediately after recording. The following section describes how to filter protocols manually.

Manually Filtering Protocols

The Manual Filtering dialog box lists in a hierarchical tree the connections that Robot detected during the recording session. In this dialog box, a **connection** has three parts:

- The name or IP address of a client
- The name or IP address of the server that communicated with the client during the connection
- The protocol of the captured requests and responses issued during the connection

Use this dialog box to select the protocols to include in the script. You select the protocols to include by adding and removing the connections listed in the dialog box. Because you are selecting protocols within the context of a connection, you select protocols in one or more of these ways:

- You can select the protocol used in all the connections to a particular server.
- You can select the protocols used in all the connections from a particular client.
- You can select a particular protocol name, regardless of the clients and servers that use it.

Controls in the Manual Filtering Dialog Box

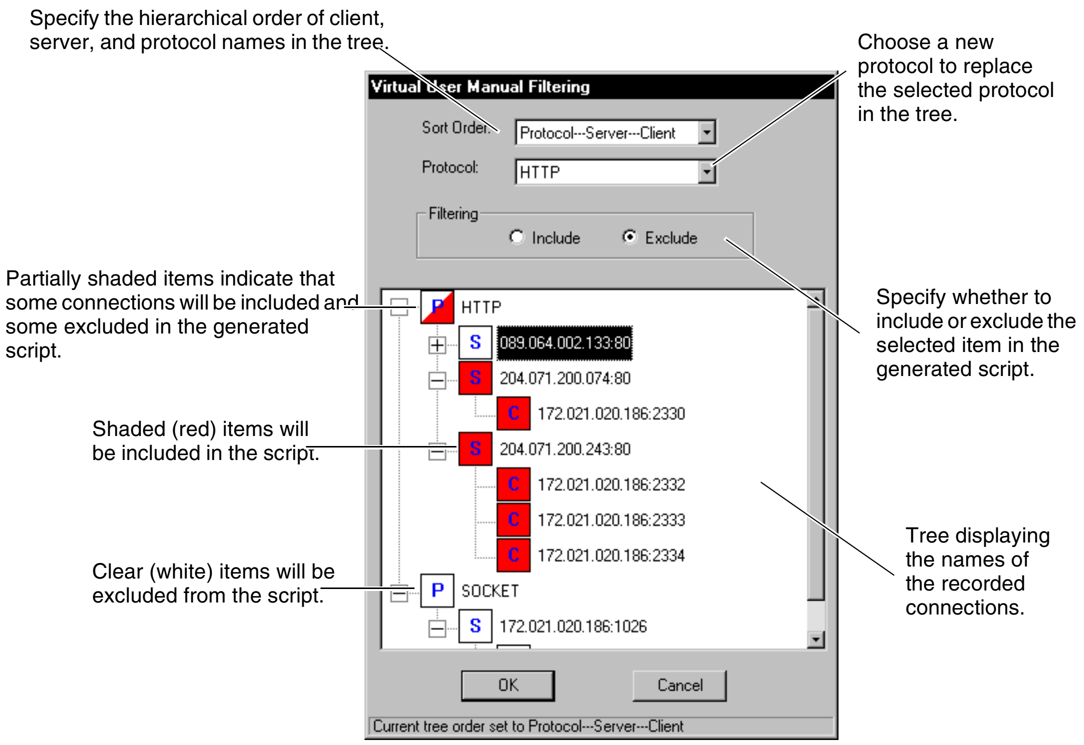
The Manual Filtering dialog box has the following controls:

Control name	Purpose
Sort Order	Changes the hierarchical order in which protocol, client, and server names are listed in the tree.
Protocol	Changes the type of requests that Robot generates for the currently selected connection in the tree. Robot converts the protocol in the current connection to the protocol type that you specify in **Protocol**. Typically, you will not want to convert captured protocols.
Include, **Exclude**	Includes or excludes selected items in the script that Robot is generating.

Control name	Purpose
Tree P Protocol S Server C Client	Displays the protocol, client, and server names for conversations that Robot captured during recording. Also indicates whether items are marked for inclusion in or exclusion from the script. ▶ Items are marked for inclusion or exclusion as follows: ▶ Items that Robot will generate to the script are shaded red. ▶ Items that Robot will exclude from the script are clear (white). If an item is partially shaded red, some of the items below will be included, and some will be excluded. When you click an item in the tree, you select that item and also any items below that item. Click the + icon to expand a branch, and click the - icon to close a branch. To change the hierarchical order of the protocol, client, and server names, select a new order in **Sort Order**. Robot attempts to detect and display the names of the clients and servers involved in the conversations. If the names cannot be resolved, Robot displays IP addresses.
OK, **Cancel**	Confirms or cancels any changes that you have made in this dialog box, and then closes the dialog box. When you close the dialog box, requests represented by shaded items are generated to the script.

Example of Manually Filtering Protocols

The following figure shows the Manual Filtering dialog box, with the components of the connection sorted by protocol, then server, then client:


Specify the hierarchical order of client, server, and protocol names in the tree.
Choose a new protocol to replace the selected protocol in the tree.
Partially shaded items indicate that some connections will be included and some excluded in the generated script.
Specify whether to include or exclude the selected item in the generated script.
Shaded (red) items will be included in the script.
Tree displaying the names of the recorded connections.
Clear (white) items will be excluded from the script.
Virtual User Manual Filtering
Filtering
OK
Cancel

Sort Order: Protocol---Server---Client

Protocol: HTTP

Filtering: Include [✓ Exclude]

- P HTTP
  - S 089.064.002.133:80
  - S 204.071.200.074:80
    - C 172.021.020.186:2330
  - S 204.071.200.243:80
    - C 172.021.020.186:2332
    - C 172.021.020.186:2333
    - C 172.021.020.186:2334
- P SOCKET
  - S 172.021.020.186:1026

Current tree order set to Protocol---Server---Client

In this example, Robot will generate HTTP requests for the following connections:

- All connections to the server 204.071.200.243:80
- The single connection to the server 204.071.200.074:80

No requests will be generated for the following items:

- All connections to the server 089.064.002.133:80
- All socket protocols

Including or Excluding Connections

In the Manual Filtering dialog box tree, each top-level item expands to display the components of one or more connections. Connection components can appear in the tree in any hierarchical order, depending on the **Sort Order** setting.

If a top-level item is marked for inclusion in the script that is being generated, all requests associated with that item (such as all HTTP connections) are included in the script. However, you can then selectively exclude one or more of the individual connections.

By selecting items to include and exclude, you can:

- Include or exclude all requests associated with a protocol, or just some of those requests (by including or excluding client or server items below it).
- Include or exclude all requests to a particular server, or just some of those requests (by including or excluding protocol or client items below it).
- Include or exclude all requests from a particular client, or just some of those requests (by including or excluding protocol or server items below it).

To include or exclude the requests associated with an item in the tree:

1. Click an item to include or exclude. Any items hierarchically below it are also selected.
2. Click **Include** or **Exclude**.
3. Repeat the above steps until all items to include in the script are shaded in red, and all items to exclude are clear (white).
4. Click **OK**.

Converting from One Protocol Type to Another

You can also use the Manual Filtering dialog box to convert the requests captured during a connection from one protocol type to another.

To convert a protocol in the Manual Filtering tree:

1. Click the item of the tree representing the protocol to convert.
2. In the **Protocol** box, select the new protocol.
3. Click **OK**.

Some requests may be lost in a protocol conversion.

Typically, you will not want to convert protocol requests. But if you need to convert, you will most likely convert to or from socket requests.

Socket requests are low-level requests that are typically issued in addition to requests made with other, higher-level protocols (such Oracle or SQL Server). As a result, you can specify that a captured protocol be converted to its associated socket requests, or that captured socket requests be converted to the associated requests in a higher-level protocol.

Playing Back a Script Quickly

After you record a script, you generally play it back from a suite, as part of a user group. However, if you want to test a script that you have just recorded or edited, you can play it back quickly.

To play back a script quickly:

1. In Robot, click **File** → **Playback**.
2. Click the name of the script to play back.
3. Click **OK**.

   TestManager appears, ready to play back the script that you selected.
4. In TestManager, click **Run** → **Suite**.
5. Click **OK** in the Run Suite dialog box.

Working with Sessions

A performance recording session contains all of the client requests and server responses issued from the time you begin recording until the time you stop recording.

When you work with sessions, you can:

- Split the session into multiple scripts.
- Regenerate the scripts from the session.
- View the session's properties.

The following sections describe each of these activities.

Splitting a Session into Multiple Scripts

Splitting a session signifies that everything you have recorded represents one logical unit of work, such as a login to a database. When you split a session, you name the completed script and start a new script. You can continue recording transactions and splitting the session into as many scripts as you want.

NOTE: If you split a session into multiple scripts, you should examine the resulting scripts to make sure that they begin and end at a known state. This is particularly important if you plan to use a split script as part of a loop or to run a series of scripts in a different order than you recorded them. Check the state of connections used in the script and any `sqlprepare` emulation commands or VU commands that declare or manipulate cursors.

How to Split a Session into Multiple Scripts

To split a session into multiple scripts:

1. During recording, at the point where you want to end one script and begin a new one, click the **Split Script** button on the Session Record floating toolbar.
2. Enter a name for the script that you are ending, or accept the default name.

   You will specify a name for the script that you are about to begin when you finish recording client requests for it.

   Alternatively, to *cancel* the requests you made since you began recording the current script, click **Ignore just-recorded information**. This action affects only the current script, not any previous scripts you recorded in this session. For more information, see *Cancelling Scripts During Recording* on page 7-9.
3. Click **OK**.
4. Repeat the previous steps as many times as needed to end one script and begin another.
5. After you click the **Stop Recording** button to end the recording session, type or select a name for the last script you recorded, or accept the default name.

Importing a Session

You can import a session from a different computer into your current project. For example, assume someone at another site e-mails you a session file. You can import this session file, regenerate scripts, and create a suite.

To import a session file and regenerate scripts:

1. In Robot, click **Tools** → **Import Session**. The Open dialog box appears.
2. Click the session file, then click **Open**. The session and its scripts are now in your project.
3. To regenerate the scripts in the session you imported, click **Tools** → **Regenerate Test Scripts from Session**, and select the session you imported.
4. To regenerate the suite, click **Tools** → **Rational Suite TestStudio** → **Rational TestManager**.
5. Click **File** → **New Suite**. The New Suite dialog box appears.
6. Select **Existing Session**, and click **OK**.
7. TestManager displays a list of sessions that are in the project. Click the name of the session that you imported, and click **OK**.

TestManager automatically creates a suite that is ready to run.

NOTE: To import a session from one project to another, click **Tools** → **Rational Test** → **Rational TestManager**. Then click **File** → **Import Test Assets**.

Regenerating Scripts from a Session

You might want to regenerate a session's scripts for a variety of reasons—for example, to overwrite edits you made to the original scripts (restoring the scripts to the original recorded transactions), or to change the script generation options.

When you regenerate scripts from a session, the regenerated scripts inherit the properties of the original scripts.

At any time, you can regenerate a session's scripts from the session file. To do so:

1. In Robot, click **Tools** → **Regenerate Test Scripts from Session**.
2. Click the name of the session to use.

   You can regenerate scripts that are contained within a session but not those that have been deleted from the session. To see the scripts in a session, click the session name, click **Properties**, and then click **Contained Scripts.**

3. Click **OK**.

   The Generating Scripts dialog box appears. This dialog box shows how script regeneration is progressing. After a few seconds (or longer, depending on the length of the session), regeneration ends, the message *Completed successfully* appears, and the **OK** button is enabled.

4. Click **OK** to acknowledge that the regeneration operation is complete.

Changing Recording Options

When you regenerate a session's scripts, you can change many of the recording options that were set when the script was recorded. You can change the options in the following Session Record Options tabs:

- Generator
- Generator Filtering
- Generator per Protocol

For example, you can:

- Select **Use datapools** to add datapool commands to the script, even if the original script had no datapool commands generated for it. (Conversely, clear this check box to have no datapool commands included in the new script.)
- Select a different **Display returned data** value than the one used in the original script.
- Set different playback expectations than those used in the original script

When you regenerate scripts, you cannot add client/server requests to those that you originally recorded. However, you can remove some recorded requests through protocol filtering.

NOTE: When you regenerate a session's scripts, you overwrite all of the scripts in the session.

For example, to change options in the **Generator** tab:

1. Click **Tools** → **Session Record Options.**
2. Click the **Generator** tab.
3. Specify the script options to include in the new script, and then click **OK**.
4. Click **Tools** → **Regenerate Test Scripts from Session** to regenerate the script.

Viewing Session Properties

Session properties include the list of scripts in the session and a description of the session.

While viewing a session's properties, the only session property that you can modify is its description. Other session properties are automatically defined when you create the session.

To view and optionally modify session properties while you are regenerating a session's scripts in Robot:

1. Click **Tools** → **Regenerate Test Scripts from Session**.
2. Click the name of the session whose properties you want to view.
3. Click **Properties**.
4. When finished, click **OK** to save any changes, or click **Cancel**.
5. In the Regenerate Test Scripts from Session dialog box, click **Cancel**.

NOTE: If you click **OK** in the Regenerate Test Scripts from Session dialog box, the existing scripts in the session are destroyed. If you then click **Cancel** in the Generating Scripts dialog box before the scripts are regenerated, Robot will generate empty scripts.

Accessing Script Properties from Session Properties

While you are viewing a session's properties, you can view and optionally modify the properties of any script generated from the session.

To view script properties:

1. In Robot, click **Tools** → **Regenerate Test Scripts from Session**.
2. Click the name of the session whose properties you want to view. Session names are the same as session file names, but without the.wch extension.
3. Click **Properties**.
4. Click the **Contained Scripts** tab.
5. Select the script whose properties you want to view or modify.
6. Click **Properties**. The Script Properties dialog box appears.
7. When you have finished viewing and editing properties, click **OK** to save any changes in the **Script Properties** dialog box, or click **Cancel**.
8. Click **Cancel** to close the Session Properties dialog box.

9. In the Regenerate Test Scripts from Session dialog box, click **Cancel**.

Coding a Script Manually

The fastest and easiest way to generate a script is to record a session with Robot and generate the script automatically.

However, you can open an empty script and add code to it—for example, if you are hand-coding the script, or if you are copying code from another script.

To open an empty script and add code to it:

1. In Robot, click **File** → **New Test Script**, then choose the type of script to create.
2. Type a script name and, optionally, a description of the script.
3. Click **OK**. Robot creates an empty script with the appropriate scripting language headers.
4. Add the code to the script.

Creating Library Files for VU Scripts

Scripting language libraries are packaged in DLLs. You create dynamic link library (DLL) files using a development tool such as Microsoft Visual Studio. For information about making the DLLs that you create available to VU scripts, see the *VU Language Reference*.

Defining Script Properties

A script can have properties associated with it in addition to the script name. Examples of script properties include a description of the script, the purpose of the script, and any test requirements associated with the script.

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. But you can also define a script's properties after you record the script, as described in the following section.

How to Define Script Properties in Robot

To define properties for a script that is open for editing in Robot, click **File** → **Properties**.

If the script exists but is not open:

1. Click **File** → **Open** → **Test Script** to open the Open Test Script dialog box.
2. Click the script you are defining properties for.
3. Click **Properties**.
4. Define the script's properties, and then click **OK**.

For information about the properties that you can define, see the section about customizing scripts and TestManager suites in the *Using Rational Robot* manual.

Managing Scripts and Sessions

This section describes the following script and session management activities:

- Finding the scripts contained in a session
- Finding the session name associated with a script
- Removing a script from a session
- Re-recording sessions
- Re-recording scripts
- Copying scripts
- Deleting scripts and sessions

Finding the Scripts Contained in a Session

To see a list of all the scripts contained in a session:

1. In TestManager, click **View** → **Test Asset Workspace**.
2. Double-click **All** under **Session Queries**.
3. Double-click the name of the session whose script names you want to view.
4. Click **Contained Test Scripts**.

Optionally, you can open scripts or view script properties.

Finding the Session Associated with a Script

A script can be associated with only one session. To see the name of this session:

1. In Robot, click **File** → **Open** → **Test Script**.
2. Click the name of the script whose associated session you want to view.
3. Click **Properties**.
4. Click **Related Assets**.
5. View the session name in **Referenced Session**.

A script might not be associated with a session. For example, a script might have been removed from its session, as described in the next section.

Removing a Script from a Session

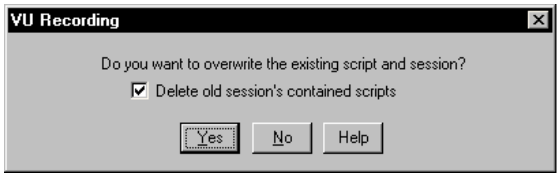
If you remove a script from a session, you can no longer regenerate that script if you regenerate the session.

To remove a script from a session:

1. In Robot, click **File** → **Open** → **Test Script**.
2. Click the name of the script to remove from its session.
3. Click **Properties**.
4. Click **Related Assets**.
5. View the session name in **Referenced Session**.
6. Click **Clear**.

Re-Recording Sessions

When you begin to record over a session that contains scripts, Robot prompts you for a confirmation. In the same dialog box, Robot also prompts you for a disposition of the scripts in the session, as follows:


VU Recording

Do you want to overwrite the existing script and session?

☑ Delete old session's contained scripts

Whether you select or clear the check box depends on what you want to do:

- Delete all of the session's scripts and their associated properties, and begin re-recording the session.
- Keep the original scripts and their properties while creating new scripts for the session.
- Overwrite the original scripts, but assign their properties to the new scripts.

The following sections describe each action. Regardless of which action you take, the original session and its properties are overwritten.

Deleting the Original Scripts and Properties

To re-record a session and delete the original scripts and their properties:

1. Click **File** → **Record Session**.
2. In the Record Session - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the Session Recording dialog box, select **Delete old session's contained scripts**, then click **Yes**.

   The session's contained scripts and their properties are deleted.
4. Continue re-recording the session, assigning any names you like to the scripts that you are recording.

Keeping the Original Scripts

To re-record a session and create new scripts while keeping the original scripts and their properties intact:

1. Click **File** → **Record Session**.
2. In the Record Session - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the Session Recording dialog box, clear **Delete old session's contained scripts,** and then click **Yes**.
4. Continue re-recording the session, assigning any names you like to the scripts that you are recording *other than the names of the original scripts*.

The original scripts will no longer be associated with this or any other session. However, you can still add the original scripts to a suite.

Overwriting the Original Scripts but Keeping Their Properties

To re-record a session and overwrite the original scripts while assigning the properties of the original scripts to the new scripts:

1. Click **File** → **Record Session**.
2. In the Record Session - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the Session Recording dialog box, clear **Delete old session's contained scripts, and** then click **Yes**.
4. Continue re-recording the session, assigning the name of *one of the original scripts* to each script that you record.

Re-Recording Scripts

Recording over a session affects all scripts in the session. To record over just one script, simply select that script's name when Robot prompts you for a script name during recording (in the Split Test Script or Stop Recording dialog box).

Also, if you plan a script in TestManager, its name appears in the list that you can choose from when you record a script.

The following table summarizes the events that take place when you select the name of a planned or existing script rather than type a new name for a script that you have just recorded:

Type of script	**Result of overwriting the script**
Planned script	The script's properties are applied to the new script. Robot does not prompt for a confirmation before recording the script because the existing script is empty.
Existing script is part of a session	Robot prompts for a confirmation that you want to overwrite the script: ▶ Click **No** to select or type another script name. ▶ Click **Yes** to overwrite the script. The properties of the original script are applied to the new script. Also, the script is removed from the original session and added to the new session.
Existing script is not part of a session	Robot overwrites the original script without prompting you for a confirmation. The properties of the original script are applied to the new script.

Copying Scripts

To copy a script in Robot:

1. Click **File** → **Open** → **Test Script**.
2. Click the name of the script to copy, and then click **OK**.
3. Click **File** → **Save As**.
4. Type a name for the new script, and then click **OK**.

The new script does not retain the properties of the original and is not associated with any session.

Deleting Scripts and Sessions

To delete a script and its properties:

1. In Robot, click **File** → **Delete**.
2. Click the name of the script to delete.

   To delete multiple scripts, hold down the CTRL key and click each script.
3. Click **OK**.
4. Click **OK** when prompted to confirm the deletion.
5. Click **Cancel** to close the Delete Script dialog box.

If you delete a script from a session, you can no longer regenerate that script if you regenerate the session that it was once associated with.

If you delete all scripts in a session, the session still remains.

To delete a session:

1. In Robot, click **File** → **Delete** → **Delete Session.**
2. Click the name of the session to delete, and then click **Delete**.
3. Click **Contained Test Scripts**.
4. When prompted to confirm the deletion, select or clear the **Delete scripts contained in the session?** check box as follows:
   – Select the check box to delete all of the session's scripts and properties in addition to deleting the session.
   – Clear the check box to leave the session's scripts and properties intact. The scripts will no longer be associated with this or any other session. However, you can still add the scripts to a suite.

5. Click **Yes** to confirm the deletion.

▸ ▸ ▸ C H A P T E R 8

Adding Features to Scripts

This chapter describes the features that you can add to a script while recording the session with Robot. The chapter includes the following topics:

- Timers
- Blocks
- Synchronization points
- Comments
- Using the Insert menu

Timers

Individual emulation commands (such as `sqlprepare` and `sqlexec`) are timed automatically. By default, these times are included in TestManager report output.

However, if you want to measures the time it takes a virtual tester to perform an activity—for example, sending a query to the server and displaying the results—you insert a timer or a block in the script.

How Timers Work

Think of a timer as a stopwatch that you click on just before you begin to perform the timed activity, and that you click off when you complete the activity.

For example, suppose you want to time how long it takes to submit a query to a database server and receive the results. During recording, you would:

1. Start the timer (click **Insert** → **Start Timer**) just before you click the button to send the query. This action inserts the VU emulation command `start_time` into the script.
2. Stop the timer (click **Insert** → **Stop Timer**) as soon as the results appear. This action inserts the VU emulation command `stop_time` into the script.

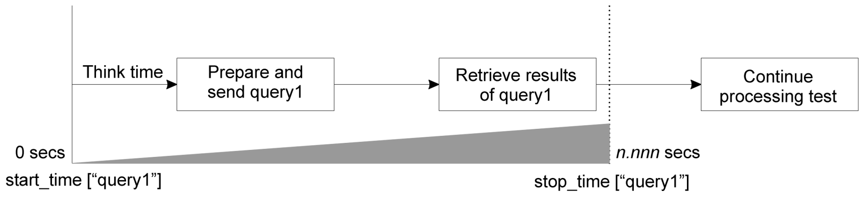
When you stop a timer, you can reuse that timer's name in another timer. There is no practical limit to the number of timers that you can add to a script.

You can nest timers within other timers (by starting and stopping the second timer before stopping the first timer), and you can overlap timers (by stopping the second timer after stopping the first timer).

If you do not explicitly stop a timer, no response time is reported for that activity.

You cannot extend a timer over multiple scripts.

The following illustration shows the `start_time` and `stop_time` emulation commands for a timer named query1:



Why Use Timers?

Use timers in the following cases:

- Time an overlapping sequence of events. You can insert a `start_time` command followed by several `stop_time` commands. You cannot overlap blocks (although you can nest them).
- Time a very specific portion of the script. You can insert the `start_time` and `stop_time` commands exactly where you want when you edit the script. You can insert a block, however, only during recording.

In other cases, however, you may want to use blocks rather than timers. Blocks not only add timers to a script, but they also add a prefix to each command ID in the block. This prefix enables you to easily identify emulation commands associated with a block both in the script and in the report output.

Adding a Timer During Recording

During recording, you can add a timer operation to a script as follows:

1. If the Session Insert floating toolbar is not already displayed, click the **Display Session Insert Toolbar** button on the Session Record floating toolbar.
2. Click the **Start Timer** button.

3. In the Start Timer dialog box, type the timer's name (40 characters maximum), and then click **OK**.
4. Perform the timed activity.
5. Immediately after receiving the results generated by the activity, click the **Stop Timer** button on the Session Insert floating toolbar.
6. In the Stop Timer dialog box, select the name of the timer you typed in step 3, and then click **OK**.

When you start and stop a timer during recording, you can view these commands in the Annotations window.

Adding a Timer During Editing

The Session Insert toolbar (or the Robot Insert menu) adds timers during recording. To add a timer during editing, type the timer commands into the script.

The following are the timer commands for VU scripts:

- `start_time` – Starts timing the activity. Insert this command immediately before the first emulation command for the activity that you are timing. The `start_time` measurement includes the think time (if any) for the next emulation command in the script.

  To exclude the think time for an emulation command, insert `start_time` after the emulation command and use the `_fs_ts` read-only variable. For example:

  ```
  http_request ["test1.001"] ...
  start_time ["timerid"] _fs_ts;
  stop_time ["timerid"];
  ```

- `stop_time` – Stops timing the activity. Insert this command immediately after the last emulation command for the activity that you are timing.

For information on using timers in VU scripts, see the `start_time` command in the *VU Language Reference*.

For information on using timers in Visual Basic scripts, see the `TimerStart` method in *Rational Test Script Services for Visual Basic*.

For information on using timers in SQABasic scripts, see the `StartTimer` command in *SQABasic Language Reference*.

Blocks

A *block* is a set of contiguous lines of code that you want to make distinct from the rest of the script. Typically, you use a block to identify a transaction within a script.

A block has the following characteristics:

- A block begins with the a comment. In the VU language, a block begins like this:

  ```
  /* Start_Block "BlockName" */
  ```

- Robot automatically starts a timer at the start of the block. In the VU language, the timer looks like this:

  ```
  start_time ["BlockName"] _fs_ts;
  ```

  Typically, the `start_time` emulation command is inserted after the first action, but with an argument to use a read-only variable that refers to the start of the first action.

- The ID of every emulation command in a block is constructed the same way—that is, by the block name followed by a unique, three-digit autonumber. For example, in the VU language:

  ```
  http_header_recv ["BlockName002"]  200;
  ```

  When you end a block, command IDs are constructed as they were before you started the block. For example, if the last command ID before the block was `Script025`, the next command ID after the block will be `Script026`.

- A block ends with a `stop_time` command plus a comment.For example, in the VU language:

  ```
  stop_time ["BlockName"]; /* Stop_Block */
  ```

A script can have up to 50 blocks.

When you end a block, Robot automatically ends the current block. In other words, blocks can be nested, but they cannot be overlapped. For example:

Valid blocks	**Invalid blocks**
Block1Start	Block1Start
Block2Start	Block2Start
Block2Stop	Block1Stop
Block1Stop	Block2Stop

You cannot extend a block over multiple scripts. If you attempt to split a script in the middle of a block, Robot ends the block when it ends the initial script.

Why Use Blocks?

You might want to use blocks for the following reasons:

- To associate the block and timer names with the emulation command that performs the transaction.
- To include the block name in TestManager reports, thus enabling you to filter the reports with the block name.
- To make the script easier to read, and to provide an immediate context for a line within the block through command IDs.

Adding a Block

To insert a block into a script:

1. If the Session Insert floating toolbar is not already displayed, click the **Insert** button on the Session Record floating toolbar.
2. Click the **Start Block** button at that point in the script where you want the block to begin—for example, just before you start to record a transaction.
3. Type the block name.

   Robot uses this name as the prefix for all command IDs in the block. The maximum number of characters for a command ID prefix is seven.
4. Click **OK**.
5. Record all of the client requests in the block.
6. Click the **Stop Block** button to end the current block, and then click **OK**.

NOTE: When you end a block, you always end the current block. If you are nesting blocks, you cannot specify which block you want to end—the **Stop Block** command always applies to the innermost block. For more information, see *Nesting Blocks* on page 8-6.

7. Continue recording the other sections of the script.

When you start and stop a block during recording, the commands are reported as annotations in the Annotations window.

NOTE: You can add a block only during recording, not during editing.

Nesting Blocks

To nest blocks, click **Start Block** on the Session Insert floating toolbar to start a new block without explicitly ending the current block.

When you nest blocks:

- Robot automatically starts a timer at the at or near the beginning of the second block.
- Timing continues on the first block (in other words, a stop timer command is not inserted for the first block).
- The second block's name replaces the first block's name as the prefix for emulation commands.

If you have nested blocks and you click **Stop Block**:

- Robot inserts a stop timer command to stop timing the current block.
- The next block up in the hierarchy becomes the current block (that is, its name is used as the prefix for emulation commands). Timing continues on this block plus other blocks that may be above it in the nesting hierarchy.

Example of Nested Blocks

The following VU language example contains three blocks—`blockA`, `blockB`, and `blockC`:

```
/* blockA begins with a Start Block command */
/* Start_Block "blockA" */
start_time ["blockA"];
...          /* Perform transaction in blockA */
http_nrecv ["blockA022"]  100 %% ;      /* 411/8147 bytes */
http_disconnect(img4_yahoo_com_80_5);

/* blockB begins with a second Start Block command */
/* Start_Block "blockB" */
start_time ["blockB"];
/* Perform transaction in blockB */
http_nrecv ["blockB012"]  100 %% ;      /* 5812 bytes */
http_disconnect(D141_217_90_3_80);

/* blockC begins with a third Start Block command */
/* Start_Block "blockC" */
start_time ["blockC"];
/* Perform transaction in blockC */
http_nrecv ["blockC054"]  100 %% ;      /* 4577 bytes */
http_disconnect(D141_217_90_3_80_17);

/* A Stop Block command ends the current block (blockC) */
stop_time ["blockC"]; /* Stop_Block */
moe_si_umich_edu_80 = http_request ["blockB013"] ...;
...                /* Resume blockB transaction */
http_nrecv ["blockB018"]  100 %% ;      /* 5076 bytes */
http_disconnect(moe_si_umich_edu_80);


/* A second Stop Block command ends the current block (blockB) */
```

```
stop_time ["blockB"]; /* Stop_Block */
ntdwwaag_v1_compuserve_com_80_38 = http_request ["BlockA023"]...;
...                /* Resume blockA transaction */
http_nrecv ["BlockA031"]  100 %% ;      /* 3791/3787 bytes */
http_disconnect(ntdwwaag_v1_compuserve_com_80_38);

/* A third Stop Block command ends the current block (blockA) */
stop_time ["blockA"]; /* Stop_Block */
```

Synchronization Points

A **synchronization point** lets you coordinate the activities of a number of virtual testers by pausing the execution of each tester at a particular point—the synchronization point—until one of the following events occur:

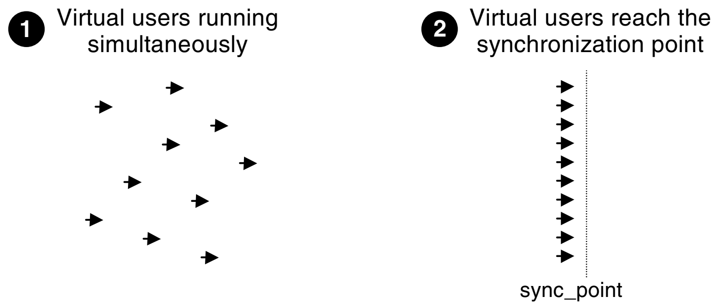
- All virtual testers associated with the synchronization point arrive at the synchronization point.
- A timeout period is reached before all virtual testers arrive at the synchronization point. You specify the timeout period in the TestManager Synchronization Point dialog box.
- You manually release the virtual testers while monitoring a suite run in TestManager.

When one of the above events occurs, TestManager releases the virtual testers, allowing them to continue performing the transaction.

How Synchronization Points Work

At the start of a test, all virtual testers begin executing their assigned scripts. They continue to run until they reach the synchronization point. In a script, a synchronization point is the command **sync_point** (VU script) or **SQASyncPointWait** (SQABasic script). In a suite, a synchronization point is a time specified in the suite.

The following figure illustrates a synchronization point in a script:


1 Virtual users running simultaneously
2 Virtual users reach the synchronization point
sync_point

The virtual testers pause at the synchronization point until TestManager releases them. Typically, TestManager releases synchronized virtual testers when they all arrive at the synchronization point.

Why Use Synchronization Points?

By synchronizing virtual testers to perform the same activity at the same time, you can make that activity occur at some particular point of interest in your test—for example, when the application-under-test sends a query to the server.

Typically, synchronization points that you insert into scripts are used in conjunction with timers to determine the effect of varying workload on the timed activity. For example, to the effect of workload on data retrieval, you could take the following general steps:

1. While recording the script (named VU1 in this example) that will submit the query and display the result, perform the following actions:
   - **a.** Insert a synchronization point named `TestQuery` into the script.
   - **b.** Click the **Start Block** button (see page *Adding a Block* on page 8-5).

     The block times the transaction you are about to perform. The block also associates the block and timer names with the name of the emulation command that performs the transaction.
   - **c.** Submit the query and wait for the results to be displayed.
   - **d.** Click the **Stop Block** button.
2. While recording the virtual tester that will provide the workload, insert another `TestQuery` synchronization point just before you begin to record the activity that provides the load—for example, just before you click the button to submit an order form. Name this script VU2.
3. Add VU1 and VU2 to a suite.

4. Run the suite a number of times, each time using a different number of the VU2 virtual testers. However, you only need one VU1 user in each test.

   Theoretically, as the number of synchronized VU2 virtual testers increases, the time reported by the VU1 timer should also increase.

In this example, the `TestQuery` synchronization point ensures that:

- All VU2 virtual testers submit their forms at the same time—thereby providing maximum concurrent workload.
- The VU1 virtual tester submits its query at the same time that the VU2 virtual testers are loading the server—thereby providing maximum workload at a critical time.

Inserting Synchronization Points

You can insert a synchronization point into a script (through Robot) or into a suite (through TestManager).

- **Into a script** – You can insert a synchronization point into a script in one of the following ways:
  - During recording, through the **Sync Point** toolbar button or through the Insert menu.
  - During script editing, by manually typing the synchronization point command name into the script.

Insert a synchronization point into the script to control exactly where the script pauses execution. For example, you can insert a synchronization point command just before you send a request to a server.

You should also use this method if the synchronization point will depend upon some logic that you add to the script during editing.

- **Into a suite** – You can insert a synchronization point into a suite through the TestManager Synchronization Point dialog box.

  Insert a synchronization point into the suite to pause execution before or between scripts rather than within a script. In addition, inserting a synchronization point into a suite offers these advantages:
  - You can easily move the location of the synchronization point without having to edit a script.
  - The synchronization point is visible within the suite rather than hidden within a script.

A script can have multiple synchronization points, each with a unique name. The same is true of a suite. A given synchronization point name can be referenced in multiple scripts and/or suites.

The following sections describe the various ways to insert synchronization points.

Inserting a Synchronization Point During Recording

To insert a synchronization point into a script during recording:

1. If the Session Insert floating toolbar is not already displayed, click the **Insert** button on the Session Record floating toolbar.
2. Click the **Sync Point** button immediately before you begin to record the activity that you are synchronizing.

   For example, to synchronize multiple virtual testers so that they all submit a query at the same time, first insert the synchronization point, and then perform the user action that sends the query to the server.
3. Type the synchronization point name.
4. Click **OK**.

When you insert a synchronization point during recording, the command is reported as an annotation in the Annotations window.

Inserting a Synchronization Point During Editing

You can only use the Session Insert toolbar (or the Robot Insert menu) to insert a synchronization point into a script during recording. To insert a synchronization point during editing, type a synchronization point command into a script.

For information on inserting synchronization points in VU scripts, see the `sync_point` command in the *VU Language Reference*.

For information on inserting synchronization points in Visual Basic scripts, see the `SyncPoint` method in *Rational Test Script Services for Visual Basic*.

For information on inserting synchronization points in SQABasic scripts, see the `SQASyncPointWait` command in *SQABasic Language Reference*.

Inserting a Synchronization Point into a Suite

When you insert a synchronization point into a suite, you can do more than simply assign a synchronization point name to a script. For example:

- You can specify whether you want the virtual testers to be released at the same time or at different times.

If the virtual testers are to be released at different times (that is, in a staggered fashion), you can specify the minimum and maximum times within which all virtual testers must be released.

- You can specify a timeout period.

For more information about inserting a synchronization point into a suite, see *Using Rational TestManager*.

Release Times and Timeouts for Synchronization Points in Test Scripts

You cannot define minimum and maximum release times or timeout periods for synchronization points that you insert into scripts (as you can for synchronization points that you insert into suites). By default:

- Virtual testers held at a script-based synchronization point are released simultaneously.
- There is no time limit to how long virtual testers can be held at the synchronization point.

However, if a synchronization point in a suite has a release time range and timeout period defined for it, the release times and timeout period apply to *all* synchronization points of that same name—even if a synchronization point is in a script.

Scope of a Synchronization Point

The scope of a synchronization point includes all scripts and all user groups that reference a particular synchronization point name.

For example, suppose a suite contains the following user groups:

- A Data Entry user group of 75 virtual testers. This user group runs a script containing the synchronization point Before Query.
- An Engineering user group of 10 virtual testers. This user group runs a different script than the Data Entry groups runs. But this script also contains a synchronization point named Before Query.
- A Customer Service user group of 25 virtual testers. This user group runs a script that contains no synchronization points. However, the user group does have a synchronization point defined for it. This synchronization point is also named Before Query.

At suite runtime, TestManager releases the virtual testers held at the Before Query synchronization point when all 110 virtual testers arrive at their respective synchronization points.

Comments

Use comments to document the script and to help you find your way around the script if you later need to edit it. Comments are ignored at compile time and during playback.

In VU, a comment begins with the characters `/*` and ends with the characters `*/` —for example:

```
/* This is a VU comment. */
```

In Visual Basic, comments begin with a single quotation mark:

```
'This is a Visual Basic comment.
```

In SQABasic, comments begin with a single quotation mark or the `rem` statement:

```
'This is an SQABasic comment.
Rem This is an SQABasic comment.
```

Adding Comments During Recording

To insert a comment into a script during recording:

1. If the Session Insert floating toolbar is not already displayed, click the **Insert** button on the Session Record floating toolbar.
2. Click the **Comment** button at that point in the script where you want to insert the comment.
3. Type your comment in the Comment dialog box (60 characters maximum), and then click **OK**.

When you add a comment during recording, the comment is reported as an annotation in the Annotations window.

Adding Comments During Editing

To add a comment during editing, type the comment directly into the script.

Comments that you type in manually during editing are not limited to the 60-character maximum that applies when you add comments during recording.

Using the Insert Menu

The preceding sections describe how to use the Session Insert floating toolbar to add timers, synchronization points, blocks, and comments to a script during recording.

During recording, you can also use the Robot Insert menu to add these features.

If Robot is minimized while you are recording (its default state), click the **Open Robot Window** button on the Session Record floating toolbar. This button restores the Robot window, letting you access the Insert menu.

▸ ▸ ▸ Part IV

Playing Back GUI Scripts

▸ ▸ ▸ C H A P T E R 9

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 TestManager log
- Analyzing verification point results with the Comparators

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.	*Restoring the Test Environment Before Playback* on page 9-3 *Setting GUI Playback Options* on page 9-3
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.	*Playing Back a GUI Script* on page 9-18
3. Analyze the results using the TestManager log.	*Viewing Results in the TestManager Log* on page 9-20 and the TestManager Help.
4. Use the appropriate Comparator to determine the cause of verification point failures.	*Analyzing Verification Point Results with the Comparators* on page 9-21 and the Comparators Help.
5. If the script fails, edit, debug, or rerecord the script so that it runs as required.	Chapter 5, *Editing, Compiling, and Debugging Scripts*
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 re-record the scripts.	*Creating Shell Scripts to Play Back Scripts in Sequence* on page 2-25

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.	*Restoring the Test Environment Before Playback* on page 9-3 *Setting GUI Playback Options* on page 9-3
2. Play back each script against a new build of the application-under-test.	*Playing Back a GUI Script* on page 9-18
3. Analyze the results using the TestManager log.	*Viewing Results in the TestManager Log* on page 9-20 and the TestManager Help.
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.	*Analyzing Verification Point Results with the Comparators* on page 9-21 and the Comparators Help.
5. Use the log to enter defects.	The TestManager Help
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 5, *Editing, Compiling, and Debugging Scripts*

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.

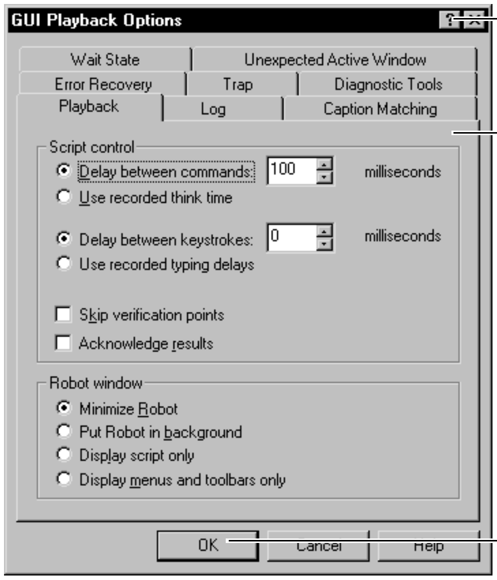
You can set these options either before you begin playback or early in the playback process.

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**.



GUI Playback Options

Script control
- [✓ Delay between commands: 100 milliseconds] / Use recorded think time
- [✓ Delay between keystrokes: 0 milliseconds] / Use recorded typing delays
- ☐ Skip verification points
- ☐ Acknowledge results

Robot window
- [✓ Minimize Robot]
- Put Robot in background
- Display script only
- Display menus and toolbars only

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

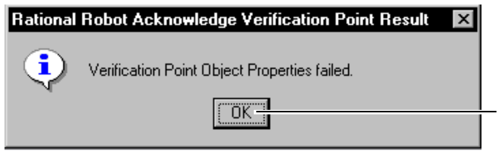
Set the options on each tab.

*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.



Rational Robot Acknowledge Verification Point Result

Verification Point Object Properties failed.

*Click **OK** to continue playback.*

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 log.

To set this option:

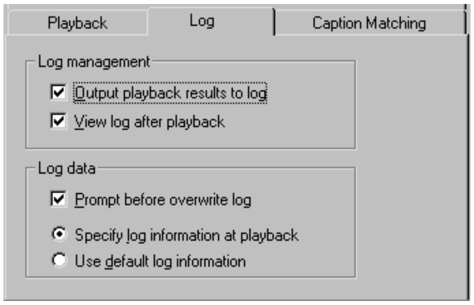
1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-3.)
2. In the **Playback** tab, do one of the following:
   – Select **Acknowledge results** to have Robot display a pass/fail result message box for each verification point. You must click **OK** before playback 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 TestManager. (For more information, see the TestManager Help)

To set the log options:

1. Open the GUI Playback Options dialog box. (See *Setting GUI Playback Options* on page 9-3.)
2. Click the **Log** tab.



Log management
☑ Output playback results to log
☑ View log after playback
Log data
☑ Prompt before overwrite log
● Specify log information at playback
○ Use default log information

3. To output the playback results to the log so you can view them, select **Output playback results to log**.

   If you clear this, you cannot to view the playback results in the log.

4. To have the log 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** → **TestManager**, 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.

   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.

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 3-12.) 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 4-8.)

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"
```

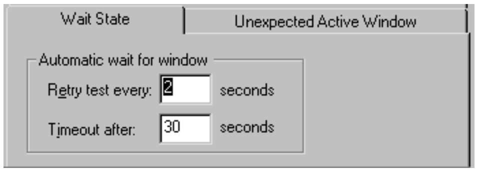
This example specifies that Robot should start an application, find a window on the desktop named “Form1”, find a pushbutton named “Command5”, and generate a click on that button. However, suppose Robot gets to the `SetContext` line in the script and fails to find a window named “Form1”. This may not 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 9-3.)
2. Click the **Wait State** tab.



Retry test every: 2 seconds
Timeout after: 30 seconds

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.

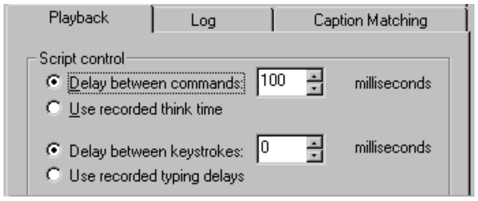
Setting Delay Options for Commands and Keystrokes

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 9-3.)
2. Click the **Playback** tab.



(●) Delay between commands: 100 milliseconds
( ) Use recorded think time
(●) Delay between keystrokes: 0 milliseconds
( ) Use recorded typing delays

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.

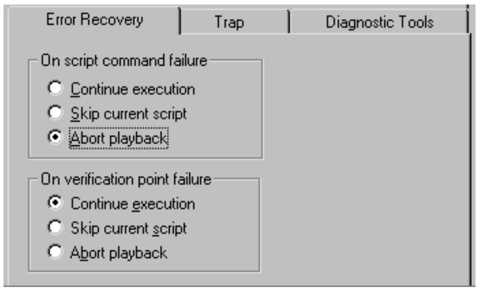
Setting Error Recovery Options

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 9-3.)

2. Click the **Error Recovery** tab.



On script command failure
- ( ) Continue execution
- ( ) Skip current script
- (●) Abort playback

On verification point failure
- (●) Continue execution
- ( ) Skip current script
- ( ) Abort playback

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.

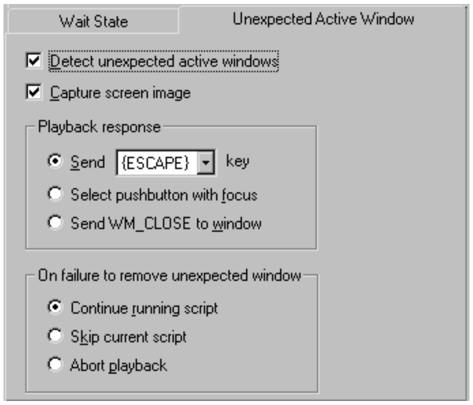
Failures are stored in the log.

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 9-3.)
2. Click the **Unexpected Active Window** tab.



☑ Detect unexpected active windows
☑ Capture screen image

Playback response
- ● Send {ESCAPE} key
- ○ Select pushbutton with focus
- ○ Send WM_CLOSE to window

On failure to remove unexpected window
- ● Continue running script
- ○ Skip current script
- ○ Abort playback

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 the Image Comparator Help.)

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 about an application 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:

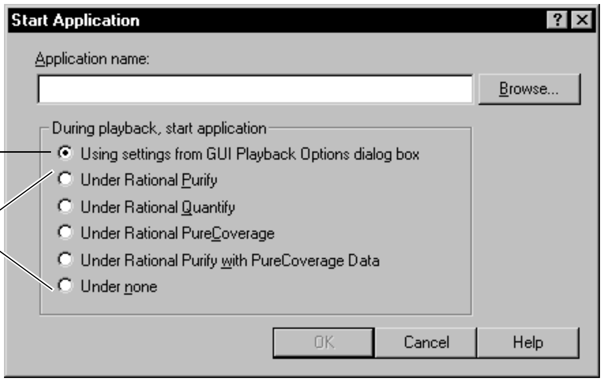
- Start Visual C/C++ and Visual Basic applications with the **Start Application** button or menu command.
- Start Java applications with the **Start Java Application** button or menu command.

For more information, see *Starting an Application* on page 3-1.

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.

How the Start Application and Diagnostic Tools Options Interact

When you record a script, you use the Start Application or Start Java Application dialog boxes to indicate how you want each application to start. These dialog boxes includes options for starting the application under a diagnostic tool during playback. (For more information, see *Starting an Application* on page 3-1.)



Starts application using the tool selected in the GUI Playback Options dialog box.

Overrides the tool selected in the GUI Playback Options dialog box.

- (●) Using settings from GUI Playback Options dialog box
- ( ) Under Rational Purify
- ( ) Under Rational Quantify
- ( ) Under Rational PureCoverage
- ( ) Under Rational Purify with PureCoverage Data
- ( ) Under none

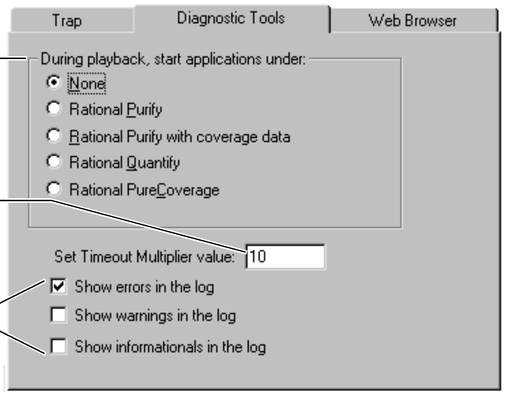
If you select a diagnostic tool during recording, that selection overrides the tool selected in the GUI Playback Options dialog box.

However, instead of selecting a diagnostic tool during recording, you can select **Using settings from GUI Playback Options dialog box**. In that case, you can specify the diagnostic tool for playback in the GUI Playback Options dialog box.

Setting the Diagnostic Tools 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 9-3.)
2. 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, change this value. This multiplies wait state and delay values.

c. Select the type of information to show in the log.

- (●) None
- ( ) Rational Purify
- ( ) Rational Purify with coverage data
- ( ) Rational Quantify
- ( ) Rational PureCoverage

Set Timeout Multiplier value: 10

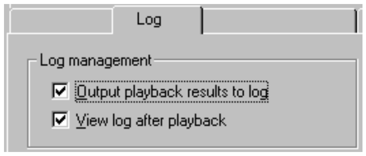
- [✓] Show errors in the log
- [ ] Show warnings in the log
- [ ] Show informationals in the log

NOTE: If a diagnostic tool was selected when the application was started during recording, that tool overrides the tool selected in the GUI Playback Options dialog box.

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 **Set timeout multiplier value**, 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 9-6.

Set timeout multiplier value is not used if the application is not run under a diagnostic tool.

3. If you selected any of the log check boxes in step 2c, click the **Log** tab and select both **Log management** check boxes.



Log management
- ☑ Output playback results to log
- ☑ View log after playback

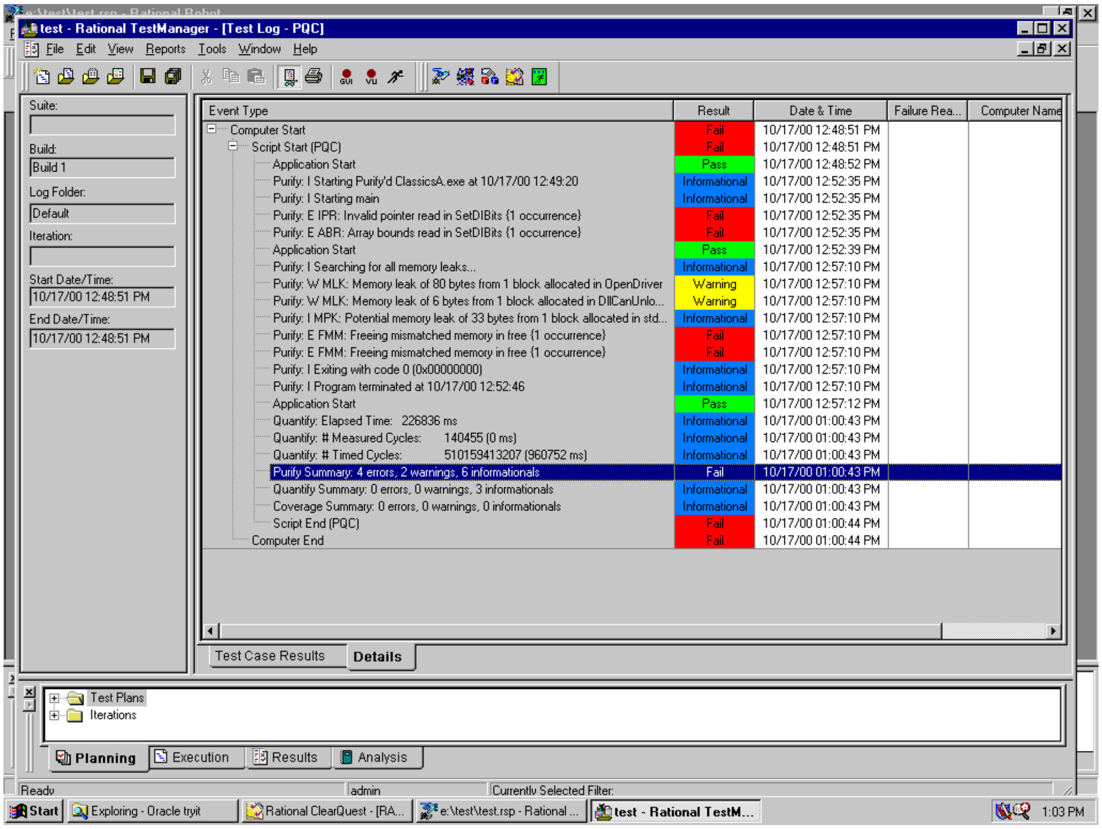
If **Output playback results to log** is cleared, the diagnostic tool opens (instead of the log) after you play back the script. (For more information about the log options, see *Setting Log Options for Playback* on page 9-5.)

4. 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.


test - Rational TestManager - [Test Log - PQC]
File Edit View Reports Tools Window Help
Test Case Results
Details
Test Plans
Iterations
Planning
Execution
Results
Analysis
Ready
admin
Currently Selected Filter:
Start
Exploring - Oracle tryit
Rational ClearQuest - [RA...
e:\test\test.rsp - Rational ...
test - Rational TestM...
1:03 PM

Suite:	
Build:	Build 1
Log Folder:	Default
Iteration:	
Start Date/Time:	10/17/00 12:48:51 PM
End Date/Time:	10/17/00 12:48:51 PM

Event Type	Result	Date & Time	Failure Rea...	Computer Name
Computer Start	Fail	10/17/00 12:48:51 PM		
Script Start (PQC)	Fail	10/17/00 12:48:51 PM		
Application Start	Pass	10/17/00 12:48:52 PM		
Purify: I Starting Purify'd ClassicsA.exe at 10/17/00 12:49:20	Informational	10/17/00 12:52:35 PM		
Purify: I Starting main	Informational	10/17/00 12:52:35 PM		
Purify: E IPR: Invalid pointer read in SetDIBits {1 occurrence}	Fail	10/17/00 12:52:35 PM		
Purify: E ABR: Array bounds read in SetDIBits {1 occurrence}	Fail	10/17/00 12:52:35 PM		
Application Start	Pass	10/17/00 12:52:39 PM		
Purify: I Searching for all memory leaks...	Informational	10/17/00 12:57:10 PM		
Purify: W MLK: Memory leak of 80 bytes from 1 block allocated in OpenDriver	Warning	10/17/00 12:57:10 PM		
Purify: W MLK: Memory leak of 6 bytes from 1 block allocated in DllCanUnlo...	Warning	10/17/00 12:57:10 PM		
Purify: I MPK: Potential memory leak of 33 bytes from 1 block allocated in std...	Informational	10/17/00 12:57:10 PM		
Purify: E FMM: Freeing mismatched memory in free {1 occurrence}	Fail	10/17/00 12:57:10 PM		
Purify: E FMM: Freeing mismatched memory in free {1 occurrence}	Fail	10/17/00 12:57:10 PM		
Purify: I Exiting with code 0 (0x00000000)	Informational	10/17/00 12:57:10 PM		
Purify: I Program terminated at 10/17/00 12:52:46	Informational	10/17/00 12:57:10 PM		
Application Start	Pass	10/17/00 12:57:12 PM		
Quantify: Elapsed Time: 226836 ms	Informational	10/17/00 01:00:43 PM		
Quantify: # Measured Cycles: 140455 (0 ms)	Informational	10/17/00 01:00:43 PM		
Quantify: # Timed Cycles: 510159413207 (960752 ms)	Informational	10/17/00 01:00:43 PM		
Purify Summary: 4 errors, 2 warnings, 6 informationals	Fail	10/17/00 01:00:43 PM		
Quantify Summary: 0 errors, 0 warnings, 3 informationals	Informational	10/17/00 01:00:43 PM		
Coverage Summary: 0 errors, 0 warnings, 0 informationals	Informational	10/17/00 01:00:43 PM		
Script End (PQC)	Fail	10/17/00 01:00:44 PM		
Computer End	Fail	10/17/00 01:00:44 PM		

A summary line at the end of the log indicates the total number of errors, warnings, and informational messages for each tool. The highlighted line in the figure above is an example of a summary line.

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 plays back more than one application under that tool.

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

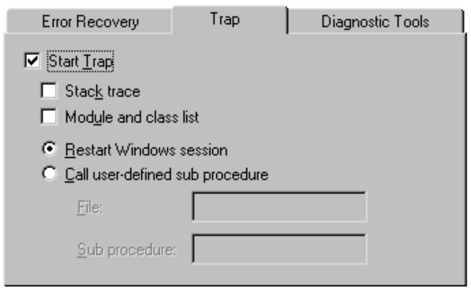
If one of these errors occurs, Trap appends the error data to the existing Sqatrap.log file in the Rational installation directory, or creates a new file if one does not exist. (For more information, see *Analyzing Results in the Sqatrap.log File* on page 9-17.)

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 9-3.)
2. Click the **Trap** tab.



☑ Start Trap
☐ Stack trace
☐ Module and class list
◉ Restart Windows session
○ Call user-defined sub procedure
File:
Sub procedure:

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

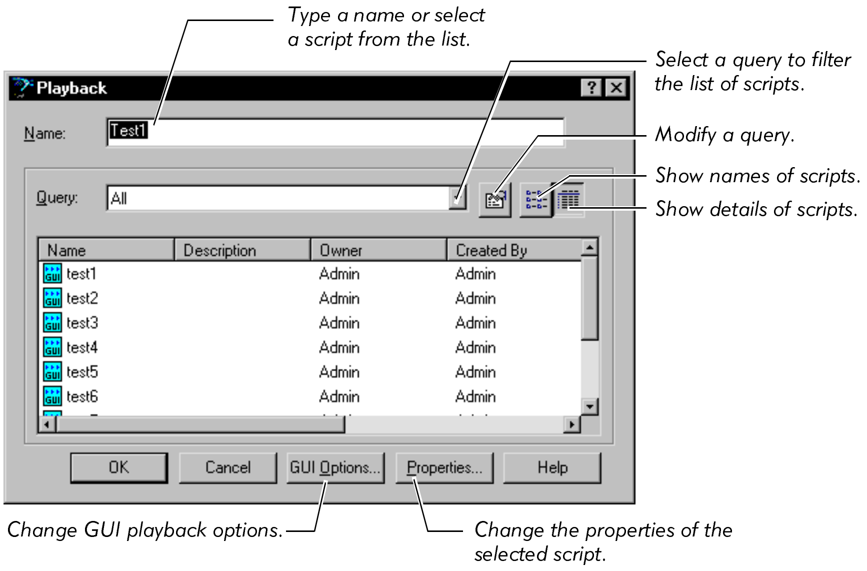
Other information can be written to Sqatrap.log depending on settings in the **Trap** tab of the GUI Playback Options dialog box.

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 9-3.)
3. Click the **Playback Script** button on the toolbar.


Type a name or select a script from the list.
Select a query to filter the list of scripts.
Modify a query.
Show names of scripts.
Show details of scripts.
Change GUI playback options.
Change the properties of the selected script.
Playback
OK
Cancel
GUI Options...
Properties...
Help

Name: Test1

Query: All

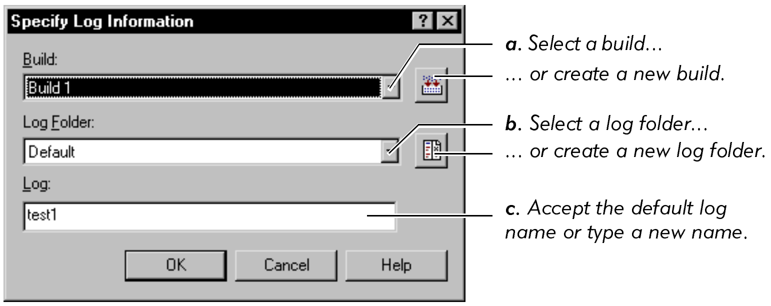
Name	Description	Owner	Created By
test1		Admin	Admin
test2		Admin	Admin
test3		Admin	Admin
test4		Admin	Admin
test5		Admin	Admin
test6		Admin	Admin

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.

   
   Specify Log Information
Build:
Build 1
Log Folder:
Default
Log:
test1
OK
Cancel
Help
a. Select a build...
... or create a new build.
b. Select a log folder...
... or create a new log folder.
c. Accept the default log name or type a new name.

   For information about builds, log folders, and logs, see the TestManager Help.
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.

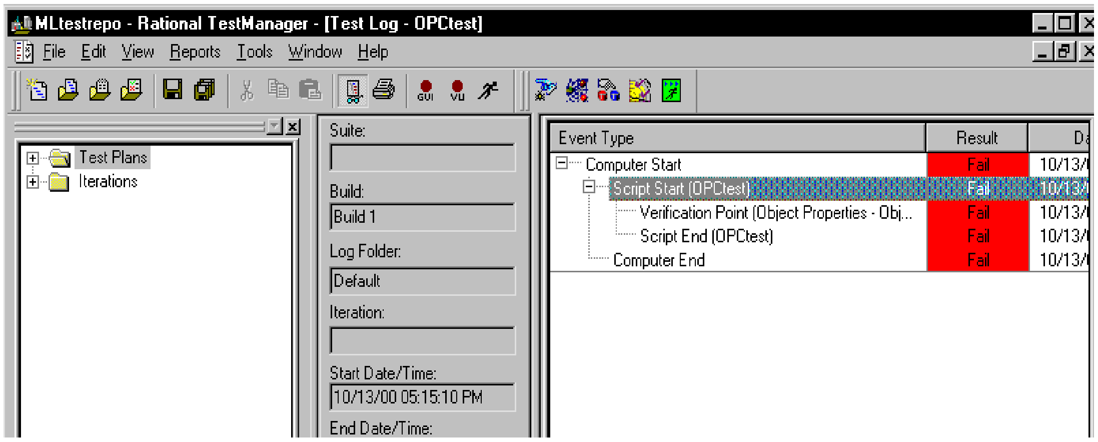
When you begin playback, the Robot main window is minimized by default. You can change this behavior in the **Playback** tab of the GUI Playback Options dialog box.

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 log, as described in the next section.

Viewing Results in the TestManager Log

After playback finishes, you can use the log to view the playback results, including verification point failures, procedural failures, aborts, and any additional playback information.


MLtestrepo - Rational TestManager - [Test Log - OPCtest]
File Edit View Reports Tools Window Help
GUI
VU
Test Plans
Iterations

Suite:

Build: Build 1

Log Folder: Default

Iteration:

Start Date/Time: 10/13/00 05:15:10 PM

End Date/Time:

Event Type	Result	Da
Computer Start	Fail	10/13/
Script Start (OPCtest)	Fail	10/13/
Verification Point (Object Properties - Obj...	Fail	10/13/
Script End (OPCtest)	Fail	10/13/
Computer End	Fail	10/13/

The following table gives you more information about the log.

To	Do this	For information, see
Control the log information and display	Set options in the **Log** tab of the GUI Playback Options dialog box.	*Setting Log Options for Playback* 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.	*Setting Log Options for Playback* on page 9-5
Analyze a failure in a Comparator	Select a verification point failure in the log and click **View** → **Verification Point**.	The next section, *Analyzing Verification Point Results with the Comparators*
Enter defects into Rational ClearQuest from the log	Select the failed event in the log and click **Defect** → **Generate**.	TestManager Help

For detailed information about the log, see the TestManager Help.

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.

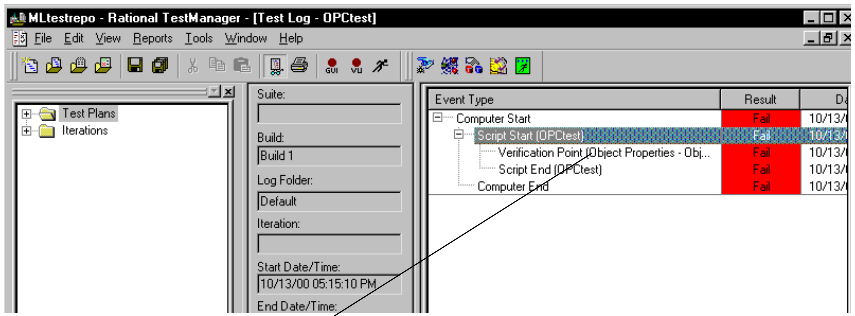
There are four Comparators, as follows:

Comparator	Description	For information see
Object Properties	Compares the baseline data to the data that caused a failure for the Object Properties verification point.	Object Properties Comparator Help
Text	Compares the baseline data to the data that caused a failure for the Alphanumeric verification point.	Text Comparator Help
Grid	Compares the baseline data to the data that caused a failure for the following verification points: Clipboard, Menu, and Object Data.	Grid Comparator Help
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.	Image Comparator Help

To open a Comparator from the log:

▶ In the **Log Event** column of a log, select a verification point and click **View** → **Verification Point**.

As the following figure shows, selecting an Object Properties verification point in the log and clicking **View** → **Verification Point** opens the Object Properties Comparator.


MLtestrepo - Rational TestManager - [Test Log - OPCtest]
File Edit View Reports Tools Window Help
Test Plans
Iterations

Suite:

Build: Build 1

Log Folder: Default

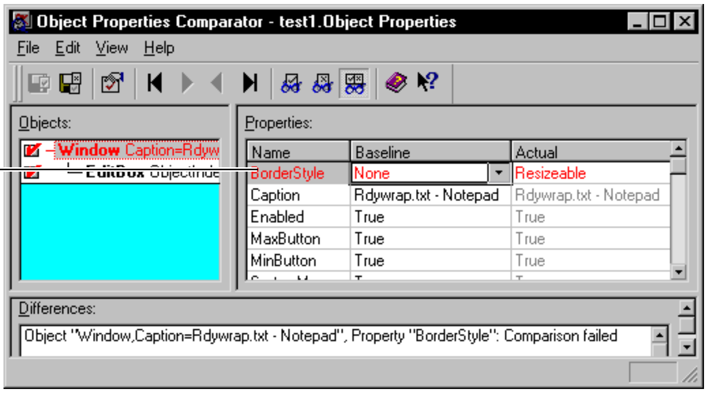
Iteration:

Start Date/Time: 10/13/00 05:15:10 PM

End Date/Time:

Event Type	Result	Da
Computer Start	Fail	10/13/
Script Start (OPCtest)	Fail	10/13/
Verification Point (Object Properties - Obj...	Fail	10/13/
Script End (OPCtest)	Fail	10/13/
Computer End	Fail	10/13/

Select a verification point in the log and click **View Verification Point***....*



Properties:

Name	Baseline	Actual
BorderStyle	None	Resizeable
Caption	Rdywrap.txt - Notepad	Rdywrap.txt - Notepad
Enabled	True	True
MaxButton	True	True
MinButton	True	True

Differences:

Object "Window,Caption=Rdywrap.txt - Notepad", Property "BorderStyle": Comparison failed

... to analyze the results in a 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 log.

▸▸▸ C H A P T E R 10

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 session scripts
- Using datapools with GUI scripts

You should familiarize yourself with the concepts and procedures in this chapter before you begin to work with datapools.

NOTE: This chapter describes datapool access from scripts played back in a TestManager suite. For information about datapool access from GUI scripts played back in Robot, see the online help for Rational Robot.

What Is a Datapool?

A **datapool** is a test dataset. It supplies data values to the variables in a script during script playback.

Datapools let you automatically pump test data to virtual testers under high-volume conditions that potentially involve hundreds of virtual testers performing thousands of transactions.

Typically, you use a datapool so that:

- Each virtual tester that runs the script can send realistic data (which can include unique data) to the server.
- A single virtual tester that performs the same transaction multiple times can send realistic data to the server in each transaction.

If you do not use a datapool during script playback, each virtual tester sends the same literal values to the server (the values that were captured when you recorded the script).

For example, suppose you record a session script that sends order number 53328 to a database server. If 100 virtual testers run this script, order number 53328 is sent to the server 100 times. If you use a datapool, each virtual tester can send a different order number to the server.

Datapool Tools

You create and manage datapools with either Robot or TestManager, as follows:

Activity	Robot	TestManager
Automatically generate datapool commands in a session script.	●	
Create a datapool and automatically generate datapool values.	●	●
Edit the DATAPOOL_CONFIG section of a session script.	●	
Edit datapool column definitions and 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 activities.

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 Robot or TestManager software 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 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 *Using Rational TestManager*.

NOTE: TestManager automatically copies a .csv file to each Agent computer that uses it. If an Agent's .csv file becomes out-of-date, TestManager automatically updates it.

Datapool Cursor

The datapool **cursor**, or row-pointer, can be shared among all users that access the datapool, or it can be unique for each user.

Sharing a datapool cursor among all users allows for a coordinated test. Because each row in the datapool is unique, each user can share the same cursor and still send unique records to the database.

Also, a shared cursor can be persistent across suite runs. For example, suppose the last datapool row accessed in a suite run is row 100:

- If the cursor is persistent across suite runs, datapool row access resumes with row 101 the first time the datapool is accessed in a new suite run.
- If the cursor is not persistent, datapool row access resumes with row 1 the first time the datapool is accessed in a new suite run.

NOTE: GUI users can share a cursor when playback occurs in a TestManager suite, but not when playback occurs in Robot.

For information about defining the scope of a cursor, see the description of the **Cursor** argument on page 10-15.

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.

With session scripts, you can control row access order through the **Access Order** setting in the Robot Configure Datapool in Test Script dialog box. (See page 10-16.)

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 you play back a script just once during a test run, that script probably does not need to access a datapool.

But often during a test run, and especially during performance testing, you need to run the same script multiple times — for example:

- During performance testing, you will probably want to run multiple instances of a script, so that the script is executed many times simultaneously. (Remember, a virtual tester is one runtime instance of a script.)
- During functional and performance testing, you will often want to run multiple iterations of a script, so that the script is executed many times consecutively (simulating a virtual tester performing the same task over and over).

If the values used in each script instance and each script iteration 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 solve:

- *Problem:* During recording, you create a personnel file for a new employee, using the employee's unique social security number. Each time the script is played back, 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 script is played back.

- *Problem:* You delete a record during recording. During playback, each instance and iteration of the script 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 script is played back.

- *Problem:* The client application reads a database record while you record a script for a performance test. During playback, that same record is read hundreds of times. Because the client application is well designed, it puts the record in cache memory, making its retrieval deceptively fast in subsequent fetches. The response times that the performance test yields will be inaccurate.

  Solution: Use a datapool to request a different record each time the script is played back.

Planning and Creating a Datapool

Here is a summary of the stages involved in preparing a datapool for use in testing. The order shown is the typical order for planning and creating a datapool for session scripts:

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 want 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 in a value for that field. With session scripts, the value is automatically assigned to a script variable. During playback, that 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.

   To help you determine the datapool columns you need, record a preliminary session script. Rational Robot automatically captures all the values supplied to the client application during recording and lists them in the `DATAPOOL_CONFIG` section at the end of the script. For more information, see *Finding Out What Data Types You Need* on page 10-9.

2. **Generate datapool code.**

   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 session scripts, a `DATAPOOL_CONFIG` section must also be present. This section contains a variety of information about how the datapool is created and accessed.

   Datapool code is generated in either of these ways:

   – With *session scripts*, Robot generates datapool code automatically when you finish recording a script. Robot is aware of all the variables in the script that are assigned values during recording, and it matches up each of these variables with a datapool column.

     To have Robot generate datapool commands automatically during recording, make sure **Use datapools** is selected in the **Generator** tab of the Session Record Options dialog box, and then record the script.

   – With *GUI scripts*, you manually insert the datapool commands and match up script variables with datapool columns. For information about coding datapool commands, see *Using Datapools with GUI Scripts* on page 10-23.

3. **Create and populate the datapool.**

   After the datapool commands are in the script, you can create and populate the datapool.

   To start creating and populating a datapool for a session script you are editing in Robot, click **Edit** → **Datapool Information**.

   If you are creating a datapool for exclusive use by a GUI script, use TestManager to create and populate the datapool. For more information, see *Using Rational TestManager*.

   Creating and populating a datapool for a session script involves these general steps:

   – Editing the `DATAPOOL_CONFIG` section of the script. For example, you might want to change the default datapool access flags, or exclude a datapool column from being created for a particular script variable. Or, you can accept all the default settings that Robot specifies when it creates this section in a session script.

     For information about editing the `DATAPOOL_CONFIG` section of a script, see *Step 1. Editing Datapool Configuration* on page 10-13.

   – Defining 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, see *Step 2. Defining Datapool Columns and Generating the Data* on page 10-19.

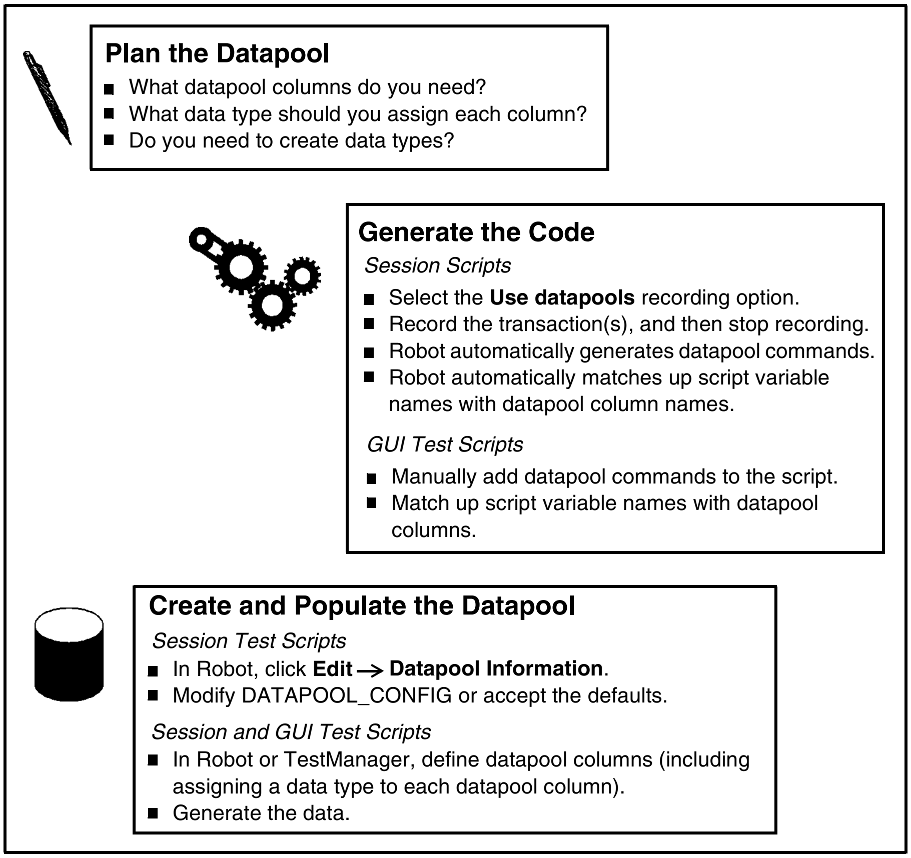
You also assign a data type to each datapool column. Data types supply a datapool column with its values. For information about data types, see *Data Types* on page 10-8.

– Generating the data. Once you configure the datapool and define its columns, you populate the datapool simply by clicking **Generate Data**.

With Robot, you can create and populate a datapool immediately after recording or at any other time, as long as the datapool commands are in the script.

NOTE: You can also create and populate a datapool file manually and import it into the project. For more information, see *Using Rational TestManager*.

The following figure illustrates the three stages of datapool creation:



Plan the Datapool

- What datapool columns do you need?
- What data type should you assign each column?
- Do you need to create data types?

Generate the Code

Session Scripts

- Select the **Use datapools** recording option.
- Record the transaction(s), and then stop recording.
- Robot automatically generates datapool commands.
- Robot automatically matches up script variable names with datapool column names.

GUI Test Scripts

- Manually add datapool commands to the script.
- Match up script variable names with datapool columns.

Create and Populate the Datapool

Session Test Scripts

- In Robot, click **Edit** → **Datapool Information**.
- Modify DATAPOOL_CONFIG or accept the defaults.

Session and GUI Test Scripts

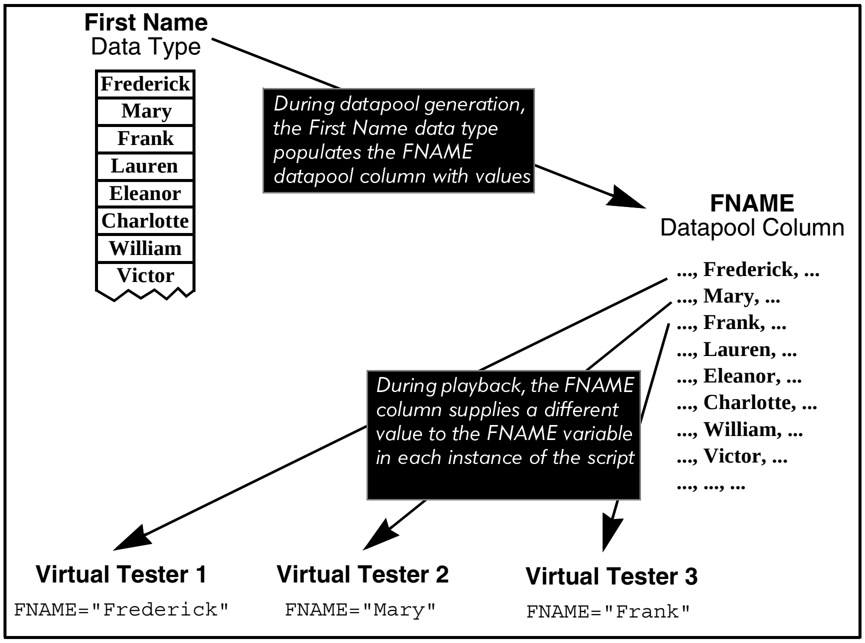
- In Robot or TestManager, define datapool columns (including assigning a data type to each datapool column).
- Generate the data.

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 you assign this data type to the datapool column FNAME. When Robot automatically generates the datapool, it populates the FNAME column with all of the values in the Names - First data type.

Here is 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 contains the kind of values you want to supply to a script variable.

User-defined data types are useful in situations such as:

– 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.

  To ensure that the variable is assigned a valid value from the datapool:

  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 are used in some foreign-language character sets. For more information, see the section *Generating Multi-Byte Characters* on page 10-12.

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.

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, will a variable contain names, dates, order numbers, and so on.

Here are two ways you can find the kind of values that are supplied to a variable:

▶ With session scripts, you can view the `DATAPOOL_CONFIG` section that Robot automatically adds to the end of the script. (Robot adds this information to a session script when you select **Use datapools** in the **Generator** tab of the Session Record Options dialog box.)

  The `DATAPOOL_CONFIG` section contains a line for each value assigned to a script variable during recording. In the following example, the value `329781` is assigned to the variable *`CUSTID`*:

```
INCLUDE, "CUSTID", "string", "329781"
```

  For more information about the `DATAPOOL_CONFIG` section of a script, see *Step 1. Editing Datapool Configuration* on page 10-13.

- With GUI scripts, you need to 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.

Finding Values in GUI Scripts

Here 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"
ComboListBox Click, "ObjectIndex=1", "Text=Discover"

'Credit Card Expiration Date
EditBox Left_Drag, "ObjectIndex=4", "Coords=19,13,16,12"
InputKeys "12/31/99"
```

To make the task of searching for values easier, 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.

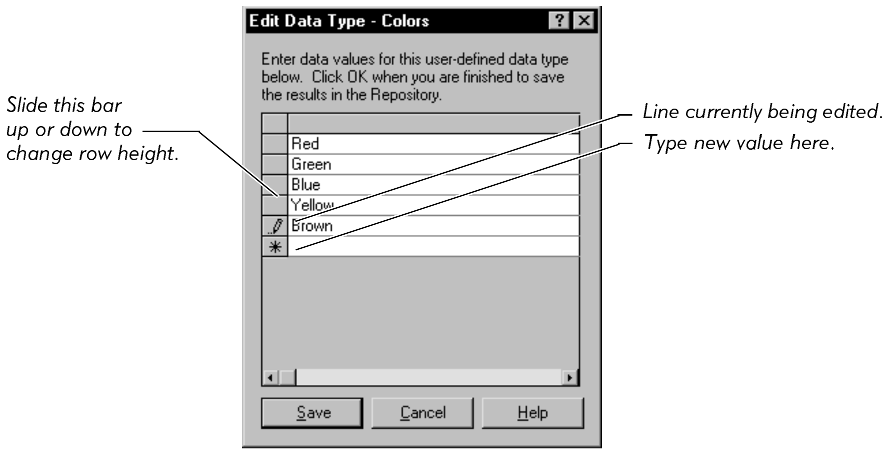
To create a user-defined data type in TestManager:

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 in 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 the values.
9. Click **Save**.

The following figure shows the data type Colors being populated with five values:


Edit Data Type - Colors

Enter data values for this user-defined data type below. Click OK when you are finished to save the results in the Repository.

Red
Green
Blue
Yellow
Brown

Slide this bar up or down to change row height.
Line currently being edited.
Type new value here.
Save
Cancel
Help

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 *Using Rational TestManager.*

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 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 *Using Rational TestManager*.

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 either of these ways:

- Through a user-defined data type. For information, see the section *Creating User-Defined Data Types* on page 10-10.

  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. It is included in the Japanese version of Microsoft Windows.

- Through the Read From File data type. For information, see the section *Using Rational TestManager*.

Using Datapools with Sessions

Robot can insert datapool commands into a performance testing session automatically. If you want Robot to do so, take these steps before you begin to record a session:

1. In Robot, click **Tools** → **Session Record Options**.
2. Click the **Generator** tab.
3. Select **Use datapools**, and then click **OK**.
4. Record your transaction(s), and then stop recording.

The datapool commands are included in the script that Robot generates during recording. Next, you create the datapool itself, as described in the following section.

Creating a Datapool with Robot

In Robot, you create a datapool in two basic steps:

1. Edit the `DATAPOOL_CONFIG` section of the session script, or accept the defaults. (See the next section, *Step 1. Editing Datapool Configuration*.)
2. Define datapool columns and generate the data. (See *Step 2. Defining Datapool Columns and Generating the Data* on page 10-19.)

   You cannot automatically generate data for a datapool that has more than 150 columns.

NOTE: `DATAPOOL_CONFIG` is included only in session scripts. As a result, this section is not applicable for GUI scripts, or if you are creating a datapool for exclusive use by a GUI script. To create a datapool that is accessed only by GUI scripts, see *Using Datapools with GUI Scripts* on page 10-23.

Step 1. Editing Datapool Configuration

You begin the process of creating a datapool by editing the `DATAPOOL_CONFIG` statement that Robot automatically generates in a script.

`DATAPOOL_CONFIG` has two basic purposes:

- During *datapool creation*, it specifies the datapool columns for Robot to create, if any.
- During *test runtime*, it provides information such as the access order of datapool rows, and whether script variables should be assigned values from the datapool or use the literal values provided during recording.

The best way to edit `DATAPOOL_CONFIG` is by editing the Robot Configure Datapool in Test Script dialog box rather than by editing the script directly.

To edit datapool configuration and to begin the process of defining and generating a datapool:

1. If the script that will access the datapool is not open for editing, click **File** → **Open** → **Test Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in Test Script dialog box.

   This dialog box lets you edit the `DATAPOOL_CONFIG` section of the script.
3. Either accept the defaults in the Configure Datapool in Test Script dialog box, or make any appropriate changes.

   Use the table on page 10-15 to help you modify the settings in this dialog box.

   NOTE: By default, the **Usage** column (see page 10-18) contains the value **EXCLUDE** for each script variable listed in the grid. This means that Robot does not create a datapool column for these variables when it creates the datapool. To have Robot automatically create datapool columns when it creates the datapool, change the **Usage** values to **INCLUDE** or **OVERRIDE**.
4. When finished making any changes, click **Save**.

   The `DATAPOOL_CONFIG` section of the script is updated according to the values set in the Configure Datapool in Test Script dialog box.

5. Take one of these actions:

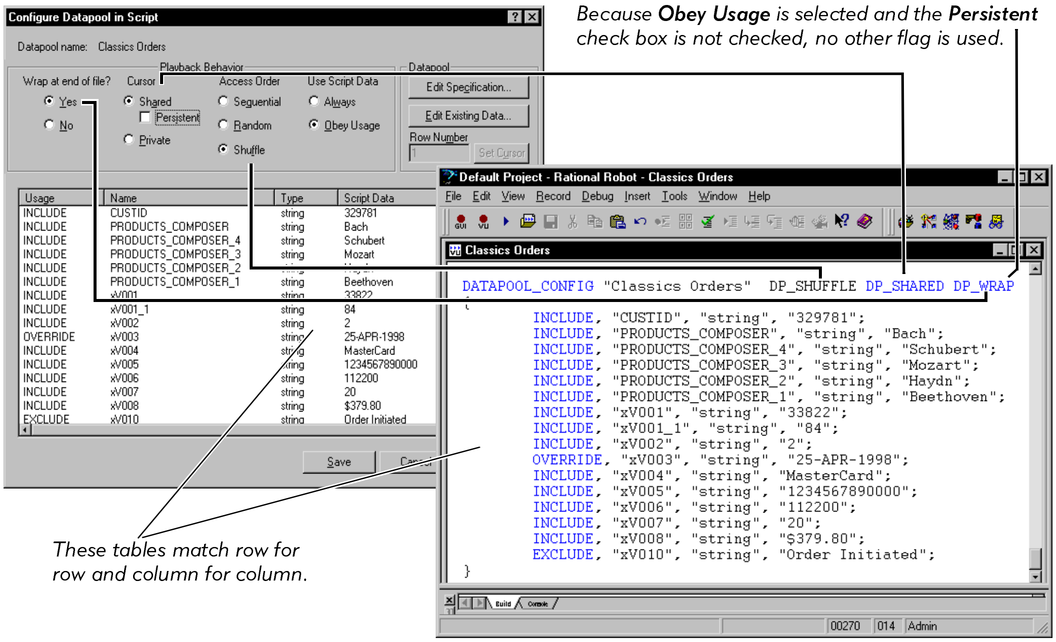
   – Click **Create** to define and populate the new datapool.

     If the datapool you are trying to create already exists, the **Create** button does not appear in the dialog box. Instead, the **Edit Specification** button appears, allowing you to edit datapool column definitions, and the **Edit Existing Data** button appears, allowing you to edit datapool values.

   – Click **Close** if you do not want to define and populate a datapool at this time.

6. If you clicked **Create** in the previous step, continue by following the instructions in the section *Step 2. Defining Datapool Columns and Generating the Data* on page 10-19.

Here is how the Configure Datapool in Test Script dialog box maps to the `DATAPOOL_CONFIG` section of a script:


Configure Datapool in Script
Playback Behavior
Datapool
Edit Specification...
Edit Existing Data...
Row Number
Set Cursor
Save
Cancel
Default Project - Rational Robot - Classics Orders
File Edit View Record Debug Insert Tools Window Help
Classics Orders
00270
014
Admin

Because **Obey Usage** is selected and the **Persistent** check box is not checked, no other flag is used.

Datapool name: Classics Orders

Wrap at end of file? [✓ Yes] No

Cursor: [✓ Shared] Persistent ☐ Private

Access Order: Sequential Random [✓ Shuffle]

Use Script Data: Always [✓ Obey Usage]

Row Number: 1

Usage	Name	Type	Script Data
INCLUDE	CUSTID	string	329781
INCLUDE	PRODUCTS_COMPOSER	string	Bach
INCLUDE	PRODUCTS_COMPOSER_4	string	Schubert
INCLUDE	PRODUCTS_COMPOSER_3	string	Mozart
INCLUDE	PRODUCTS_COMPOSER_2	string	Haydn
INCLUDE	PRODUCTS_COMPOSER_1	string	Beethoven
INCLUDE	xV001	string	33822
INCLUDE	xV001_1	string	84
INCLUDE	xV002	string	2
OVERRIDE	xV003	string	25-APR-1998
INCLUDE	xV004	string	MasterCard
INCLUDE	xV005	string	1234567890000
INCLUDE	xV006	string	112200
INCLUDE	xV007	string	20
INCLUDE	xV008	string	$379.80
EXCLUDE	xV010	string	Order Initiated

```
DATAPOOL_CONFIG "Classics Orders"  DP_SHUFFLE DP_SHARED DP_WRAP
{
        INCLUDE, "CUSTID", "string", "329781";
        INCLUDE, "PRODUCTS_COMPOSER", "string", "Bach";
        INCLUDE, "PRODUCTS_COMPOSER_4", "string", "Schubert";
        INCLUDE, "PRODUCTS_COMPOSER_3", "string", "Mozart";
        INCLUDE, "PRODUCTS_COMPOSER_2", "string", "Haydn";
        INCLUDE, "PRODUCTS_COMPOSER_1", "string", "Beethoven";
        INCLUDE, "xV001", "string", "33822";
        INCLUDE, "xV001_1", "string", "84";
        INCLUDE, "xV002", "string", "2";
        OVERRIDE, "xV003", "string", "25-APR-1998";
        INCLUDE, "xV004", "string", "MasterCard";
        INCLUDE, "xV005", "string", "1234567890000";
        INCLUDE, "xV006", "string", "112200";
        INCLUDE, "xV007", "string", "20";
        INCLUDE, "xV008", "string", "$379.80";
        EXCLUDE, "xV010", "string", "Order Initiated";
}
```

These tables match row for row and column for column.

For more information about the parts of the `DATAPOOL_CONFIG` section of a script, see the description of `DATAPOOL_CONFIG` in the *VU Language Reference*.

NOTE: Typically, a script has just one `DATAPOOL_CONFIG` section. If a script has multiple `DATAPOOL_CONFIG` sections (for example, to accommodate a script that accesses multiple databases and servers), the Configure Datapool in Test Script dialog box accesses the first one. To edit the others, you must edit the script directly.

Modifying DATAPOOL_CONFIG

Use the following table to help you define the fields and columns in the Configure Datapool in Test Script dialog box (see step 3 in the previous instructions):

Field or column	Description
Datapool name	The name assigned to the datapool. The datapool name defaults to the script name. You cannot modify **Datapool name**.
Wrap at end of file?	Sets the action to take after the last row in the access order is reached: ▶ **Yes** – Resume at the beginning of the access order. ▶ **No** – End access to the datapool. If you attempt to retrieve a datapool value after the end of the datapool is reached, a runtime error occurs. To ensure that unique datapool rows are fetched, choose **No**, and make sure the datapool has at least as many rows as the number of users (and user iterations) that will be requesting rows at runtime. With an access order of **Random**, this value is ignored.
Cursor	Specifies whether the datapool cursor is shared by all users accessing the datapool (**Shared**) or is unique to each user (**Private**). Also specifies whether a shared cursor is persistent across suite runs: ▶ With a shared cursor, all users work from the same access order. For example, if the access order for a Colors column is Red, Blue, and Green, the first user to request a value is assigned Red, the second is assigned Blue, and the third is assigned Green. ▶ If you check the **Persistent** box, the datapool cursor is persistent across suite runs. For example, if you have a persistent cursor with **Access Order** set to **Sequential**, and datapool row number 100 was the last row accessed in the last suite run, the first row accessed in the next suite run is 101. A persistent cursor resumes row access based on the last time the cursor was accessed as a *persistent* cursor. For example, suppose a cursor is persistent, and the last row accessed for that cursor in a suite run is 100. Then, the same suite is run again, but the cursor is now private. Row access ends at 50. If the cursor is set back to persistent the next time the suite is run, row access resumes with row 101, not 51. With persistent cursors, you can use the **Row** box to set the row to be accessed first in the next test run. Persistent cursors are only valid with shared cursors, and when **Access Order** is set to either **Sequential** or **Shuffle**. ▶ With a private cursor, each user starts at the top of its access order. With **Random** or **Shuffle** access, the access order is unique for each user and operates independently of the others. With **Sequential** access, the access order is the same for each user (ranging from the first row stored in the file to the last), but it operates independently for each user.

(Continued)

Field or column	Description
Access Order	Determines the sequence in which datapool rows are accessed: ▶ **Sequential** – Rows are accessed in the order in which they are physically stored in the datapool file, beginning with the first row in the file and ending with the last. ▶ **Random** – Rows are accessed in any order, and any given row can be accessed multiple times or not at all. ▶ **Shuffle** – Each time TestManager rearranges, or “shuffles,” the access order of all datapool rows, a unique sequence results. Each row is referenced in a shuffled sequence only once. Think of non-sequential access order (**Shuffle** and **Random**) as being like a shuffled deck of cards. With **Shuffle** access order, each time you pick a card (access a row), you place the card at the bottom of the pack. With **Random** access order, the selected card is returned anywhere in the pack — which means that one card might be selected multiple times before another is selected once. Also, with **Shuffle**, after each card has been selected once, you either resume selecting from the top of the same access order (**Wrap at end of file?** is **Yes**), or no more selections are made (**Wrap at end of file?** is **No**). With **Random**, you never reach the end of the pack (there is no end-of-file condition, so **Wrap at end of file?** is ignored).
Use Test Script Data	Specifies the source of the values that script variables are assigned during suite runtime, as follows: ▶ **Always** – Script variables are assigned the values provided during recording rather than values from the datapool. Recorded values are listed in the **Test Script Data** column. This option overrides the runtime meaning of the `INCLUDE` directive in the **Usage** column. It also adds the flag `OVERRIDE` to the `DATAPOOL_CONFIG` section of the script. This option provides a convenient way to run the script even if the datapool file is missing or incomplete. ▶ **Obey Usage** – Script variables associated with the `INCLUDE` directive in the **Usage** column are assigned datapool values. Script variables not associated with the `INCLUDE` directive are assigned values in the **Test Script Data** column. No flag is added to `DATAPOOL_CONFIG` with this option.

(Continued)

Field or column	Description
Datapool	Exits this dialog box to let you further define the datapool, and shows the next row to be accessed in the row access order, as follows: ▶ **Create** or **Edit Specification** – Lets you define datapool columns in a new or existing datapool, and lets you populate the datapool with values. ▶ **Edit Existing Data** – Lets you edit values in an existing datapool. ▶ **Row Number** – Shows the datapool row to be accessed first in the next test run. This box applies only to persistent cursors (the **Persistent** box must be checked). The row number is modifiable. Valid row numbers are 1 through 2,147,483,647 (commas are not allowed). If you specify a number that is not in the datapool, an error occurs at test runtime. After you specify a starting row number, click **Set Cursor**. Any changes you make in the **Datapool** group box do not affect the `DATAPOOL_CONFIG` section of the script.

(Continued)

Field or column	Description
Usage	Specifies one of the following directives to apply during database creation and during suite runtime. To change an individual directive, right-click the directive name: ▶ **INCLUDE** - During datapool creation, creates a column for the script variable in **Name**. The column is assigned the same name. - During suite runtime, assigns a value to the script variable in **Name** from the corresponding datapool column. You can override the runtime meaning of all `INCLUDE` directives by selecting **Always** in the **Use Test Script Data** group box. With **Always** selected, all script variables are assigned the associated values in the **Test Script Data** column. ▶ **EXCLUDE** - During datapool creation, does not create a column for the script variable in **Name**. - During suite runtime, assigns the value in **Test Script Data** to the script variable in **Name**. Datapool values are not used. ▶ **OVERRIDE** - During datapool creation, creates a column for the script variable in **Name**. The column is assigned the same name. - During suite runtime, assigns the value in **Test Script Data** to the script variable in **Name**. Datapool values are not used. You can select multiple **Usage** items using standard Windows selection methods (for example, holding down the CONTROL key while clicking each item to change). When all items are selected, right-click on one of them to change them all.
Name	The name of a script variable that is assigned a value during recording. If Robot creates a datapool column for this variable (if **Usage** is either `INCLUDE` or `OVERRIDE`), the datapool column is assigned the same name. This value can only be modified in the script.
Type	The data type of the value in **Test Script Data**. The data type is always `string`. This value can only be modified in the script.

(Continued)

Field or column	Description
Script Data	A value that was provided during recording. The value was assigned to the script variable in **Name**. If there is no value in this column for a particular script variable, a length of 1 is assigned to the datapool column associated with the script variable. This value can only be modified in the script.

Step 2. Defining Datapool Columns and Generating the Data

To complete the creation of the datapool that you started in *Step 1. Editing Datapool Configuration* on page 10-13, you define the datapool's columns and populate it with data. You do so in the Datapool Specification dialog box.

NOTE: If the Datapool Specification dialog box is not open, see *Step 1. Editing Datapool Configuration* on page 10-13 to learn how to open it.

The Datapool Specification dialog box contains the **Datapool Fields** grid. Each row in the grid represents a datapool field — that is, a column of data in the datapool file.

When the dialog box opens, the grid lists a datapool column name and a default column definition for each script variable that is assigned the value `INCLUDE` or `OVERRIDE` in the Configure Datapool in Test Script dialog box.

You define and populate the datapool as follows:

1. To insert one or more new columns into the datapool file:

   a. Click the row located either just before or just after the location where you want to insert the new datapool column. (Note that the order in which datapool column names are listed in **Name** determines the order in which values are stored in a datapool record.)

   An arrow appears next to the name of the datapool row you clicked.

   b. Click either **Insert before** or **Insert after**, depending on where you want to insert the datapool column.

   c. Type a name for the new datapool column (40 characters maximum).

   Make sure there is a script variable of the same name listed in the Configure Datapool in Test Script dialog box. The case of the names must match.

2. For each datapool column in the grid, assign a data type to the column, and modify the default property values for the column as appropriate.

   For information about the data types and other properties you can define for a datapool column, see *Using Rational TestManager*.

   To see an example of datapool columns defined in the Datapool Specification dialog box, see *Using Rational TestManager*.

3. When finished defining datapool columns, type a number in the **No. of records to generate** field.

   If a different row has to be retrieved with each fetch, make sure the datapool has at least as many rows as the number of users (and user iterations) that will be requesting rows at runtime.

4. Click **Generate Data**.

   You cannot generate data for a datapool that has more than 150 columns.

   Alternatively, if you do not want to generate any data now, click **Save** to save your datapool column definitions, and then click **Close**.

5. 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 Configure Datapool in Test Script dialog box, click **Edit Existing 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 the contents of the datapool are as you expect.

Editing Datapool Column Definitions with Robot

To edit datapool column definitions in Robot, you must begin in the Configure Datapool in Test Script dialog box.

This section provides the basic steps for editing datapool column definitions while in Robot. For information about the Configure Datapool in Test Script dialog box, see *Step 1. Editing Datapool Configuration* on page 10-13.

To edit a datapool's column definitions while in Robot:

1. If the script that will access the datapool is not open for editing, click **File** → **Open** → **Test Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in Test Script dialog box.
3. Either accept the defaults in the Configure Datapool in Test Script dialog box, or make any appropriate changes.

   Use the table on page 10-15 to help you modify the settings in this dialog box.
4. When finished making any changes, click **Save**.
5. Click **Edit Specification** to open the Datapool Specification dialog box, where you update datapool column definitions.

   For information about the data types and other properties you can define for a datapool column, see *Using Rational TestManager*.

   To see an example of datapool columns defined in the Datapool Specification dialog box, see *Using Rational TestManager*.
6. To insert one or more new columns into the datapool file:

   a. Click the row located either just before or just after the location where you want to insert the new datapool column. (Note that the order in which datapool column names are listed in **Name** determines the order in which values are stored in a datapool record.)

      An arrow appears next to the name of the datapool row you clicked.

   b. Click either **Insert before** or **Insert after**, depending on where you want to insert the datapool column.

   c. Type a name for the new datapool column (40 characters maximum).

      Make sure there is a script variable of the same name listed in the Configure Datapool in Test Script dialog box. Case of the names must match.
7. When finished modifying datapool columns, type a number in the **No. of records to generate** field.

   If a different row has to be retrieved with each fetch, make sure the datapool has at least as many rows as the number of users (and user iterations) that will be requesting rows at runtime.
8. Click **Generate Data**.

You cannot generate data for a datapool that has more than 150 columns.

Alternatively, if you do not want to generate any data now, click **Save** to save your datapool column definitions, and then click **Close**.

9. 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.

Editing Datapool Values with Robot

To edit datapool values in Robot, you must begin in the Configure Datapool in Test Script dialog box.

This section provides the basic steps for editing datapool values while in Robot. For information about the Configure Datapool in Test Script dialog box, see *Step 1. Editing Datapool Configuration* on page 10-13.

To view or edit a datapool's values while in Robot:

1. If the script that will access the datapool is not open for editing, click **File** → **Open** → **Test Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in Test Script dialog box.
3. Either accept the defaults in the Configure Datapool in Test Script dialog box, or make any appropriate changes.

   Use the table on page 10-15 to help you modify the settings in this dialog box.
4. When finished making any changes, click **Save**.
5. Click **Edit Existing Data**.
6. In the Edit Datapool dialog box, edit datapool values as appropriate.

   For information about editing datapool values, see *Using Rational TestManager*.
7. When finished editing datapool values, click **Save**, and then click **Close**.

For an example of the datapool values that TestManager generates, see *Using Rational TestManager*.

Cancelling Your Edits

To abandon all 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.

Using Datapools with GUI Scripts

A GUI script can access a datapool when it is played back in Robot. Also, when a GUI script is played back in a TestManager suite, the GUI script can access the same datapool as other GUI scripts and/or session scripts.

There are differences in the way GUI scripts and session scripts are set up for datapool access:

- You must add datapool commands to GUI scripts manually while editing the script in Robot. Robot adds datapool commands to session scripts automatically.
- There is no `DATAPOOL_CONFIG` statement in a GUI script. The command `SQADatapoolOpen` defines the access method to use for the datapool.

Although there are differences in setting up datapool access in GUI scripts and session scripts, you define a datapool for either type of script using TestManager in exactly the same way.

Here are the general tasks involved in providing access to a datapool from a GUI script. The steps 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.

For information about recording a GUI script for datapool access and adding datapool commands to a GUI script, see Chapter 5 of the *SQABasic Language Reference*. For additional information and examples, see the SQABasic Help, under the index keyword *datapools*.

For information about creating a datapool with TestManager, see the TestManager Help.

Accessing a Datapool from GUI and Session Scripts

If a GUI script and a session script provide the same set of values to the client application during recording, the scripts can access the same datapool during playback in a TestManager suite.

Here is the suggested order of steps for creating a datapool and setting up access to it from GUI scripts and session scripts:

1. Record the procedure and create the datapool as described in *Using Datapools with Sessions* on page 10-12.
2. Record the GUI script.
3. Edit the GUI script and add datapool commands to it.

For information about recording GUI scripts and adding datapool commands to GUI scripts, see the *Using Rational Robot* manual.

If you want to ensure that virtual testers retrieve a unique row of data from the datapool, follow the guidelines listed in *Using Rational TestManager*.

▸ ▸ ▸ Part V

Testing IDE Applications

▸ ▸ ▸ C H A P T E R 11

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
- Verifying that the Visual Basic extension is loaded

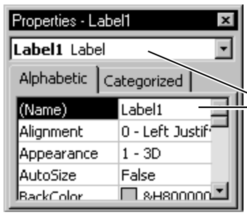
About Robot Support for Visual Basic Applications

Rational Robot provides comprehensive support for testing 32-bit applications built with Visual Basic version 4.0 through 6.0. 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.

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.

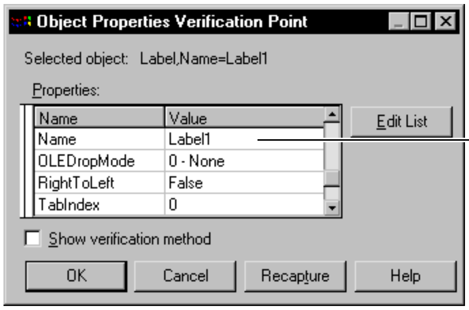


(Name)	Label1
Alignment	0 - Left Justif
Appearance	1 - 3D
AutoSize	False
BackColor	&H800000

... The object name that appears 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 ...



Selected object: Label,Name=Label1

Properties:

Name	Value
Name	Label1
OLEDropMode	0 - None
RightToLeft	False
TabIndex	0

... and the same name that is captured when you create an Object Properties verification point on 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.

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.

▸ ▸ ▸ C H A P T E R 12

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
- 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, 5.0, and 6.0. 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
- Oracle Trees (Tree-views)
- Base-table blocks (single- and multi-record)
- Items, including OLE containers

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 and generate the form(s) afterward.
- 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 the *Installing Rational Testing Products* manual.

You then need to run the Enabler, which instruments the code in your .fmb file to make your application testable by Robot.

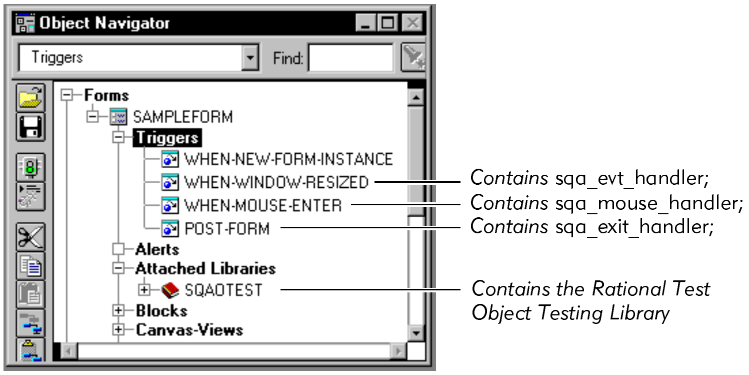
Running the Enabler on Your Application

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.

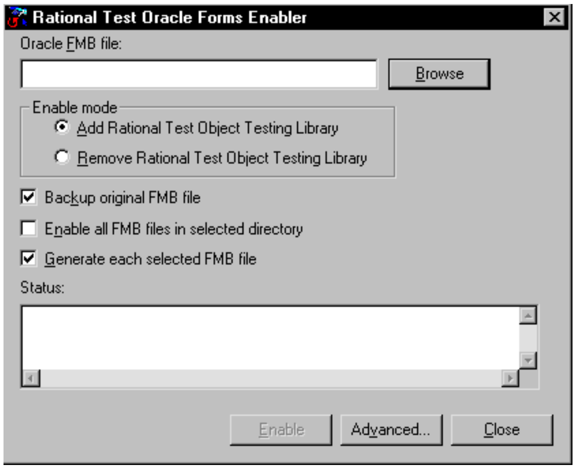
You can leave the triggers and the Object Testing Library in your application when you distribute it. For more information, see *Distributing Your Application* on page 12-6.

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:


Rational Test Oracle Forms Enabler
Oracle FMB file:
Browse
Enable mode
Add Rational Test Object Testing Library
Remove Rational Test Object Testing Library
Backup original FMB file
Enable all FMB files in selected directory
Generate each selected FMB file
Status:
Enable
Advanced...
Close

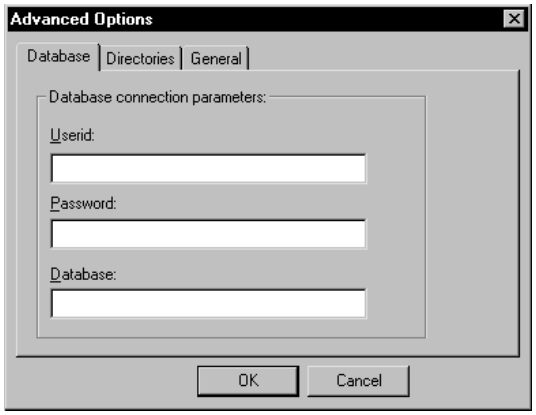
2. Click **Browse**. Select the .fmb file that you want to make testable and click **OK**.
3. Click **Add Rational Test Object Testing Library**.
4. Set the following options as needed:

   Backup original FMB file – Creates a backup file before the file is enabled.

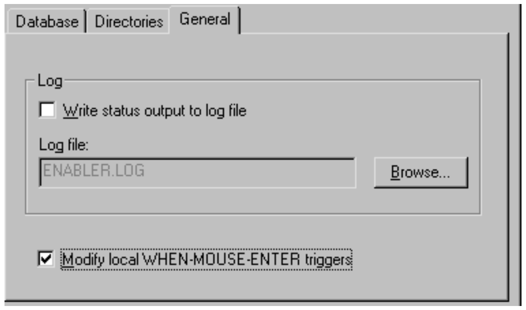
   Enable all FMB files in selected directory – Enables every .fmb file in the directory. If this check box is not selected, only the .fmb file in the **Oracle FMB file** box is enabled.

   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 6.0 Builder, 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.


Database
Directories
General
Log
Write status output to log file
Log file:
ENABLER.LOG
Browse...
Modify local WHEN-MOUSE-ENTER triggers

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 `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, clear **Modify local WHEN-MOUSE-ENTER triggers**. (For information about the Enabler and triggers, see *What Happens When You Run the Enabler?* on page 12-2.)

    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 6.0 or 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 12-3. 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.

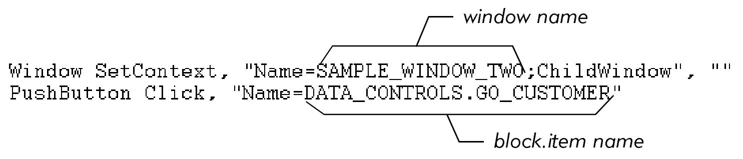
You can record actions against visual objects, and you can test both visual and nonvisual objects.

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:



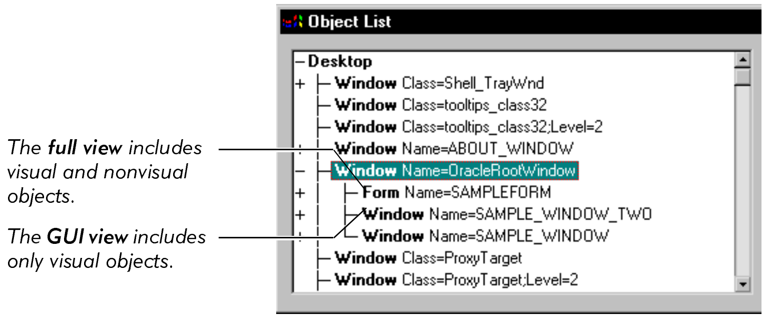
```
Window SetContext, "Name=SAMPLE_WINDOW_TWO;ChildWindow", ""
PushButton Click, "Name=DATA_CONTROLS.GO_CUSTOMER"
```

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.

The following figure shows the full view and the GUI view (collapsed):



*The **full view** includes visual and nonvisual objects.*

*The **GUI view** includes only visual objects.*

```
– Desktop
+ ├ Window Class=Shell_TrayWnd
  ├ Window Class=tooltips_class32
  ├ Window Class=tooltips_class32;Level=2
  ├ Window Name=ABOUT_WINDOW
– ├ Window Name=OracleRootWindow
+ │ ├ Form Name=SAMPLEFORM
+ │ ├ Window Name=SAMPLE_WINDOW_TWO
  │ └ Window Name=SAMPLE_WINDOW
  ├ Window Class=ProxyTarget
  ├ Window Class=ProxyTarget;Level=2
```

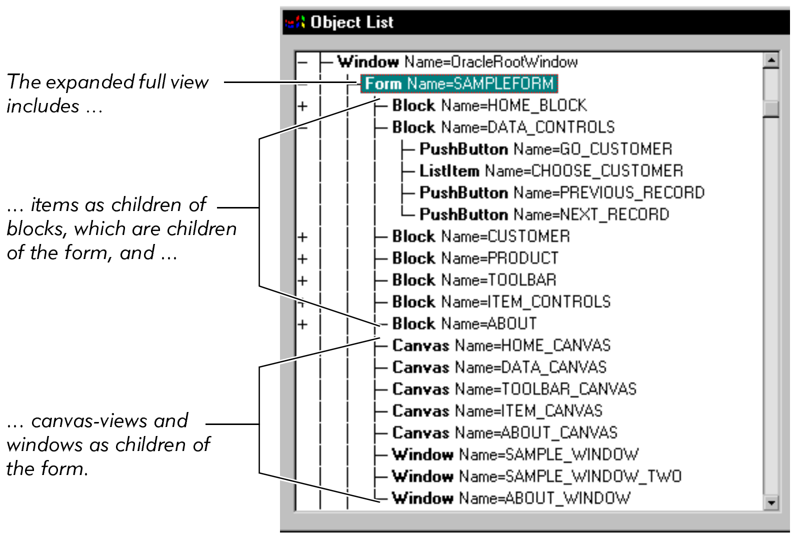
To see both Robot views of an Oracle application:

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 expanded full view includes ...

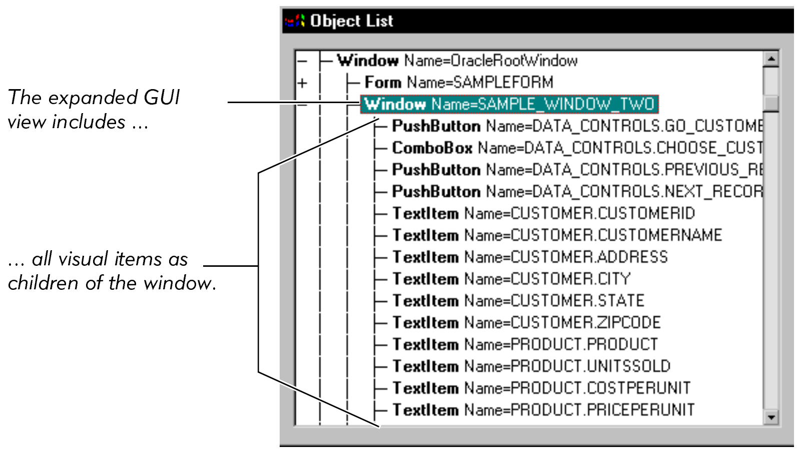
... items as children of blocks, which are children of the form, and ...

... canvas-views and windows as children of the form.

Object List

```
- Window Name=OracleRootWindow
  Form Name=SAMPLEFORM
+   Block Name=HOME_BLOCK
-   Block Name=DATA_CONTROLS
      PushButton Name=GO_CUSTOMER
      ListItem Name=CHOOSE_CUSTOMER
      PushButton Name=PREVIOUS_RECORD
      PushButton Name=NEXT_RECORD
+   Block Name=CUSTOMER
+   Block Name=PRODUCT
+   Block Name=TOOLBAR
+   Block Name=ITEM_CONTROLS
+   Block Name=ABOUT
    Canvas Name=HOME_CANVAS
    Canvas Name=DATA_CANVAS
    Canvas Name=TOOLBAR_CANVAS
    Canvas Name=ITEM_CANVAS
    Canvas Name=ABOUT_CANVAS
    Window Name=SAMPLE_WINDOW
    Window Name=SAMPLE_WINDOW_TWO
    Window Name=ABOUT_WINDOW
```

The following figure shows the expanded GUI view of the Window branch:



The expanded GUI view includes ...

... all visual items as children of the window.

Object List

```
- Window Name=OracleRootWindow
+   Form Name=SAMPLEFORM
-   Window Name=SAMPLE_WINDOW_TWO
      PushButton Name=DATA_CONTROLS.GO_CUSTOME
      ComboBox Name=DATA_CONTROLS.CHOOSE_CUST
      PushButton Name=DATA_CONTROLS.PREVIOUS_RE
      PushButton Name=DATA_CONTROLS.NEXT_RECOR
      TextItem Name=CUSTOMER.CUSTOMERID
      TextItem Name=CUSTOMER.CUSTOMERNAME
      TextItem Name=CUSTOMER.ADDRESS
      TextItem Name=CUSTOMER.CITY
      TextItem Name=CUSTOMER.STATE
      TextItem Name=CUSTOMER.ZIPCODE
      TextItem Name=PRODUCT.PRODUCT
      TextItem Name=PRODUCT.UNITSSOLD
      TextItem Name=PRODUCT.COSTPERUNIT
      TextItem Name=PRODUCT.PRICEPERUNIT
```

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 12-2.

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
  Check Box item
  Display item
  Image item
  List item
  OLE Container item
  Oracle Tree (Tree-view)
  Push Button item
  Radio Button item
  Radio Group item
  Text item
  Oracle VBX Control 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 12-13 and *Testing LOVs and Record Groups* on page 12-15.

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.

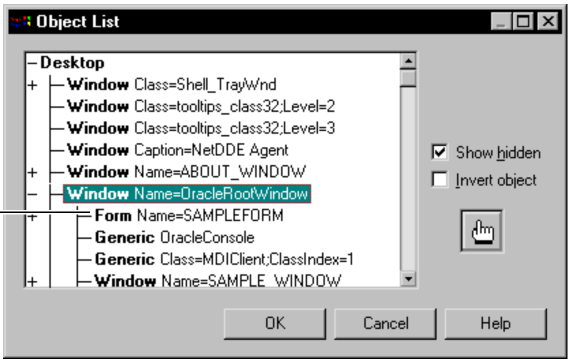
Testing Properties of Nonvisual Objects

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**.

Contains the full view of objects, which includes nonvisual objects.



```
-Desktop
+ |-Window Class=Shell_TrayWnd
  |-Window Class=tooltips_class32;Level=2
  |-Window Class=tooltips_class32;Level=3
  |-Window Caption=NetDDE Agent
+ |-Window Name=ABOUT_WINDOW
- |-Window Name=OracleRootWindow
+   |-Form Name=SAMPLEFORM
    |-Generic OracleConsole
    |-Generic Class=MDIClient;ClassIndex=1
+   |-Window Name=SAMPLE_WINDOW
```

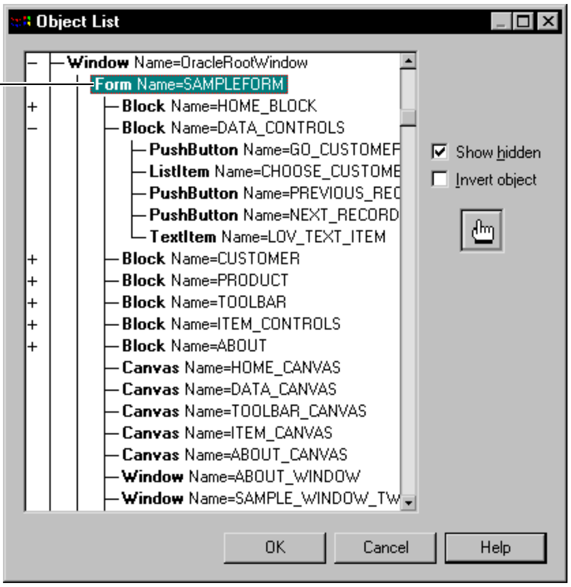
☑ Show hidden
☐ Invert object

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.

4. 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 4-10.)

The form includes blocks, items, canvas-views, and windows.



```
Window Name=OracleRootWindow
  Form Name=SAMPLEFORM
    Block Name=HOME_BLOCK
    Block Name=DATA_CONTROLS
      PushButton Name=GO_CUSTOMEF
      ListItem Name=CHOOSE_CUSTOME
      PushButton Name=PREVIOUS_REC
      PushButton Name=NEXT_RECORD
      TextItem Name=LOV_TEXT_ITEM
    Block Name=CUSTOMER
    Block Name=PRODUCT
    Block Name=TOOLBAR
    Block Name=ITEM_CONTROLS
    Block Name=ABOUT
    Canvas Name=HOME_CANVAS
    Canvas Name=DATA_CANVAS
    Canvas Name=TOOLBAR_CANVAS
    Canvas Name=ITEM_CANVAS
    Canvas Name=ABOUT_CANVAS
    Window Name=ABOUT_WINDOW
    Window Name=SAMPLE_WINDOW_TW
```

5. Select the object to test and click **OK**. If the Select Object dialog box still appears, click **OK**.

6. 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 12-7.

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 12-15.

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

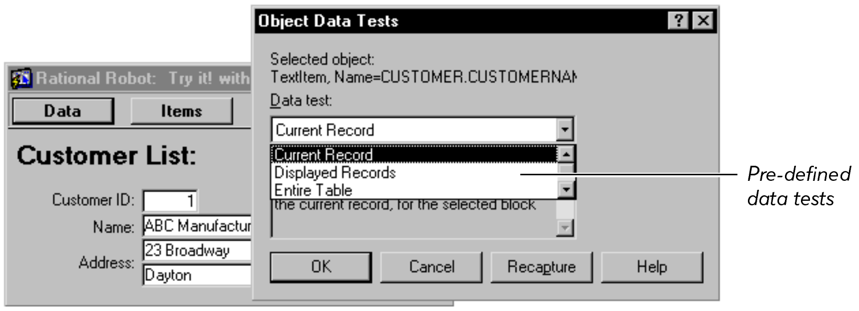
NOTE: Before you can test an Oracle Forms application, you need to run the Enabler. For instructions, see *Making Oracle Forms Applications Testable* on page 12-2.

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.


Object Data Tests

Selected object:
TextItem, Name=CUSTOMER.CUSTOMERNAI

Data test: Current Record

Current Record
Displayed Records
Entire Table

the current record, for the selected block

Customer List:

Customer ID: 1
Name: ABC Manufactur
Address: 23 Broadway
Dayton

You can use the Object Data Test Definition to define additional data tests. (For information, see Appendix A, *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 12-13.)

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

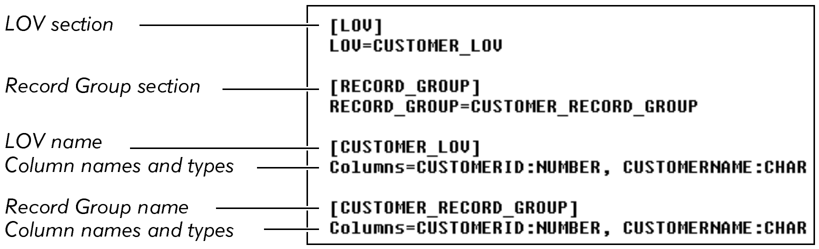
Before you test an LOV or record group, you need to create an .sqa text file. To create this file:

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.



```
[LOV]
LOV=CUSTOMER_LOV

[RECORD_GROUP]
RECORD_GROUP=CUSTOMER_RECORD_GROUP

[CUSTOMER_LOV]
Columns=CUSTOMERID:NUMBER, CUSTOMERNAME:CHAR

[CUSTOMER_RECORD_GROUP]
Columns=CUSTOMERID:NUMBER, CUSTOMERNAME:CHAR
```

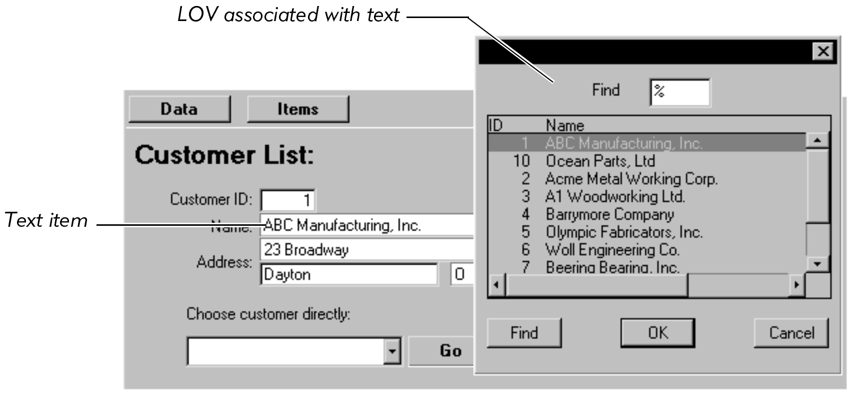
After you create the .sqa file, you can capture the data in:

- 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.



Customer ID: 1
Name: ABC Manufacturing, Inc.
Address: 23 Broadway
Dayton 0

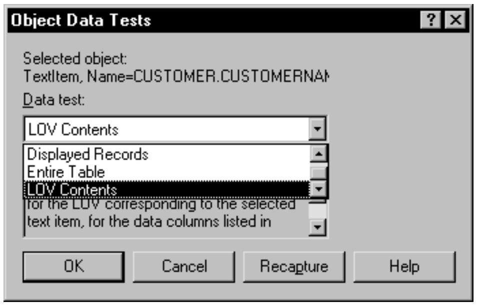
Find: %

ID	Name
1	ABC Manufacturing, Inc.
10	Ocean Parts, Ltd
2	Acme Metal Working Corp.
3	A1 Woodworking Ltd.
4	Barrymore Company
5	Olympic Fabricators, Inc.
6	Woll Engineering Co.
7	Beering Bearing, Inc.

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**.


Object Data Tests
Selected object:
TextItem, Name=CUSTOMER.CUSTOMERNAI
Data test:
LOV Contents
Displayed Records
Entire Table
LOV Contents
for the LOV corresponding to the selected
text item, for the data columns listed in
OK
Cancel
Recapture
Help

6. Click **OK** to open the Object Data Verification Point dialog box.

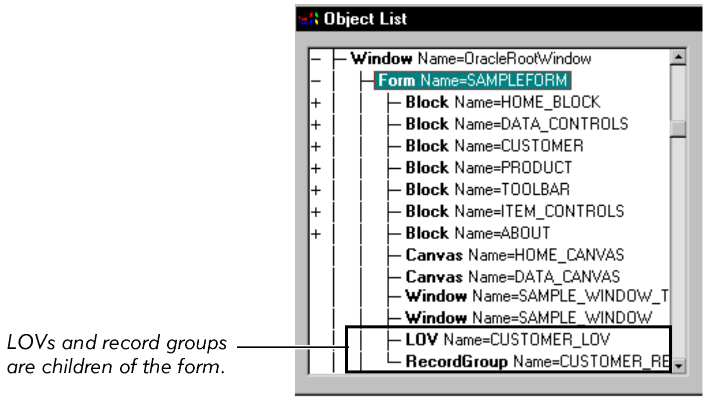
   If you typed incorrect information in the .sqa file, a message appears in the data grid in the dialog box.
7. Complete the verification point as usual.

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.

6. Expand the **Form** branch by double-clicking the plus sign.



```
Object List
–  ├ Window Name=OracleRootWindow
–  │  ├ Form Name=SAMPLEFORM
+  │  │  ├ Block Name=HOME_BLOCK
+  │  │  ├ Block Name=DATA_CONTROLS
+  │  │  ├ Block Name=CUSTOMER
+  │  │  ├ Block Name=PRODUCT
+  │  │  ├ Block Name=TOOLBAR
+  │  │  ├ Block Name=ITEM_CONTROLS
+  │  │  ├ Block Name=ABOUT
   │  │  ├ Canvas Name=HOME_CANVAS
   │  │  ├ Canvas Name=DATA_CANVAS
   │  │  ├ Window Name=SAMPLE_WINDOW_T
   │  │  ├ Window Name=SAMPLE_WINDOW
   │  │  ├ LOV Name=CUSTOMER_LOV
   │  │  └ RecordGroup Name=CUSTOMER_RE
```

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 12-15.

7. 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.

8. Complete the verification point as usual.

▸ ▸ ▸ CHAPTER 13

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 your browser for testing
- 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 both record and play back scripts in both Microsoft Internet Explorer versions 4.x and later and Netscape Navigator versions 4.x. Scripts can be recorded and played back on a variety of Windows platforms, including Windows NT 4.0, Windows 2000, Windows 98, and Windows 95.

Configuring Your Browser for Testing

Before you record scripts, you should configure Internet Explorer and/or Netscape Navigator so that scripts will play back in the same way as when you recorded them. For best results, you should configure Internet Explorer and/or Navigator 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 in Internet Explorer 5:

1. Start Internet Explorer 5.
2. Click **Tools** → **Internet Options**.
3. Click the **Security** tab.
4. Make sure the **Internet** icon is selected.
5. Click the **Custom Level** button.
6. Scroll to the **Cookies** section.
7. Click **Disable** in both cookies options.
8. Click **OK** two times.

To disable the cookie prompt in Netscape Navigator:

1. Start Netscape Navigator.
2. Click **Edit** → **Preferences**.
3. Click the **Roaming Access** category.
4. Click the **Item Selection** tab.
5. Uncheck **Cookies**.
6. Click **OK**.

Making HTML Applications Testable

To make HTML applications testable:

- Verify that the HTML extension is loaded in Robot.
- Enable HTML testing in Robot.
- For recording on Netscape, enable caching on the server.

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** or **HTML-Navigator** 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.

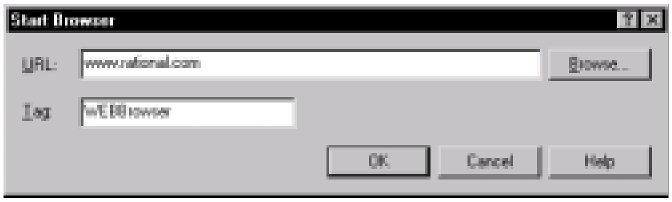
The next time you start Robot, only the extensions for the selected environments are loaded.

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 or Netscape Navigator 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 2-15.
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.


Start Browser

URL: www.rational.com

Tag: WEBBrowser

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 the selected browser and navigates to the specified URL.
9. Continue recording in Robot, as described in *Recording a New GUI Script* on page 2-15.

> **NOTE:** If you start the browser 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.

Enabling Cache for Netscape Recording

If you are recording on Netscape Navigator, you must make sure that the cache is turned on the server that is hosting the web pages or applications you are testing. It is also a good idea to make sure the cache is enabled on your browser. It is enabled by default. If caching is not enabled on the server, recording will not work.

Testing Data in HTML Elements

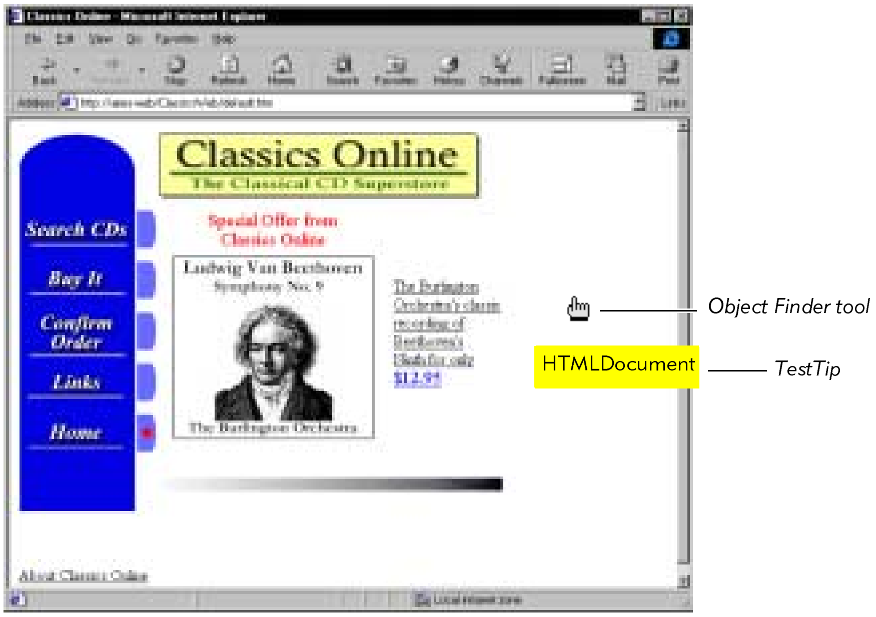
Use the Object Data verification point to test the data in HTML elements. For example, you can use this verification point to test whether a purchase order has been processed or whether a Submit button returns the page that it is supposed to. For more specific information about verification points, see Chapter 4, *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 13-3.
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 4-6.

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 4-10.

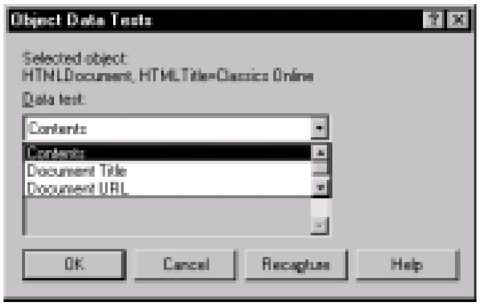
   For example, to test for the existence of a particular string of text on a page (any text within the <BODY> and </BODY> tags), drag the Object Finder tool over the page until **HTMLDocument** appears in the TestTip and in the Selected Object field. You can see examples of these items in the following figure.



   For a list of the HTML elements that you can test, see *How Robot Maps HTML Elements* on page 13-8.

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**.



Object Data Tests

Selected object:
HTMLDocument, HTMLTitle=Classics Online

Data test:
Contents

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 13-10.

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 4-19.

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 4-21.

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	<INPUT type=Submit> <INPUT type=Reset> <INPUT type=Button> <BUTTON>	Used for elements in forms created with the <INPUT> tag where the type attributed is either Submit or Reset.
CheckBox	<INPUT type=Checkbox>	Used for elements in forms created with the <INPUT> tag where the type attributed is Checkbox.
RadioButton	<INPUT type=Radio>	Used for elements in forms created with the <INPUT> tag where the type attribute is Radio.
ComboBox	<SELECT size=1> <OPTION> . . . </SELECT>	Used for elements in forms created with the <SELECT> tag where the size attribute is equal to one.
ListBox	<SELECT size=>*n*> <OPTION> . . . </SELECT>	Used for elements in forms created with the <SELECT> tag where the size attribute is greater than one.
EditBox	<INPUT type=Text> <INPUT type=TextArea> Text is for single line controls. TextArea is for multiline controls.	Used for elements in forms created with the <INPUT> tag where the type attribute is equal to Text or TextArea.
HTMLLink	<A> . . . </A>	Used for anchor elements.
HTMLImage	<IMG>	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 A, *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.

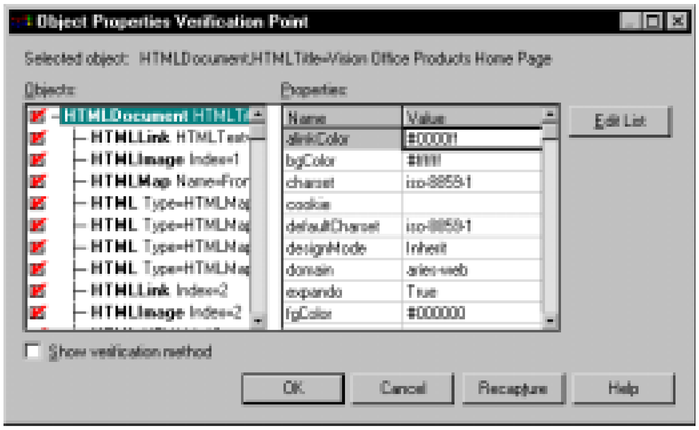
The Object Properties verification point provides you with information about more than 20 properties for each HTML element. Some properties provide you with the same information as a data test. For example, a radio button's Value property provides you with the same information as a Contents data test.

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 13-3.
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 4-6.

5. Select the element to test and then click **OK**. The element's properties are displayed as follows:



Object Properties Verification Point

Selected object: HTMLDocument,HTMLTitle=Vision Office Products Home Page

Objects:

- HTMLDocument HTMLTi
  - HTMLLink HTMLText
  - HTMLImage Index=1
  - HTMLMap Name=Fror
  - HTML Type=HTMLMa
  - HTML Type=HTMLMa
  - HTML Type=HTMLMa
  - HTML Type=HTMLMa
  - HTMLLink Index=2
  - HTMLImage Index=2

Properties:

Name	Value
alinkColor	#0000ff
bgColor	#ffffff
charset	iso-8859-1
cookie	
defaultCharset	iso-8859-1
designMode	Inherit
domain	aries-web
expando	True
fgColor	#000000

6. Click **OK** to insert the verification point.

Playing Back Scripts in Netscape Navigator

With Robot, you can now record scripts in both Internet Explorer and Netscape Navigator, and play them back in either browser. Netscape playback requires at minimum a 200 MHz Pentium with at least 64 MB of RAM.

If you are recording and/or playing back on Netscape Navigator, you must make sure that the cache is turned on on the server that is hosting the web pages or applications you are testing. It is also a good idea to make sure the cache is enabled on your browser. It is enabled by default. If caching is not enabled on the server, record and playback will not work.

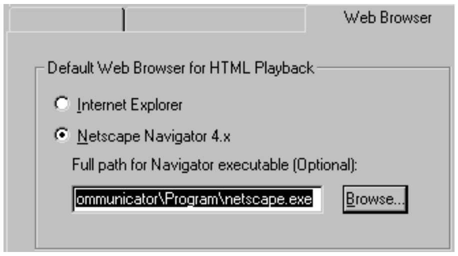
Configuring Robot for Netscape Playback

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.



Default Web Browser for HTML Playback

Internet Explorer [✓ Netscape Navigator 4.x]

Full path for Navigator executable (Optional): ommunicator\Program\netscape.exe

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.

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
- 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 record and playback requires a Web server that allows disk caching. Cache must be enabled on the server for Netscape record and playback to work.

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 log 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 2-21.

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" ><img src="lookpix.gif"  border =0  alt="Lookup a
document"></A>

<A HREF="search.htm"  ><img ID="SearchButton" src="searchpix.gif"
border=0></A>
```

For more information about recognition methods, see the *SQABasic Language Reference*.

▸ ▸ ▸ C H A P T E R 14

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, 1.3	**Microsoft Java SDK 3.1**	**Netscape 4.x (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 to 1.3
- Sun Java Runtime Environment (JRE) 1.1.4 to 1.3
- Microsoft JVM (jview) 3.1

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
- Oracle Class

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 14-7.

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.

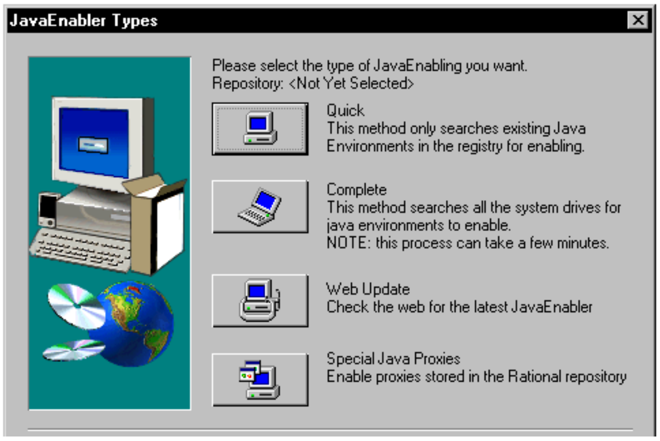
Running the Java Enabler

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.



Please select the type of JavaEnabling you want.
Repository: <Not Yet Selected>

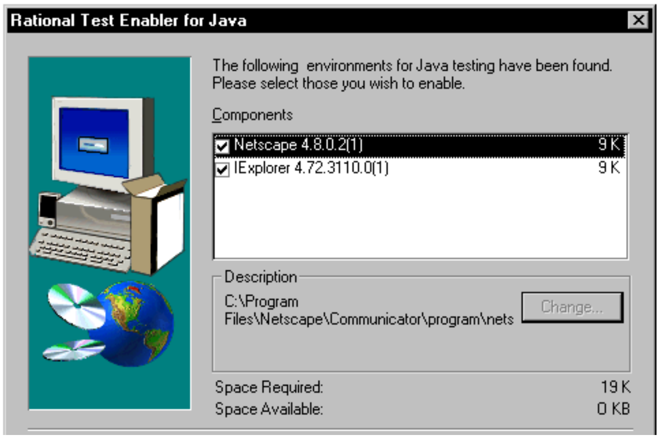
Quick
This method only searches existing Java Environments in the registry for enabling.

Complete
This method searches all the system drives for java environments to enable.
NOTE: this process can take a few minutes.

Web Update
Check the web for the latest JavaEnabler

Special Java Proxies
Enable proxies stored in the Rational repository

4. Select the environments to enable.



The following environments for Java testing have been found.
Please select those you wish to enable.

Components

☑ Netscape 4.8.0.2(1)	9 K
☑ IExplorer 4.72.3110.0(1)	9 K

Description
C:\Program Files\Netscape\Communicator\program\nets

Space Required: 19 K
Space Available: 0 KB

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 14-2.

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	Updates 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	Updates 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	▶ Copies the sqarobot.jar file to the Jre\Lib\Ext directory. ▶ Updates the accessibility.properties file to reference the Robot runtime monitor class.
Java 2 SDK, Standard Edition, V 1.3	▶ Copies the sqarobot.jar file to the Jre\Lib\Ext directory. ▶ Updates 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	▶ 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 14-5.

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

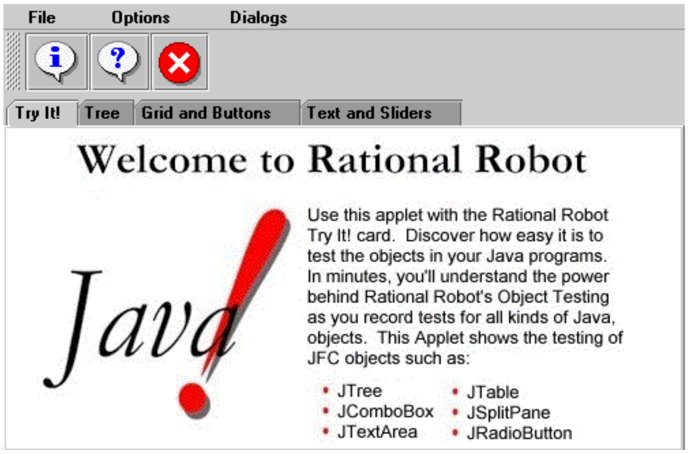
To install the Swing foundation classes on a computer running Windows 98 or Windows 95:

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**.



Welcome to Rational Robot

Use this applet with the Rational Robot Try It! card. Discover how easy it is to test the objects in your Java programs. In minutes, you'll understand the power behind Rational Robot's Object Testing as you record tests for all kinds of Java, objects. This Applet shows the testing of JFC objects such as:

- JTree
- JComboBox
- JTextArea
- JTable
- JSplitPane
- JRadioButton

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 4, *Creating Verification Points in GUI Scripts* and the Robot Help.

To test a Java component's data:

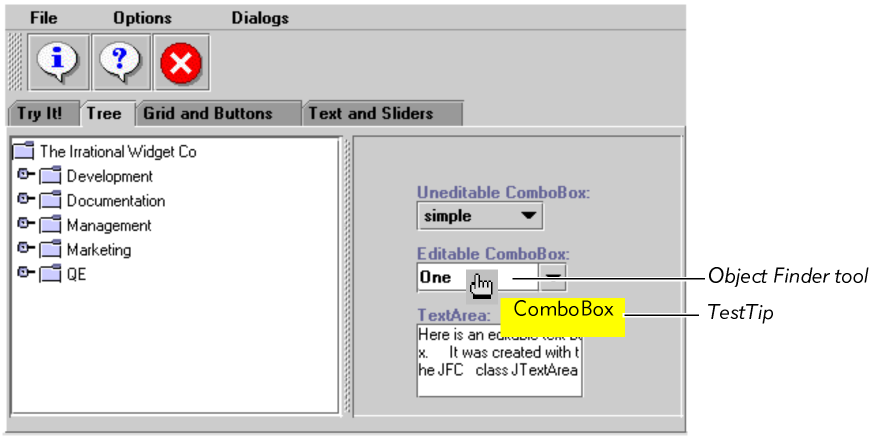
1. Start recording in Robot. For details, see *Recording a New GUI Script* on page 2-15.
2. Open the Java applet or application that you want to test.

   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 3-1.
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 4-6.

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 4-10.

   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.

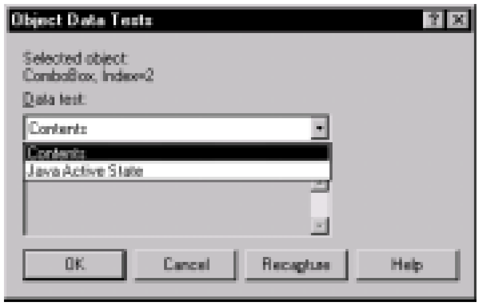


7. Release the mouse button.

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.



Selected object:
ComboBox, Index=2
Data test:
Contents
Contents
JavaActiveState

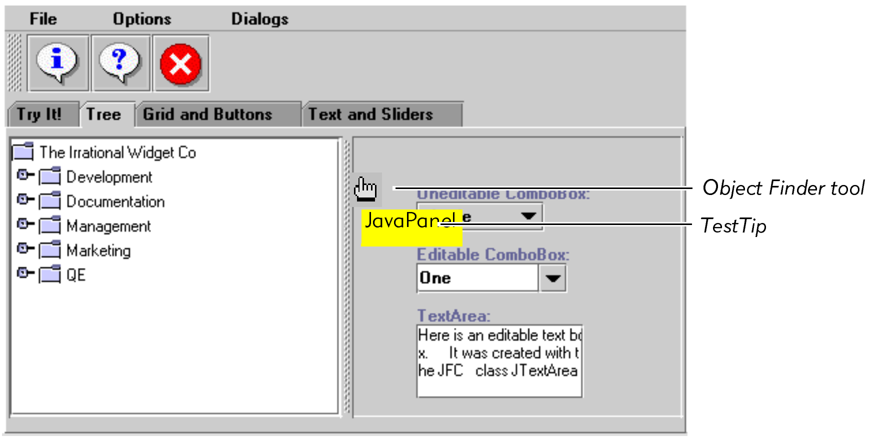
10. Complete the verification point as usual.

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 14-10.
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 4-10.



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 EditBox, 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 14-10.

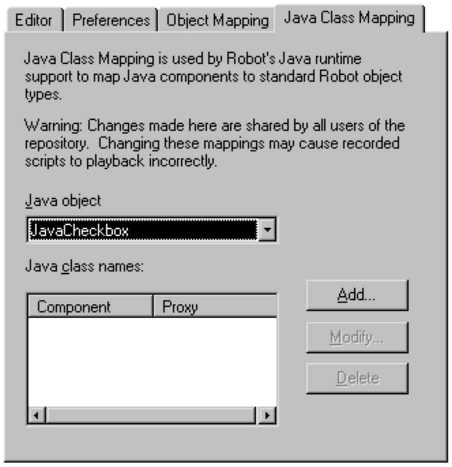
Support for Custom Java Components

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:



Java Class Mapping is used by Robot's Java runtime support to map Java components to standard Robot object types.

Warning: Changes made here are shared by all users of the repository. Changing these mappings may cause recorded scripts to playback incorrectly.

Java object: JavaCheckbox

Java class names:

Component	Proxy

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\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	Captures and displays the menu header and items for each menu.
	Java Active State	Captures and displays the currently selected menu item.
	Menu Test	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.

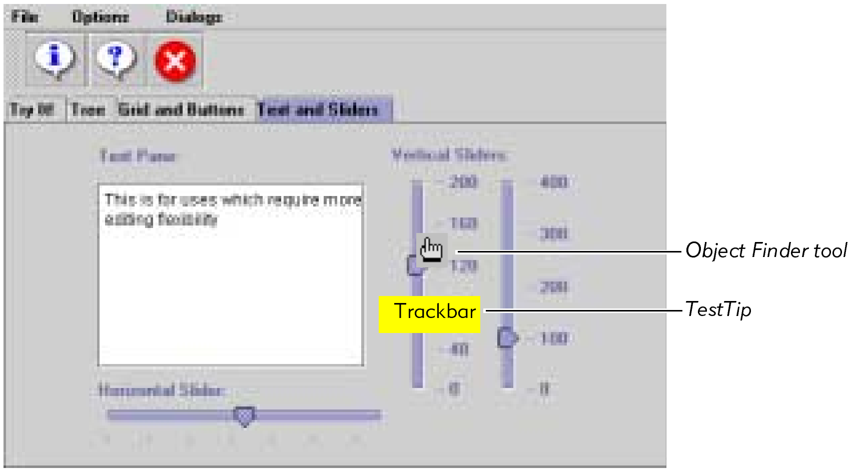
Properties describe a component's characteristics such as its appearance, state, behavior, and data. The Rational Object Testing technology inspects and verifies all properties of the components in your application, including hidden properties that cannot be tested manually.

To test the properties of Java components:

1. Start recording in Robot. For details, see *Recording a New GUI Script* on page 2-15.
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 4-6.

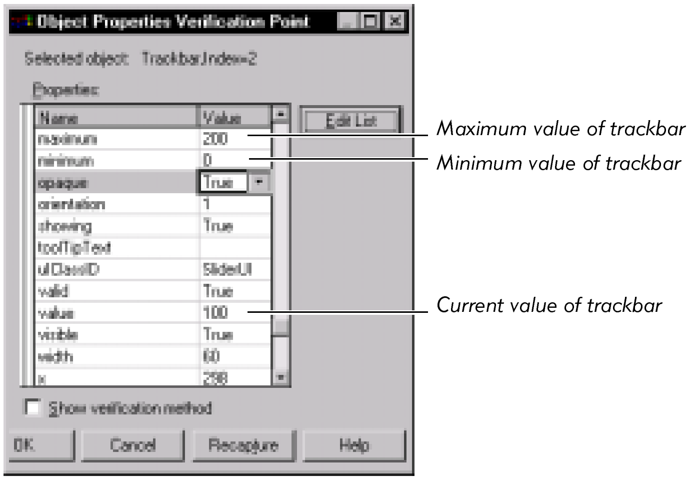
5. In the Select Object dialog box, drag the Object Finder tool over the page until the component that you want to test appears in the TestTip, as described in *Selecting and Identifying the Object to Test* on page 4-10.

   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.



6. Release the mouse button. If the Select Object dialog box is still open, click **OK**.

   The object's properties appear:


Object Properties Verification Point
Edit List
Maximum value of trackbar
Minimum value of trackbar
Current value of trackbar
OK
Cancel
Recapture
Help

Selected object: Trackbar,Index=2

Properties:

Name	Value
maximum	200
minimum	0
opaque	True
orientation	1
showing	True
toolTipText	
uiClassID	SliderUI
valid	True
value	100
visible	True
width	60
x	298

☐ Show verification method

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.

▸ ▸ ▸ C H A P T E R 15

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
- 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.

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, *Value-Based Recording*.)
`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, *Value-Based Recording*.)
`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.

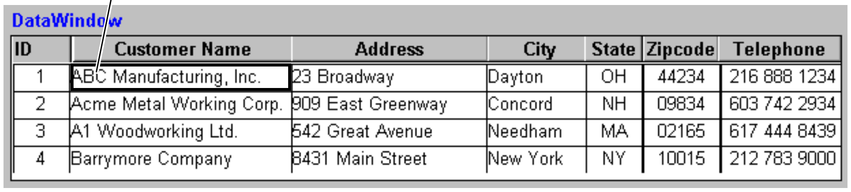
Robot records coordinates for all other DataWindow presentation styles and for DropDownListBoxes.

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.

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 ...



DataWindow

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	09834	603 742 2934
3	A1 Woodworking Ltd.	542 Great Avenue	Needham	MA	02165	617 444 8439
4	Barrymore Company	8431 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.

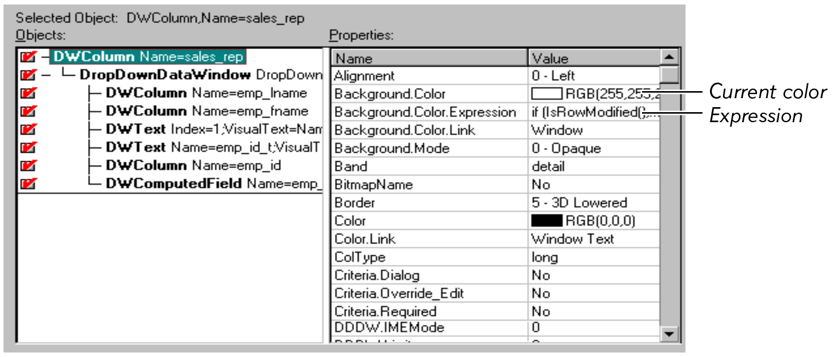
Testing an Expression Value of a DataWindow Property

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.)



Selected Object: DWColumn,Name=sales_rep

Objects:

```
DWColumn Name=sales_rep
 └ DropDownDataWindow DropDown
    ├ DWColumn Name=emp_lname
    ├ DWColumn Name=emp_fname
    ├ DWText Index=1;VisualText=Nam
    ├ DWText Name=emp_id_t;VisualT
    ├ DWColumn Name=emp_id
    └ DWComputedField Name=emp_
```

Properties:

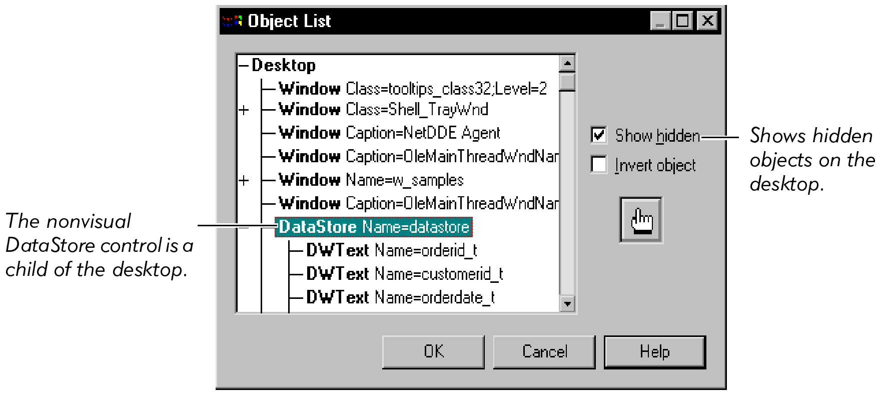
Name	Value
Alignment	0 - Left
Background.Color	⬜ RGB(255,255,2
Background.Color.Expression	if (IsRowModified(),...
Background.Color.Link	Window
Background.Mode	0 - Opaque
Band	detail
BitmapName	No
Border	5 - 3D Lowered
Color	⬛ RGB(0,0,0)
Color.Link	Window Text
ColType	long
Criteria.Dialog	No
Criteria.Override_Edit	No
Criteria.Required	No
DDDW.IMEMode	0

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).



```
–Desktop
   –Window Class=tooltips_class32;Level=2
+  –Window Class=Shell_TrayWnd
   –Window Caption=NetDDE Agent
   –Window Caption=OleMainThreadWndNar
+  –Window Name=w_samples
   –Window Caption=OleMainThreadWndNar
   –DataStore Name=datastore
      –DWText Name=orderid_t
      –DWText Name=customerid_t
      –DWText Name=orderdate_t
```

[✓] Show hidden
[ ] Invert object

Shows hidden objects on the desktop.

The nonvisual DataStore control is a child of the desktop.

For information about selecting objects from the Windows desktop, see *Selecting and Identifying the Object to Test* on page 4-10.

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**.

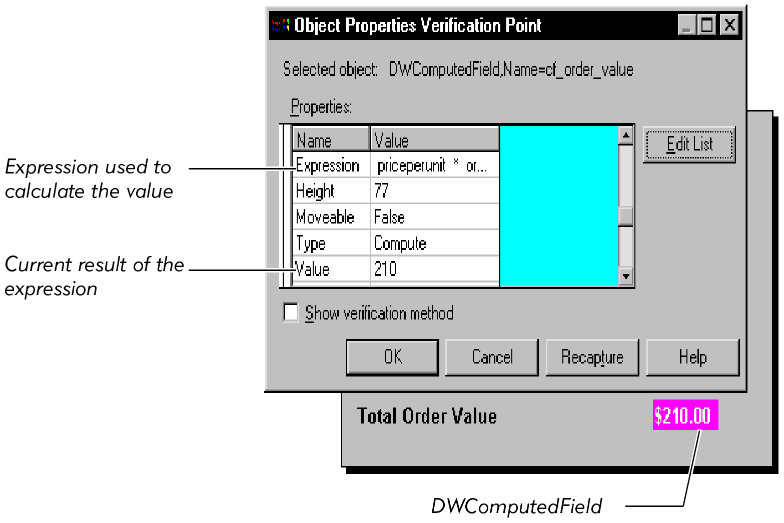
   The Object Data Tests dialog box appears.
4. To capture the data stored in the *child dropdown* of the DataWindow or ListBox, select the **DropDown Contents** data test.

   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.


Object Properties Verification Point
Expression used to calculate the value
Current result of the expression
Show verification method
Edit List
OK
Cancel
Recapture
Help
Total Order Value
$210.00
DWComputedField

Selected object: DWComputedField,Name=cf_order_value

Properties:

Name	Value
Expression	priceperunit * or...
Height	77
Moveable	False
Type	Compute
Value	210

▸ ▸ ▸ C H A P T E R 16

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 7.5. 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 A, *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.

(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.

▸ ▸ ▸ C H A P T E R 17

Testing Delphi Applications

This chapter describes the Delphi 3.0, 4.0, and 5.0 support in Rational Robot. It includes the following topics:

- About Rational Robot support for Delphi
- Making Delphi applications testable
- Testing a component's properties
- Testing a component's data

About Rational Robot Support for Delphi

With Rational Robot, you get comprehensive support for testing your Delphi applications. 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 Delphi component by its internal name.

NOTE: Robot supports the testing of applications built with Delphi 3.0, 4.0, and 5.0 for Windows NT, Windows 95, Windows 98, and Windows 2000.

Use Robot to test Delphi and third-party components, including:

- Visual Component Library (VCL) components
- Win 32 controls
- ActiveX controls
- Data-aware components
- Nonvisual components
- Internet-enabled controls
- Visual inheritance forms

Making Delphi Applications Testable

Before you can test your Delphi applications, you must install the Rational Object Testing Library for Delphi and the Rational Test Delphi Enabler. You then need to run the Enabler, which adds one line of code (`SQASrvr,`) to your project, and then you need to recompile your project in Delphi to make it Robot-testable.

NOTE: You may freely distribute the Rational Object Testing Library for Delphi and the Enabler. The SQAOBTST 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.

Installing the Rational Object Testing Library and Enabler

Be sure to install Delphi 3.0, 4.0, or 5.0 for Windows NT, 95, 98, or 2000 before you install the Object Testing Library for Delphi and the Enabler.

To install the Library and Enabler:

1. Insert the Rational Suite CD-ROM into your CD-ROM drive.
2. Do one of the following:
   – From the Windows taskbar: Choose **Start** → **Run**.
   – From Program Manager: Choose **File** → **Run**.
3. Type `*drive*:\SETUP.EXE` and choose **OK** to start the Rational Setup wizard.
4. When the list of Rational Test Enablers appears, choose **Rational Test Delphi Enabler**.
5. Follow the onscreen instructions to complete the installation.

Running the Rational Test Delphi Enabler

After you install the Object Testing Utilities, you need to run the Delphi Enabler, which adds one line of code to your project, and then you need to compile the project.

You only need to run the Enabler once for each project you want to make testable.

NOTE: The Rational Object Testing Library is not visible and is non-intrusive, and there are no license restrictions on it. Therefore, you can leave it in the application when you distribute it.

Adding the Rational Object Testing Library

To add the Object Testing Library to your project:

1. Click **Start** → **Programs** → **Borland Delphi** → **Rational Test Delphi Project Converter** to open the Enabler.
2. Click **Browse** under **Delphi Project**. Select the project you want to make Robot-testable and click **OK**.
3. Choose the correct compiler version, depending on which version of Delphi you have installed on your computer.
4. Select **Add Rational Object Testing Library**.
5. Optionally, uncheck **Backup Project File** if you don't want a backup file to be created before the project file is converted.
6. Optionally, uncheck **Launch Delphi After Conversion** if you don't want Delphi to be automatically started after the conversion. (If this is not checked, you will need to start Delphi manually after the conversion.)
7. Click **Convert**. The Enabler will:
   - Optionally create a backup file of the project.
   - Add the line `SQASrvr,` to your project file (after the `uses` line).
   - Optionally start Delphi.
8. Click the **Close** button in the Enabler. If an information dialog pops up, click **Yes** to reload the project.
9. In Delphi, recompile your project.

Removing the Rational Object Testing Library

The Rational Object Testing Library is not visible and is non-intrusive, and there are no license restrictions on it. Therefore, you can leave it in the application when you distribute it.

However, if you choose to remove the Object Testing Library from your project:

1. Click **Start** → **Programs** → **Borland Delphi** → **Rational Test Delphi Project Converter** to open the Enabler.
2. Click **Browse** under **Delphi Project**. Select the project from which you want to remove the Library and click **OK**.
3. Choose the correct compiler version, depending on which version of Delphi you have installed on your computer.
4. Select **Remove Rational Object Testing Library**.

5. Optionally, uncheck **Backup Project File** if you don't want a backup file to be created before the project file is converted.
6. Optionally, uncheck **Launch Delphi After Conversion** if you don't want Delphi to be automatically started after the conversion. (If this is not checked, you will need to start Delphi manually after the conversion.)
7. Click **Convert**. The Enabler will:
   – Optionally create a backup file of the project.
   – Remove the line `SQASrvr,` from your project file (after the `uses` line).
   – Optionally start Delphi.
8. Click the **Close** button in the Enabler. If an information dialog pops up, click **Yes** to reload the project.
9. In Delphi, recompile your project.

Testing a Component's Properties

There are two methods you can use to test the properties of a Delphi component. You can use either method to test all properties shown by the Delphi Object Inspector.

▸ **Object Properties verification point** – Use to test properties while recording or editing a script.

  For information about testing properties, see the Robot Help.

▸ **Object Scripting commands** – Use to test properties programmatically while editing a script.

  For information about the Object Scripting commands, see chapter 5 of the *SQABasic Language Reference*.

NOTE: Before you can test a Delphi application, you need to run the Rational Test Delphi Enabler. For instructions, see *Making Delphi Applications Testable* on page 17-2.

Testing a Component's Data

Use the Object Data verification point to test the data in Delphi components. For instructions, see *Creating an Object Data Verification Point* on page 4-8.

You can use the Object Data verification point to define additional data tests. For instructions, see *Defining a Data Test* on page 4-11.

NOTE: Before you can test a Delphi application, you need to run the Rational Test Delphi Enabler. For instructions, see *Making Delphi Applications Testable* on page 17-2.

▸ ▸ ▸ Part VI

Appendixes

▸▸▸ APPENDIX A

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

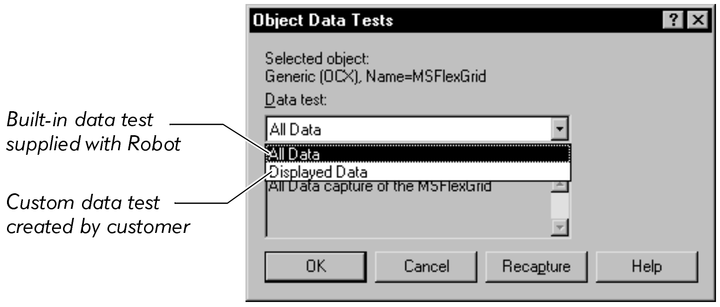
For information about the Object Data verification point, see *Object Data verification point* in the Robot Help Index.

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).


Built-in data test supplied with Robot
Custom data test created by customer
Object Data Tests
Selected object:
Generic (OCX), Name=MSFlexGrid
Data test:
All Data
All Data
Displayed Data
All Data capture of the MSFlexGrid
OK
Cancel
Recapture
Help

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:

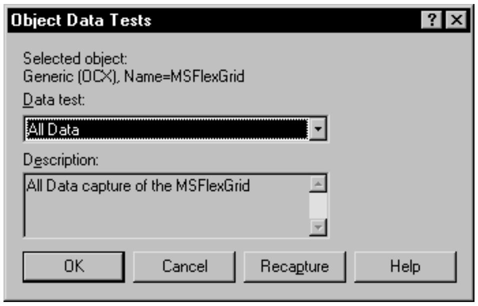
- Access to the documentation or Help that came with the object that you want to test.
- A good understanding of the object's properties and how the properties relate.
- A good understanding of the Object Data verification point. For information, see *Object Data verification point* in the Robot Help 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.


Object Data Tests

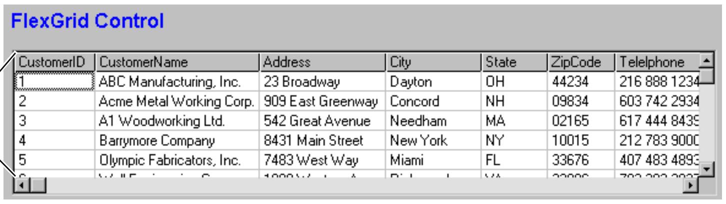
Selected object:
Generic (OCX), Name=MSFlexGrid

Data test: All Data

Description:
All Data capture of the MSFlexGrid

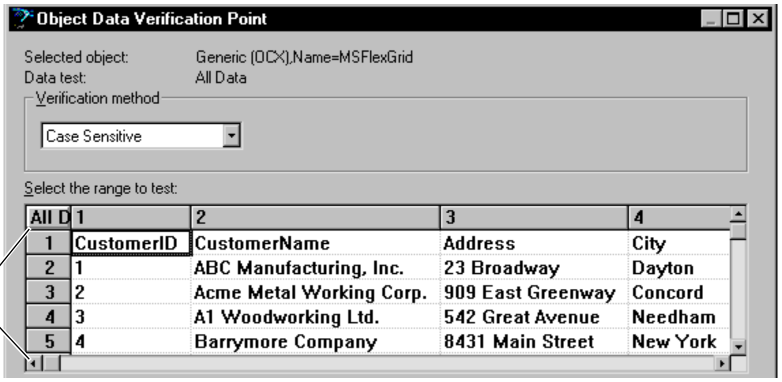
When you use this data test, Robot captures the data from every cell in the MSFlexGrid control, as shown in the following figures.

Robot cycles through every column and row in the MSFlexGrid control to extract the values from each cell...


FlexGrid Control

CustomerID	CustomerName	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	09834	603 742 2934
3	A1 Woodworking Ltd.	542 Great Avenue	Needham	MA	02165	617 444 843
4	Barrymore Company	8431 Main Street	New York	NY	10015	212 783 900
5	Olympic Fabricators, Inc.	7483 West Way	Miami	FL	33676	407 483 489

... and then displays the data in the Robot data grid.


Object Data Verification Point

Selected object: Generic (OCX),Name=MSFlexGrid
Data test: All Data

Verification method: Case Sensitive

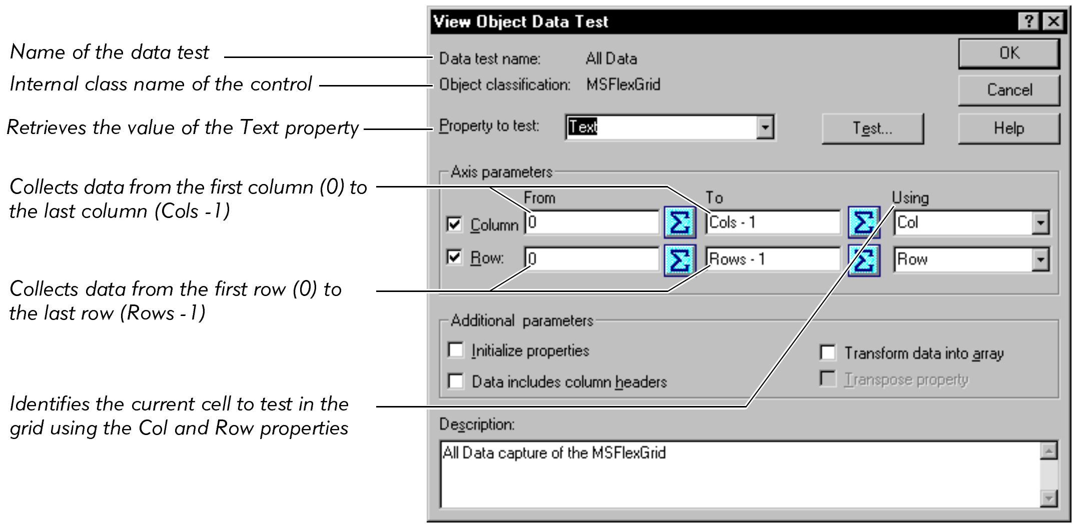
Select the range to test:

All D	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
4	3	A1 Woodworking Ltd.	542 Great Avenue	Needham
5	4	Barrymore Company	8431 Main Street	New York

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.

The following figure shows the main information in the dialog box:



Name of the data test

Internal class name of the control

Retrieves the value of the Text property

Collects data from the first column (0) to the last column (Cols -1)

Collects data from the first row (0) to the last row (Rows -1)

Identifies the current cell to test in the grid using the Col and Row properties

View Object Data Test

Data test name: All Data

Object classification: MSFlexGrid

Property to test: Text

Axis parameters

	From	To	Using
☑ Column	0	Cols - 1	Col
☑ Row:	0	Rows - 1	Row

Additional parameters

☐ Initialize properties

☐ Data includes column headers

☐ Transform data into array

☐ Transpose property

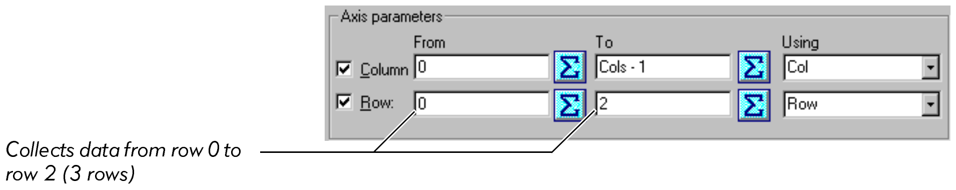
Description:

All Data capture of the MSFlexGrid

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

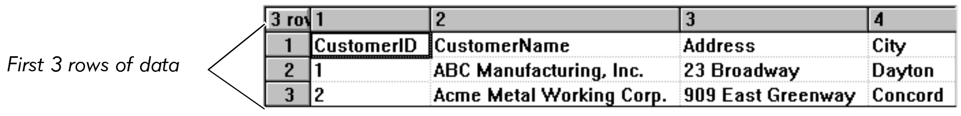
Suppose you want to test only the first three rows in the control instead of all the rows. You could do this by making a copy of the *All Data* test, and then editing the copy so that the rows range from 0 to 2.



Axis parameters

	From	To	Using
☑ Column	0	Cols - 1	Col
☑ Row:	0	2	Row

Collects data from row 0 to row 2 (3 rows)



First 3 rows of data

3 ro	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.

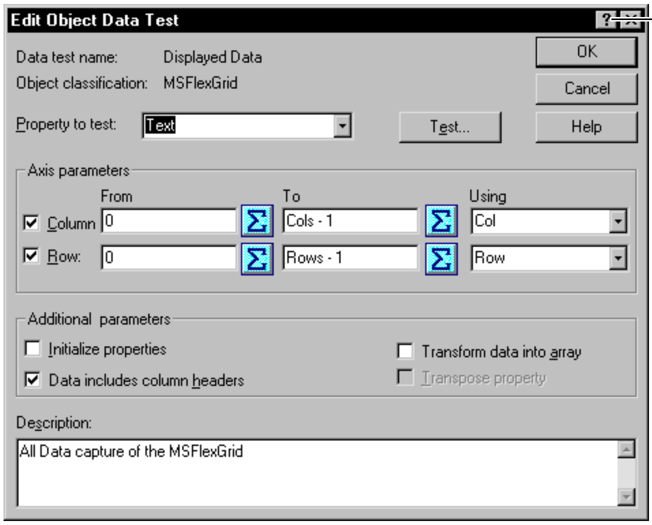
     As you move the Object Finder tool over an object, the object type appears in the yellow TestTip.
   - Click **Browse** to open the Object List dialog box, select the object from the list, and click **OK**.

     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 2-20.)
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:



Data test name: Displayed Data
Object classification: MSFlexGrid
Property to test: Text

	From	To	Using
☑ Column	0	Cols - 1	Col
☑ Row:	0	Rows - 1	Row

☐ Initialize properties
☑ Data includes column headers
☐ Transform data into array
☐ Transpose property

Description:
All Data capture of the MSFlexGrid

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.

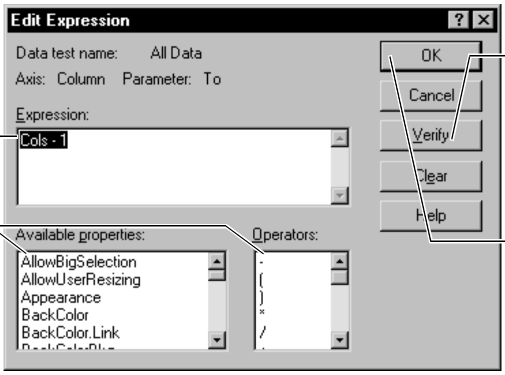
   An expression is a single value or property, or a combination of values, properties, and operators.

If you click the **Expression** button, the Edit Expression dialog box appears.

Do the following if you clicked **Expression**:

a. Type an expression here ...

... or build an expression by double-clicking in these lists.



*b. Click **Verify** to check the syntax of the expression. Click **OK** in the message box. If the syntax is invalid, the incorrect area is highlighted.*

*c. Click **OK** to accept the expression. If you did not verify the expression, Robot verifies it now. Click **OK** in the message box. If the syntax is invalid, the incorrect area is highlighted.*

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.

▸ ▸ ▸ APPENDIX B

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

```
rtrobo.exe [scriptname] [/user userid] [/password password]
  [/project full path and projectname] [/play]
  [/purify] [/quantify] [/coverage] [/close] [/nolog]
```

Syntax Element	Description
rtrobo.exe	Rational Robot executable file.
scriptname	Name of the script to run.
/user *userid*	User name for log in.
/password *password*	Optional password for log in. Do not use this parameter if there is no password.
/project *full path and projectname*	Name of the project that contains the script referenced in *scriptname* preceded by its full path.
/play	If this keyword is specified, plays the script referenced in *scriptname*. If not specified, the script opens in the editor.
/purify	Used with /play. Plays back the script referenced in *scriptname* under Rational Purify.
/quantify	Used with /play. Plays back the script referenced in *scriptname* under Rational Quantify.
/coverage	Used with /play. Plays back the script referenced in *scriptname* 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 Login dialog box:

```
/user userid
/password password
/project full path and projectname
```

EXAMPLE

```
rtrobo.exe VBMenus /user admin /project C:\Sample Files\Projects\Default
/play /close
```

In this example, the user “admin” opens the script “VBMenus”, which is in the project “Default” located in the directory “c:\Sample Files\Projects”. The script is opened for playback, and then it is closed when playback ends. The results are logged.

▸ ▸ ▸ Index