

Using Rational LoadTest

Version 2000.02.10

Using Rational LoadTest

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

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

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

Revised 04/2000

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

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

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

P/N 800-023369-000

▶ ▶ ▶ Contents

Preface

Resources	xiii
Using Help	xiv
Contacting Rational Technical Publications	xiv
Contacting Rational Technical Support	xv

1 What Is LoadTest?

About LoadTest	1-1
Why Use LoadTest?	1-2
LoadTest Basics	1-3
Scripts	1-4
The Scripting Languages	1-4
Sessions	1-5
Master and Agent Computers	1-5
Schedules	1-6
Types of Users	1-6
Types of Tests	1-7
Testing Response Times	1-11
Client Response Time	1-11
Server Response Time	1-11
The LoadTest Environment	1-12
Logging into LoadTest	1-14

2 Before You Begin

About Performance and Functional Tests	2-1
Planning Performance Tests	2-2
Setting Pass and Fail Criteria for Performance Tests	2-3
Identifying Performance Testing Requirements	2-3
Designing a Realistic Workload	2-4
Designing Performance Tests	2-5
Examples of Performance Tests	2-6
Planning Virtual User Scripts	2-11
Setting Virtual User Recording Options	2-12
Virtual User Recording Considerations	2-12
Modifying Virtual User Scripts	2-13
Correcting Errors in Virtual User Scripts	2-15
Analyzing Performance Results	2-15
Planning Functional Tests	2-19
Distributed Functional Testing	2-19
Setting Pass and Fail Criteria for Functional Tests	2-20
Identifying Functional Testing Requirements	2-20
Scheduling Functional Tests	2-20
Designing Functional Tests	2-21
GUI Recording Considerations	2-22
Modifying GUI Scripts	2-23
Correcting Errors in GUI Scripts	2-23
Analyzing Functional Results	2-24

3 Setting Recording Options

About Virtual User Recording	3-1
Setting the Recording Method	3-2
API Recording	3-4
Network Recording	3-4
Proxy Recording	3-6
Setting Script Generation Options	3-9
Modifying the Contents of a Script	3-10
Setting Filtering Options	3-16
Providing HTTP, Oracle, TUXEDO, and IIOP Information	3-20

Setting General Recording Options	3-25
Autonaming Prefixes	3-25
Start Application	3-25
Setting the Recorder Window	3-26
Defining a Client or Server Computer	3-27
Removing a Computer or Port	3-29
Authenticating Login	3-29
When to Modify the Authentication Datapool	3-30
Modifying the Authentication Datapool with TestManager	3-30
Modifying the Authentication Datapool During Recording	3-31
Unique Features of the Authentication Datapool	3-32
Managing Proxies	3-32
Starting and Stopping Proxy Service	3-32
Monitoring Proxy Activities	3-34
Deleting Client/Server Pairs	3-34
Deleting a Proxy	3-35
Re-Creating Proxies that Have Been Removed	3-35
4 Recording Virtual User Scripts	
Recording a Session	4-1
What You Can Record in a Session	4-2
Where Files Are Stored	4-2
Restoring Robot During Recording	4-2
Recording a Single Script in a Session	4-3
Using the Floating Toolbars	4-5
If Script Generation Problems Occur	4-5
Providing a Missing Password	4-5
Getting Feedback During Recording	4-7
The Virtual User Recorder During Recording	4-7
The Virtual User Recorder After Recording	4-9
Cancelling Scripts During Recording	4-9
Canceling the Script in a Single-Script Session	4-9
Canceling the Current Script in a Multi-Script Session	4-10
Canceling All Scripts in a Multi-Script Session	4-10

Choosing the Protocols to Include in a Script	4-11
Manually Filtering Protocols	4-11
Playing Back a Script Quickly	4-15
Working with Sessions	4-15
Splitting a Session into Multiple Scripts	4-15
Importing a Session	4-16
Regenerating the Scripts Recorded in a Session	4-17
Viewing Session Properties	4-18
Coding a Virtual User Script Manually	4-20
Creating Library Files	4-20
Defining Script Properties	4-20
How to Define Script Properties in Robot	4-21
Managing Scripts and Sessions	4-21
Finding the Scripts Contained in a Session	4-21
Finding the Session Associated with a Script	4-22
Removing a Script from a Session	4-22
Re-Recording Sessions	4-22
Re-Recording Scripts	4-24
Copying Scripts	4-25
Deleting Scripts and Sessions	4-25
5 Adding Features to Virtual User Scripts	
Timers	5-1
How Timers Work	5-1
Why Use Timers?	5-2
Adding a Timer During Recording	5-2
Adding a Timer During Editing	5-3
Blocks	5-4
Why Use Blocks?	5-5
Adding a Block	5-5
Nesting Blocks	5-6
Synchronization Points	5-7
How Synchronization Points Work	5-7
Why Use Synchronization Points?	5-8

Inserting Synchronization Points	5-9
Scope of a Synchronization Point	5-11
Comments	5-11
Adding Comments During Recording	5-12
Adding Comments During Editing	5-12
Using the Insert Menu	5-12
6 Working with Datapools	
What Is a Datapool?	6-2
Datapool Tools	6-2
Datapool Cursor	6-4
Datapool Limits	6-4
What Kinds of Problems Does a Datapool Solve?	6-5
Planning and Creating a Datapool	6-6
Data Types	6-9
Standard and User-Defined Data Types	6-9
Finding Out What Data Types You Need	6-10
Creating User-Defined Data Types	6-11
Generating Unique Values from User-Defined Data Types	6-12
Generating Multi-Byte Characters	6-13
Using Datapools with Virtual User Scripts	6-13
Creating a Datapool with Robot	6-13
Editing Datapool Column Definitions with Robot	6-21
Editing Datapool Values with Robot	6-23
Using Datapools with GUI Scripts	6-24
Accessing a Datapool from GUI and Virtual User Scripts	6-24
Managing Datapools with TestManager	6-25
Creating a Datapool with TestManager	6-25
Editing Datapool Column Definitions with TestManager	6-32
Editing Datapool Values with TestManager	6-34
Renaming a Datapool	6-35
Copying a Datapool	6-35
Deleting a Datapool	6-35
Importing a Datapool	6-35
Exporting a Datapool	6-37

- Managing User-Defined Data Types6-38
 - Editing User-Defined Data Type Values6-38
 - Editing User-Defined Data Type Definitions6-39
 - Importing a User-Defined Data Type6-41
 - Renaming a User-Defined Data Type6-41
 - Copying a User-Defined Data Type6-42
 - Deleting a User-Defined Data Type6-42
- Generating and Retrieving Unique Datapool Rows6-42
 - What You Can Do to Guarantee Unique Row Retrieval. . . .6-43
- Creating a Datapool Outside Rational Test6-44
 - Datapool Structure6-45
 - Example Using Microsoft Excel6-46
 - Matching Datapool Columns with Script Variables.6-48
 - Maximum Number of Imported Columns6-48
- Creating a Column of Values Outside Rational Test6-48
 - Step 1. Create the File6-49
 - Step 2. Assign the File's Values to the Datapool Column . . .6-49
 - Generating Unique Values.6-50

7 Designing Schedules

- About Schedules7-2
- Creating a Schedule7-3
 - Creating a Schedule from a Blank Schedule.7-3
 - Creating a Schedule from a Session7-4
- Inserting User Groups into a Schedule7-5
- Inserting Scripts into a Schedule.7-8
- Inserting Other Items into a Schedule7-10
 - Inserting a Scenario7-10
 - Inserting an Executable7-12
 - Setting Schedule Items to Run in Different Sequences. . . .7-13
 - Types of Selectors.7-15
 - Inserting a Selector7-18
 - Inserting a Delay.7-19
 - Setting Schedule Items to Run at Certain Rates.7-21

Inserting a Transactor	7-22
Inserting a Synchronization Point	7-25
Opening a Schedule	7-28
Editing a Script	7-29
Editing the Properties of a Script	7-30
Editing the Text of a Script	7-31
Editing a Schedule	7-31
Editing the Properties of a Schedule	7-32
Cutting and Pasting Items	7-33
Deleting Items	7-33
Replacing Items	7-33
Editing Items	7-34
Editing Information for All User Groups	7-35
Editing the Settings of an Agent Computer	7-37
Editing the User Settings	7-38
Viewing Schedules with the Asset Browser	7-50
Deleting a Schedule	7-51
Renaming a Schedule	7-51
Using Events and Dependencies to Coordinate Execution	7-52
Setting an Event	7-53
Setting a Dependency on an Event	7-54
Setting Shared Variables	7-55
Printing and Exporting a Schedule	7-57
Saving a Schedule	7-58
Checking a Schedule	7-59
Checking Agent Computers	7-60
Controlling Runtime Information of a Schedule	7-60
Controlling How a Schedule Terminates	7-64
Running a Schedule	7-66

8 Monitoring Schedules

About Monitoring Schedules	8-2
Displaying the Schedule Views	8-4
Displaying the Histogram Views	8-5
Standard Histograms	8-6
GUI Histograms	8-7
SQL Histograms	8-8
HTTP Histograms	8-8
IIO P Histograms	8-9
Zooming In on Histogram Bars	8-9
Displaying the User Views	8-12
Compact User View	8-13
Results User View	8-14
Source User View	8-15
Message User View	8-16
Full User View	8-16
Displaying the Shared Variables View	8-17
Displaying the Script View	8-18
Displaying the Sync Points View	8-19
Displaying the Users Waiting on a Synchronization Point	8-20
Releasing a Synchronization Point	8-20
Displaying the Computer View	8-21
Viewing Resource Usage During a Run	8-21
Graphing Resource Usage During a Run	8-23
Viewing Computers at the Start or End of a Run	8-24
Displaying the Transactor View	8-25
Displaying the Group Views	8-26
Displaying the Users in a Group	8-27
Filtering and Sorting Views	8-27
Sorting the Users Displayed in a User View	8-27
Filtering a User View	8-29
Filtering a Group View	8-30
Restoring the Default Views	8-31
Changing the Value of a Shared Variable	8-31
Debugging a VU Script	8-33

Changing Monitor Defaults	8-34
Configuring Custom Histograms	8-36
Controlling the Schedule During a Run	8-38
Suspending and Resuming Virtual Users in a Schedule	8-38
Stopping a Schedule	8-39
9 Analyzing Results	
About LoadTest Reports	9-2
Running a Report and Viewing Log Files.....	9-4
Viewing the Log Files	9-4
Running a Report from the Report Bar	9-6
Running a Report from the Menu Bar	9-6
Printing a Report	9-7
Printing Report Output	9-8
Copying Report Output to the Clipboard	9-8
Copying a Report or Its Output within LoadTest	9-8
Renaming a Report or Report Output	9-10
Deleting a Report or Report Output.....	9-11
Exporting Report Output	9-12
Comparing the Output of Performance Reports.....	9-13
Customizing Reports.....	9-15
Filtering Report Data	9-16
Setting Advanced Options	9-22
Defining a Compare Report.....	9-28
Changing a Graph's Appearance or Type	9-30
Editing the Properties of a Report or Report Output	9-34
Changing Report Defaults.....	9-35
Changing the Reports that Run Automatically	9-35
Changing the Reports that Run from the Report Bar	9-37

Types of Reports	9-38
Analog	9-38
Compare	9-40
Performance	9-44
Response	9-46
Status	9-48
Trace	9-49
Usage	9-52
A Working With Toolbars	
Viewing Information About Toolbar Buttons	A-1
Displaying Toolbars	A-2
Anchoring and Floating Toolbars	A-2
Setting Toolbar Options	A-3
Adding, Deleting, and Moving Toolbar Buttons	A-3
Creating Your Own Toolbar	A-4
Resetting and Deleting Toolbars	A-4
B Configuring Master and Agent Computers	
Running More Than 245 Users	B-1
Running More Than 1000 Users	B-2
Running More Than 1000 Users on One NT Computer	B-2
Running More Than 24 Users on a UNIX Agent	B-3
Controlling TCP Port Numbers	B-3
Setting Up IP Aliasing	B-5
C Standard Datapool Data Types	
Standard Data Type Table	C-1
Data Type Ranges	C-9

Glossary

Index

▶ ▶ ▶ Preface

Rational LoadTest is a sophisticated tool for automating performance and distributed functional tests on client/server systems. You can use LoadTest to test Web, database, or transaction servers connected to clients on the network. LoadTest lets you run tests that emulate hundreds, or even thousands, of users on just a few computers.

This guide is intended to help application developers and system testers use LoadTest to create, edit, run, monitor, analyze, and manage automated tests that run across a network.

Other Resources

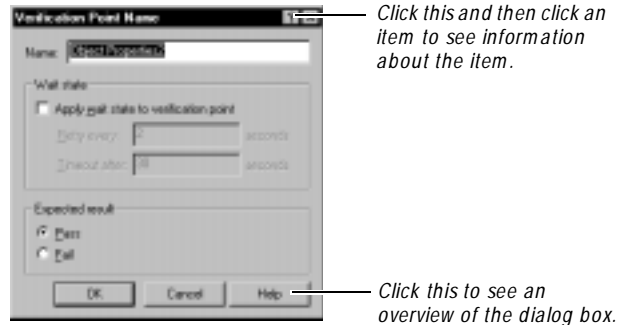
- ▶ This product contains complete online Help. From the main toolbar, choose an option from the **Help** menu.
For information about context-sensitive Help, see the following section.
- ▶ All manuals are available online in PDF format. These online manuals are on the *Rational Solutions for Windows* Online Documentation CD.
- ▶ For information about training opportunities, see the Rational University Web site: <http://www.rational.com/university>.

Using Help

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

Dialog Box Help

Most dialog box Help includes overviews and detailed item information.



Menu Command Help



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

Toolbar Button Help



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

Contacting Rational Technical Publications

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

Contacting Rational Technical Support

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

Rational Technical Support

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

Preface

What Is LoadTest?

This chapter introduces Rational LoadTest. It includes the following topics:

- ▶ About LoadTest
- ▶ LoadTest basics
- ▶ Testing response times
- ▶ The LoadTest environment
- ▶ Logging into LoadTest

About LoadTest

Rational LoadTest is a sophisticated tool for automating performance tests and distributed functional tests on client/server systems. A client/server system includes client applications accessing a database or application server, client applications accessing the TUXEDO TP monitor, or browsers accessing a Web server.

LoadTest is completely integrated with the other components of Rational PerformanceStudio, including:

- ▶ Rational Robot for recording sessions, and for generating and editing scripts
- ▶ Rational TestManager for managing test assets, such as sessions, scripts, schedules, and reports
- ▶ Rational LogViewer for viewing the user log files and the user error files, which contain the emulation data and the errors that occurred while running a VU script

With its intuitive point-and-click graphical user interface, LoadTest lets you play back the scripts you developed with Robot according to a schedule that you create. You can monitor the schedule's progress, and you can analyze the results to see how the client application and the server perform under varying workload and stress conditions.

What Is LoadTest?

LoadTest helps you discover and correct performance problems before you deploy your application in the real world. With LoadTest, you have all of the tools you need to identify, isolate, and analyze performance bottlenecks.

As an automated load testing tool, LoadTest emulates one or many users performing various computing tasks. By replacing actual users with virtual users, LoadTest removes the need for actual users to manually add workload to the server.

Because LoadTest lets you play back the activities of multiple users on a single computer, you can run tests involving hundreds, or even thousands, of users on just a few computers—or on one computer.

Why Use LoadTest?

While LoadTest has a range of uses, it excels in solving performance issues such as the following:

Problem	LoadTest Solution
Does the server perform correctly under load?	Conduct stability and stress testing of the network and servers under maximum workload conditions.
Does the system meet scalability requirements?	Determine the number of users a server can support before the system is released.
What level of performance will the clients achieve? Can the server deliver acceptable response times during simultaneous access by large numbers of users?	Measure the response times of server operations, as seen by the client under varying transaction rates and workload mixes. In addition, you can determine: <ul style="list-style-type: none">▶ Application response time (average case, best case, worst case).▶ How response time varies under different computer configurations.▶ A server's response time when a given number of virtual users are running against it.▶ How rapidly a client's response time falls as the number of virtual users accessing the server increases.▶ The server's hit rate when Web testing is being conducted.▶ The error rate and error breakdown.

(Continued)

Problem	LoadTest Solution
How do the access patterns of database tables affect performance? When is table or row locking a problem?	Run contention tests that analyze throughput and capacity under varying transaction rates, workload mixes, and server configurations.
What was the percentage of improvement for client response times after the last tunable parameter change?	Place reproducible workloads on the server to objectively measure tuning efforts.
Does the latest release of the database server produce the same output and error detection as the previous release?	Test functional regression of database operations.
Which queries cause performance problems?	Isolate queries that perform poorly.

LoadTest Basics

LoadTest lets application developers, system testers, and system integrators test multi-user client/server systems. LoadTest can be used to add maximum workload conditions to the network and server in a client/server environment, or to distribute GUI functional tests over several computers, thereby reducing the total testing time. With LoadTest, you can coordinate multiple computers, as well as emulate multiple virtual users, from a single computer running Windows NT.

Robot records user activities, and then automatically creates a script that represents the user's interactions with the server, as well as all queries and responses.

Before you begin planning and developing tests, you should be familiar with the following LoadTest concepts:

- ▶ Scripts
- ▶ The scripting languages
- ▶ Sessions
- ▶ Master and Agent computers
- ▶ Schedules
- ▶ The types of "users" that you emulate by recording and playing back scripts
- ▶ The types of tests that LoadTest helps you perform

Scripts

Much of your testing effort involves planning and recording scripts. The purpose of planning and recording scripts is to later play them back in a test to emulate user activity.

A **script** has two basic features:

- ▶ A file that can be executed by Robot or by a LoadTest schedule

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

- ▶ A set of properties, such as the type and purpose of the script

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

Two Perspectives on Scripts

When you plan tests in TestManager and when you record tests in Robot, you will probably think of scripts in the context of the script and its properties.

But when you play back a script in a LoadTest schedule, think of that runtime instance of the script as a “user”—an emulated user performing the tasks that you recorded.

The Scripting Languages

The SQABasic and VU scripting languages are used to represent GUI and virtual user emulations. Depending on the type of emulation that you specify, Robot automatically creates the script in one of these languages when you record a script.

GUI scripts – Contain SQABasic code for playing back the GUI activities that you recorded. SQABasic is the language for writing GUI scripts. It uses most of the syntax rules and core commands found in the Microsoft Basic language. If you are familiar with Microsoft Basic or Visual Basic, you are already familiar with much of the SQABasic language. For more information, see the *SQABasic Language Reference*.

Virtual user scripts – Contain VU code for playing back the client/server requests you recorded. The VU language is a C-like scripting language. It shares much of its syntax rules, operators, and library functions with the C language. If you are familiar with the C language, you are already familiar with much of the VU language.

A VU script communicates to the LoadTest system what needs to be done to perform the desired virtual user emulation. For more information, see the *VU Language Reference*.

You cannot mix SQABasic code and VU code in the same script.

Sessions

When you record virtual user scripts, you are actually recording a **session**. A session contains all of the client requests and server responses issued from the time you begin recording until the time you stop recording. During virtual user recording, you can direct Robot to split the session into more than one script.

If you do not split the session into multiple scripts, the session will consist of a single script that you name at the end of recording.

However, splitting a session into scripts is a convenient way to let you treat sections of the recording session differently in a LoadTest schedule. For example, suppose you log into a database server and perform three different transactions. If you split the session so that the login and each transaction is a separate script, you can then set up the schedule to perform many iterations of the transactions but perform the login script only once.

The names of sessions and scripts are independent, although if the session consists of only one script, you should generally give it the same name as the script so you know that the script and the session are associated with each other.

Master and Agent Computers

You coordinate the activities of all your scripts from a single NT computer where LoadTest is running, known as the **Master computer**. From the Master computer, you create, run, and monitor schedules.

During the execution of a test, you play back scripts on the Master computer, or on computers that you have designated as **Agent computers**. You use an Agent computer for the following:

- ▶ **Adding workload to the server** – If you are running a test with a large number of users, you can use Agent computers to add load to the server.
- ▶ **Running schedules with many GUI users** – If you are running a schedule with more than one GUI user, you need an Agent computer, because only one GUI user can run on each computer.

What Is LoadTest?

- ▶ **Running scripts on more than one computer** – If you are running a functional test, you can save time by running the scripts on the next available Agent computer instead of having the Master computer run all the scripts. Of course, the scripts must be modular and independent.
- ▶ **Testing hardware configurations** – If you are testing different hardware configurations, you can run scripts on different Agent computers that are set up with these hardware configurations.

Schedules

Typically, multiple scripts and multiple computers are involved in a test. At runtime, script playback is coordinated by **schedules** that you design. These schedules add an emulated workload to the server. You run these schedules from the Master computer.

Once you have used LoadTest to create schedules that describe a baseline of behavior for the server, you can run these schedules repeatedly against successive builds of your product, and you can analyze the results using LoadTest's reporting tools.

Types of Users

In LoadTest, there are two types of emulated users—GUI users and virtual users.

GUI Users

A **GUI user** is a single instance of a GUI script running on a computer. Only one GUI user at a time can run on a computer.

A GUI user emulates the activities that actual users perform as they interact with the application in their daily work. Such activities include keystrokes and mouse movements.

In general, you include GUI users in functional tests. With LoadTest, however, you can also use GUI users in performance tests. For information about functional and performance tests, see *Types of Tests* on page 1-7.

Virtual Users

A **virtual user** is a single instance of a virtual user script running on a computer. Unlike GUI users, many virtual users can run on a computer simultaneously.

A virtual user emulates client/server requests sent to a server. When you record a virtual user script, Robot records a client's requests—such as Oracle, Microsoft SQL Server, and HTTP requests—to the server. Robot also records the server's responses. This network traffic is the only activity that Robot records. Robot ignores GUI actions such as keystrokes and mouse clicks.

You include virtual users in performance tests. For information about performance tests, see *Performance Tests* on page 1-7.

Because many virtual users can run on a computer simultaneously, virtual users let you add a workload to a client/server system. Virtual users also let you determine scalability and measure server response times.

GUI Users vs. Virtual Users

The following table summarizes the differences between GUI users and virtual users:

GUI User	Virtual User
Only one GUI user can run on a computer at one time.	Many virtual users can run on a computer at the same time.
Plays back GUI actions such as keystrokes and mouse clicks against GUI objects. These actions are recorded and stored in GUI scripts, written in the SQABasic language.	Plays back the requests that a client sends to a server. These requests are recorded and stored in virtual user scripts, written in the VU language.
Can be used in functional tests and performance tests.	Used in performance tests to add user load to the system—for example, to measure server response times under varying workloads.
Runs either in Robot or in a LoadTest schedule.	Runs in a LoadTest schedule only.

Types of Tests

You can use LoadTest to perform the following types of tests:

- ▶ Performance tests
- ▶ Distributed functional tests

NOTE: A third type of automated testing, structural testing of Web sites, is described in the *Rational Robot: Try It! With Rational SiteCheck* card.

Performance Tests

A **performance test** helps you determine whether a multi-client system is performing within user-defined standards under varying loads. Performance tests require the LoadTest software.

What Is LoadTest?

Performance tests involve loading the server with many virtual users. For example, you might have a timer in one virtual user to find out how much time a query takes when 1000 other virtual users are sending requests to the same server at the same time.

Additionally, one or more GUI users can be included in the test to measure a variety of performance issues that occur when many virtual users are running against the same server. Because GUI users measure client response times, this end-to-end response time is more indicative of what a real user experiences when there is significant client processing or screen-painting time associated with the user activity than you are measuring. Also, a GUI user is a good cross-check for your virtual user responses.

The term “performance testing” includes the following types of tests:

- ▶ Load tests
- ▶ Stress tests
- ▶ Contention tests
- ▶ Configuration tests

Load Tests

Load testing is designed to test client or server response times under varying load. Load tests also are used to help testers compute the maximum number of transactions a server can handle over a given time period. In addition, when a client/server system uses workload balancing or a distributed architecture, load testing can help ensure that the load balancing or distribution methods work as designed.

Load tests can involve virtual users only (for measuring server response times), or virtual users and GUI users (for measuring client response times).

Stress Tests

Stress testing is the process of running your client application under extreme conditions to see if they or the server “break.” Examples of stress testing include:

- ▶ Continuously running a client application for many hours.
- ▶ Performing a large number of transactions.
- ▶ Having hundreds of users perform the same operation or a specific combination of operations at virtually the same moment.

Other types of stress on the system include insufficient memory, unavailable services or hardware, or diminished shared resources on the server.

Stress testing helps you ensure that your client application or the server is able to handle production conditions, where the ineffective management of computer resources can result in system crashes.

You can test the following types of stress conditions:

Type of stress condition	Type of users needed
A heavy volume of users performing the same activity simultaneously	Virtual users
A few users continuously repeating the same action on the client application hundreds or thousands of times or over long periods of time	GUI users
A few users continuously repeating the same client requests to a server hundreds or thousands of times or over long periods of time	Virtual users
Combinations of the preceding three conditions	Virtual and GUI users

Contention Tests

Contention testing involves executing a combination of GUI and virtual users on one or more computers to simulate an actual user environment. For example, you might have GUI users and multiple virtual users accessing the same database to reveal problems in areas such as locking, deadlock conditions, and concurrency controls.

Contention testing is often difficult to perform because it requires precise coordination between users. In LoadTest's point-and-click environment, you can conduct multi-computer tests in which GUI and virtual users wait for conditions to be satisfied on their own computers, or on other computers, before they continue running. For example, you can have a GUI user on one computer add a record to a database, and have a GUI user on another computer pause before attempting to read the record until the first GUI user finishes its script.

If you want to run a contention test under heavy user conditions, virtual users can add the load.

Configuration Tests

In today's heterogeneous client/server environments, each user's computer can have a different mix of hardware and software, creating a risk that the application software will run on some computers and not others. In configuration testing, you want to ensure that your product will continue to run on multiple platforms.

LoadTest lets you easily schedule the same tests to run on different Agent computers, which in turn lets you:

- ▶ Test for compatibility issues.
- ▶ Determine the minimum and optimum configuration of hardware and software for running the application.
- ▶ Learn how each part of your application performs on each computer.

While you typically perform configuration testing with GUI users only, you can also load your client/server system with virtual users to test the effects of adding memory, more processors, and so on.

Distributed Functional Tests

Distributed functional tests allow multiple GUI objects to be tested simultaneously in a distributed, efficient QA lab environment.

Fully testing a Windows client/server application might require running hundreds of GUI scripts. The testing process will take a long time if each GUI script must be executed sequentially in Robot on a single computer. You can accelerate the process considerably by distributing the GUI scripts over multiple computers and running them with LoadTest.

Because LoadTest lets you execute a number of different scripts simultaneously on networked computers, and control the execution of those scripts from a single computer, LoadTest is extremely useful for performing distributed functional testing.

Assigning scripts to multiple computers through the next available **selector** is particularly useful in distributed functional testing. With this feature, you can assign scripts to the next available computer, creating a reliable and efficient QA lab operation. If a computer crashes or loses network connectivity during test execution, testing does not stop—LoadTest simply assigns scripts to one of the other available computers.

You perform distributed functional testing with GUI users only.

Testing Response Times

LoadTest lets you measure various indicators of performance. Whereas distributed functional testing measures correctness in terms of straightforward “pass/fail” responses, performance testing measures time: How long did it take for the action to complete? How quickly was the server able to respond under heavy load conditions? You can measure the client response time or the server response time, or both.

The following sections explain the difference between server and client response times.

Client Response Time

When you use timers in a GUI script, the client response times that you measure include GUI activities and GUI overhead, such as the time it takes to display a set of records retrieved in a database query.

Testing client response times under varying load conditions requires:

- ▶ Multiple virtual users loading the system by sending requests to a server.
- ▶ One or more GUI users acting as the client application. The GUI user runs against the same server as the virtual users that are loading the system. SQABasic timers in the GUI script measure client response times for individual activities (such as accessing a database or Web site) or a sequence of activities (such as accessing a Web site and sending a query to the server).

SQABasic timers measure the duration of GUI activities in a script. For example, you may want to measure the time required to perform a database transaction on a remote server, or how long it takes the application to perform tasks on client computers with different hardware configurations. For more information about timers, see the *Using Rational Robot manual*.

Server Response Time

Server testing generally focuses on finding where bottlenecks and overload can be problematic. You identify potential problem areas by testing server response times under varying user loads.

In a virtual user script, the server response times are automatically measured for each emulation command, typically consisting of a single or a few server requests. A single user transaction can consist of many emulation commands. To measure the time of this entire block of commands, you can insert timers around the activity of interest, either during recording or by editing the script.

Testing Web Servers

To test Web server response times, you load the system with virtual users. You can measure the response time for a single server request, for a block of requests, or for multiple requests.

You start a timer when a request is made, and then stop the timer after the server's response. This measures how long the server takes to process the request and return information, without the additional time required for GUI overhead within the client application.

Testing Database Servers

You can use timers in GUI and virtual user scripts to measure client response times in a client/server system under the load of other virtual users.

LoadTest can be used to simulate large numbers of networked TU XEDO workstation client users, measuring the responsiveness of both TU XEDO system and application server processes. Because the LoadTest TU XEDO emulation commands are modeled after the industry-standard XATMI and TU XEDO ATMI application programming interfaces, extremely accurate test cases can be produced to stress every aspect of a TU XEDO server's performance.

Since there is an emulation command for every load-generating TU XEDO primitive, it's easy to measure individual response times for service calls, transaction commits, and rollbacks, as well as many other message broadcasting, queuing, and event brokering operations. You can even measure initial client connection times to aid in tuning workstation listener and workstation handler resources. All TU XEDO client/server communication modes and standard typed buffer types are supported.

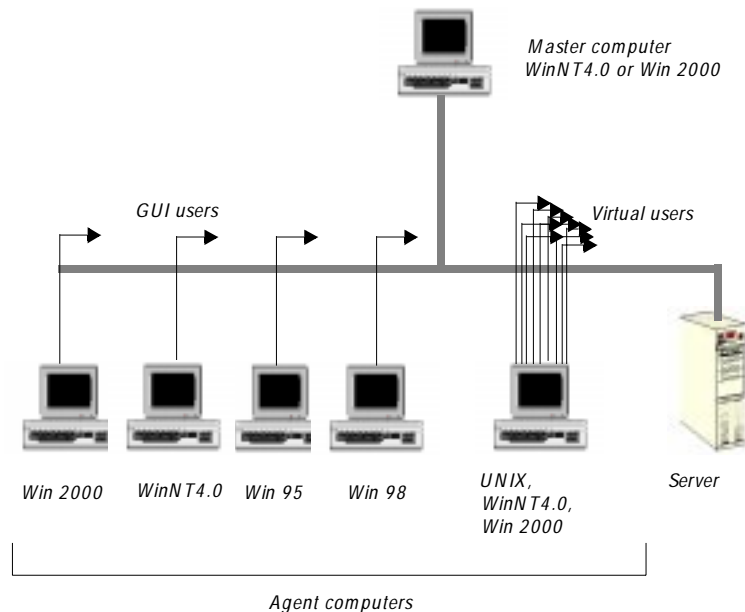
The LoadTest Environment

LoadTest enables you to run a schedule in a distributed environment consisting of a single Master computer that runs under Windows NT (on which you coordinate test execution and play back scripts), and zero or more Agent computers (on which you play back scripts).

An Agent computer can run on Windows 95 or 98, Windows NT, Windows 2000, or any of the supported UNIX Agent platforms. The following table shows the types of users you can run on the Agent platforms:

Agent platform	Type of user
Windows 2000	GUI users and virtual users
Windows NT 4.0	GUI users and virtual users
Windows 95 and 98	GUI users
Solaris 2.x	Virtual users
AIX 4.x	Virtual users
HP/UX 10.x and 11.x	Virtual users
Sequent Dynix	Virtual users

The following figure illustrates a typical LoadTest configuration:



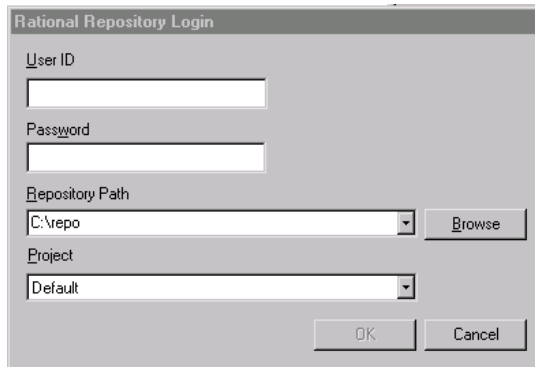
The server can run under a variety of operating systems, and can be connected to the Master and Agent computers over a TCP/IP network.

Logging into LoadTest

After you have planned your tests and recorded your scripts, you are ready to put them into a schedule. To do this, you must log into LoadTest.

To log into LoadTest:

1. Click **Start** → **Programs** → **Rational PerformanceStudio 7.1** → **Rational LoadTest**. The Rational Repository Login dialog box appears:



The image shows a dialog box titled "Rational Repository Login". It contains the following fields and controls:

- User ID**: A text input field.
- Password**: A text input field.
- Repository Path**: A dropdown menu showing "C:\repo" and a "Browse" button to its right.
- Project**: A dropdown menu showing "Default".
- OK** and **Cancel** buttons at the bottom right.

2. Type your user ID and password. If you do not know these, see your administrator.
3. Select a repository from the **Repository Path** box.
4. Select a project from the **Project** box.
5. Click **OK** to log in.

▶▶▶ C H A P T E R 2

Before You Begin

This chapter provides guidelines for planning performance testing against a database server and for functional testing with GUI users. It includes the following topics:

- ▶ About performance and functional tests
- ▶ Planning performance tests
- ▶ Planning functional tests

About Performance and Functional Tests

When you plan tests, you must first determine whether you are interested in performance testing, functional testing, or both types of testing.

In performance testing, you can measure the following things:

- ▶ The client response time. This is the time it takes for a GUI user to enter a request, the server to respond to the request, and the GUI user to see the results. If you are interested in the total end-to-end response time, as seen by the user, you run a performance test with GUI users.
- ▶ The server response time. This is the time it takes for the server to process a request. If you are interested in the server response only, you run a performance test with virtual users.

Functional testing involves GUI users. You are testing the accuracy of the application and how it behaves on different computers. You need only a few users to do this.

Functional testing tends to have well-defined objectives and outcomes. For example, if the application has a feature that saves a file to disk, it is relatively straightforward to test this feature. If a file gets saved correctly, it passes the test. If it does not get saved correctly, it fails the test.

Performance testing can be more complex than functional testing because performance itself is subjective. What one user might perceive as too slow, another user might perceive as perfectly acceptable. Therefore, when planning performance tests, you need to put some thought into what constitutes acceptable performance.

Another complication of performance testing is that performance varies widely depending on workload conditions. Querying a database on a system that is primarily used for CPU-intensive activities yields a different response time than performing the same query on a system used primarily for generating I/O-intensive database reports.

The following table summarizes the main differences between performance and functional tests:

Functional Tests	Performance Tests
Answer the question: “Does the system do what it is designed to do?”	Answer the question: “How quickly does the system perform what it is designed to do?”
Focus on how the system behaves against the functional or the design specification. The system must work as specified.	Focus on how the system behaves when executing actual business operations.
Do not use a workload model.	Model an actual workload, which is an approximation of the real-world environment you are trying to emulate.
Might deliberately use incorrect data to test error recovery and error handling. For example, if a field accepts a number from 1 to 100, the test might use the numbers 100, 1, 0, 101, and -1.	Use data that mirrors the actual work done. For stress tests, the data might not mirror the actual work done but instead will stress the capacity of the system.

Planning Performance Tests

Testing the performance of a server typically involves loading the server with many virtual users. The objective is to find out how the server performs under the load.

Some of the performance questions you might want to answer are:

- ▶ How many virtual users can the server support under normal conditions?
- ▶ Are there any situations where server performance degrades suddenly under normal conditions?

- ▶ How does the system perform when you exceed the normal conditions? In a worst-case scenario, does the system degrade gracefully or does it break down completely?
- ▶ How does the system perform under varying hardware configurations?

The following sections discuss the key steps that are involved in planning a test.

Setting Pass and Fail Criteria for Performance Tests

Because performance can be subjective, it is essential that you not only identify the features to be tested but that you also determine the criteria that will determine whether performance passes or fails. The pass or fail criteria will often involve a range of acceptable response times. For example, you could define the following as an acceptable response time:

- ▶ At 100 users, 90% of all transactions have an average response time of 5 seconds or less. No response time can exceed 20 seconds.
- ▶ At 500 users, 80% of all transactions have an average response time of 10 seconds or less. No response time can exceed 45 seconds.

Identifying Performance Testing Requirements

When planning a performance test, you need to determine the hardware and software that your test requires. For example:

- ▶ Server computers: database servers, Web servers, other server systems
- ▶ Client computers: Windows NT, 95, or 3.1 computers; network computers; or Macintosh or UNIX workstations
- ▶ Databases that will be accessed
- ▶ Applications that will be running

In addition, you need to determine the following parameters for your tests:

- ▶ How large should the test databases and other test files be in order to accurately represent the real workload?
- ▶ How should the data be distributed across the server in order to prevent I/O bottlenecks?
- ▶ If you are testing a database, how should the key database parameters be set?

Designing a Realistic Workload

If you are testing performance, it is essential that your workload model mirror the workload at your site. Therefore, you must determine the types of transactions that occur at your site. For example, do your users generally query the database and update it occasionally, or do they update it frequently? If they update the database frequently, are the updates complex and lengthy, or are they short?

Here are some additional factors to consider when designing the workload:

- ▶ **The workload interval** – The period of time the workload model is supposed to represent. For example, the workload interval could be a peak hour, an average day, or an end-of-the-month billing cycle.
- ▶ **Test variables** – The factors you will change during the performance test. For example, you might vary the number of users to understand how response time degrades as the workload increases.

It is best to change only one variable at a time. If performance changes, you know that the change was caused by that variable.

You set these test variables when you set up a schedule. For more information, see *Designing Schedules* on page 7-1.

- ▶ **Functional user classifications** – Categorize the users into groups based on the types of activities they perform. For each user group, you need to identify the number of users or the percentage of overall users. For example, you might group 20% of the users into Accounting, 30% of the users into Data Entry, and 50% of the users into Sales.

You set up user groups in a schedule. However, you should plan your scripts beforehand so that when you record them they reflect the actions of a user group. For more information about setting up user groups, see *Inserting User Groups into a Schedule* on page 7-5.

- ▶ **User work profiles** – The set of activities that the users perform and the frequency with which they perform them. The user actions should mirror as closely as possible the mix of tasks that the users actually perform. For example, if the Sales user group access the database 70% more than the other two groups, be sure that the workload reflects this.
- ▶ **User characteristics** – Determine how long a user pauses before executing a transaction, the typing rates, and the number of times a transaction is executed consecutively. It is important to model the user characteristics accurately because the values directly affect the overall performance of the system. For example, a user who thinks for 5 seconds and types 30 words per minute puts a much smaller workload on the system than a user who thinks for 1 second and types 60 words per minute.

You use delays and think times to model the user characteristics. For more information about delays, see *Inserting a Delay* on page 7-19. For more information about think times, see *Think maximum (ms)* on page 3-16.

When designing a workload, make sure to consider factors such as these to ensure an accurate test environment.

Designing Performance Tests

Once you have decided on the pass and fail criteria, hardware and software requirements, and workload model, you are ready to record the user actions and set up the tests. Some questions you might consider are:

- ▶ **The termination conditions** – If one user fails, should the test stop or should it keep running? Generally, if you are implementing a large number of virtual user tests and a few virtual users fail, the test can remain running. However, if a virtual user that performs a fundamental task (such as setting up the database) fails, the test should stop.

You set termination conditions in the schedule. For more information about setting termination conditions, see *Controlling How a Schedule Terminates* on page 7-64.

- ▶ **The stable workload** – Should the test wait until all the users are connected, or should the test begin running immediately? Generally, if you are trying to measure the response time for users, you should wait until they are all running before the actual testing begins.

You define a stable workload for reporting purposes in the Performance report. For more information, see *Reporting on a Stable Load* on page 9-25.

- ▶ **The applications that you will test** – It will not be cost-effective to test all of your applications. You do not want to spend your time and resources trying to include an application that has little effect on overall performance—for example, you might not want to include an application that updates a database at the end of the year. In general, you want to identify the 20% of the applications that generate 80% of the workload on your system. You also need to limit the number of applications to reduce test complexity.

- ▶ **The level of information to be logged** – Typically, you set the `Record_level` environment variable to `ESSENTIAL` so you can calculate response times.

Leave the `Log_level` environment variable at its default setting of `ALL` during the early stages of script creation and testing. This is useful in debugging scripts. When you are satisfied that your VU scripts are performing as desired, set the log level to `ERROR`.

Set the log level to `ERROR` when doing medium to large multi-user runs. This saves disk space. Logging all commands instead of only those that fail can quickly consume hundreds of megabytes of disk space. It also consumes CPU overhead, so it will reduce the number of users you can run on each computer.

You set the level of information logged when you set up a schedule. For more information about setting these levels, see *Editing the User Settings* on page 7-38.

You also need to set up a test database, since you will probably not run your tests on the production database.

Examples of Performance Tests

The following sections outline the objectives and approaches for some typical performance tests. Each test objective is accompanied by a table that outlines the key elements you might include when defining the test. The tables are intended only as a guide. They do not attempt to define all the possible elements you can include in your performance tests.

Number of Virtual Users Supported Under Normal Conditions

You want to determine the number of users that a server can support before you release the system. You want to be sure that the system can meet your scalability requirements. How many users can the system support before the response is unacceptable?

Assume that your database system is estimated to support approximately 500 users. Therefore, you initially plan to run your schedule with 300, 400, 500, 600, and 700 users concurrently performing multiple tasks. The following table shows the key elements you might include when designing the test:

Scripts	Schedule	Reports
<p>A script to initialize the database.</p> <p>A script to log users in.</p> <p>A script for each user task:</p> <ul style="list-style-type: none"> ▶ adding records ▶ deleting records ▶ querying the database ▶ running payroll reports 	<p>A fixed user group with one user. This user logs in, initializes the database, and sets an event indicating that the database is initialized.</p> <p>A scalable user group with many users. This group logs in and waits until the event is set. It then executes the scenario.</p> <p>A scenario that contains:</p> <ul style="list-style-type: none"> ▶ a selector to randomly select a script ▶ a script for each user task 	<p>LogViewer report to show whether all users in the schedule successfully ran to completion.</p> <p>Status report to show whether the server completed its requests successfully.</p> <p>Performance reports for each schedule run: 300, 400, 500, 600, and 700 users.</p> <p>Compare report comparing the output of all five Performance reports.</p>

Incrementally Increasing Virtual Users

A common requirement in performance testing is to model what happens across a span of time as different users perform their work. For example, you might want to test how your server performs early in the morning when people are arriving for work and starting their day. You might also want to know how the server handles an increasing workload during the day and particularly those times of peak load.

With LoadTest, it is easy to model this type of workload by incrementally loading virtual users. To incrementally load virtual users, you first develop a model of the workload that you want to test. For example, you write down the frequency and volume of use of your applications. Then, when setting up your schedule, you do the following:

- ▶ Define scheduling methods to start different virtual user groups at different times during the life of the schedule.
- ▶ For each virtual user group, set the number of virtual users that will run the script and, optionally, an iteration count, that are appropriate for your test.

By layering the start time and iteration count of your virtual users, you can build up load incrementally, or you can add spikes of load at specific times in your schedule run.

Before You Begin

The following list describes a sample test that represents overlapping shifts:

- ▶ Suppose you start a schedule with 100 virtual users. Because this group of virtual users represents the early shift of entry clerks repeating the same group of order entry transactions over and over, you set each virtual user to run many iterations of the transaction—enough iterations to keep this group of virtual users running the script until the schedule ends. You may have to experiment to determine how many iterations are needed.
- ▶ Then, through a schedule, you set a **Delay**. The **Delay type** might be from the start of the schedule, or it might begin at a certain time of day. When the delay is over, 200 new virtual users begin. This is the next shift of entry clerks, which overlaps the first shift.
- ▶ During the combined shift, which represents peak load, 300 virtual users will be performing their transactions repeatedly.

The following table summarizes a sample test that represents overlapping shifts:

Scripts	Schedule	Reports
<p>A script to initialize the database.</p> <p>A script to log users in.</p> <p>A script for each user task:</p> <ul style="list-style-type: none">▶ adding records▶ deleting records▶ querying the database▶ running payroll reports	<p>A fixed user group with one user. This user logs in, initializes the database, and sets an event indicating that the database is initialized.</p> <p>A fixed user group with 100 users. Each user logs in and waits until the event is set. Each user then executes many iterations of the scenario.</p> <p>A fixed user group with 200 users that delays for 2 hours. each user then logs in, checks that the event is set, and executes many iterations of the scenario.</p> <p>One scenario that contains:</p> <ul style="list-style-type: none">▶ a selector to randomly select a script▶ a script for each user task	<p>LogViewer report to show whether all users in the schedule successfully ran to completion.</p> <p>Status report to show whether the server completed its requests successfully.</p> <p>Two Performance reports:</p> <ul style="list-style-type: none">▶ One report on the time period from the start of the run until 2 hours have passed▶ One report on the time period from 2 hours until the end of the run <p>Compare report comparing the output of both Performance reports.</p>

How a System Performs Under Stress Conditions

Stress testing can be referred to as a relentless attempt to cause a breakdown in the server. You might suspect that there are some weak areas of the server that may break down or diminish due to some operation being performed a very high number of times or over a long period of time.

Since stress tests perform multiple simultaneous operations (such as sending hundreds of queries to the server at the same moment), virtual users provide the most practical and effective means of performing this type of stress test. Running virtual user scripts continuously helps you understand the long-term effects of running the application under stressful conditions.

In a simple stress test, you might want to conduct a test where virtual users perform the same operation continuously and repeatedly for hours on end. Your test might involve:

- ▶ Inserting thousands of records into a database
- ▶ Sending thousands of query requests to a database

The following table summarizes a sample stress test:

Scripts	Schedule	Reports
<p>A script to initialize the database.</p> <p>A script to log users in.</p> <p>A script for each user task:</p> <ul style="list-style-type: none"> ▶ adding records ▶ deleting records ▶ querying the database ▶ running payroll reports 	<p>A fixed user group with one user. This user logs in, initializes the database, and sets an event indicating that the database is initialized.</p> <p>A scalable user group with 1000 users. Each user logs in and waits at a sync point. When all the users are synchronized, each user executes many iterations of the scenario.</p> <p>One scenario that contains:</p> <ul style="list-style-type: none"> ▶ a selector to randomly select a script ▶ a script for each user task 	<p>LogViewer report to show whether all users in the schedule successfully ran to completion.</p> <p>Status report to show if the server behaved correctly, even under stress.</p> <p>Performance reports for each schedule run: 900, 1000, and 1100 users. These Performance reports show when the system starts to degrade, and ensure that the degradation is graceful.</p> <p>Compare report comparing the output of each Performance report.</p>

How Different System Configurations Affect Performance

LoadTest lends itself well to configuration testing because of the way a schedule is organized and run. You might conduct a configuration test for a variety of reasons—for example:

- ▶ You want to test how your system performs with more (or less) memory.
- ▶ You want to test how your system performs with a different amount of disk space.
- ▶ You might have several types of network cards that you want to experiment with to see which provides the best performance.

Since you are testing the same scripts under different configurations, you would run a schedule, change the computer configuration, and run another schedule with the same number of users on the same computer. If you physically change servers, you must alter the “connect” line of the VU script.

The following table summarizes a sample configuration test for 100 users:

Scripts	Schedule	Reports
<p>A script to initialize the database.</p> <p>A script to log users in.</p> <p>A script for each user task:</p> <ul style="list-style-type: none"> ▶ adding records ▶ deleting records ▶ querying the database ▶ running payroll reports 	<p>A fixed user group with one user. This user logs in, initializes the database, and sets an event indicating that the database is initialized.</p> <p>A fixed user group with 100 users. Each user logs in and waits until the event is set. Each user then executes many iterations of the scenario.</p> <p>One scenario that contains:</p> <ul style="list-style-type: none"> ▶ a selector to randomly select a script ▶ a script for each user task 	<p>LogViewer report to show whether all users in the schedule successfully ran to completion.</p> <p>Status report to show if the server returned expected responses, even under stress.</p> <p>Performance reports for each schedule run on each configuration.</p> <p>Compare report comparing the output of each Performance report.</p>

Planning Virtual User Scripts

During the test planning phase, you can use Rational TestManager to define script properties before you actually record the script in Robot.

Script properties include:

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

To plan scripts and define properties in TestManager, click **File** → **Plan Script**. For more information about planning scripts through TestManager, see the *Using Rational Robot* manual.

As you plan what you want to record, keep in mind that when you play back the script it should emulate real users performing everyday tasks—for example:

- ▶ Typically, a script contains a single transaction (such as adding a new order to a database) or multiple transactions that are executed in succession (such as adding an order to a database and then generating an invoice based on the order).

In LoadTest, you insert the script into a user group that is likely to execute the transactions in the script—for example, an Order Entry user group for transactions that add new orders to the database.

- ▶ Typically, a virtual user script represents requests to a server by *multiple* clients. In other words, when you run the script on a single computer, hundreds of virtual users might run the script. You specify the number of virtual users to run the script when you design the schedule.
- ▶ Often, a script represents a transaction that is repeated multiple times. For example, a member of an Order Entry department would not enter a single order and then quit for the day. Consequently, you might want each virtual user to run a script multiple times. You set these iterations when you design the schedule.

Setting Virtual User Recording Options

Before you record a virtual user script, make sure your virtual user recording options are set the way you want them for the recording session. For example, you use recording options to set the recording method to use.

You can review and set virtual user recording options in any of these ways:

- ▶ Before you begin recording, click **Tools** → **Virtual User Record Options**.
- ▶ When you initiate recording, click **Options** in the Record Virtual User dialog box.
- ▶ When you end recording, click **Options** in the Stop Recording dialog box. At this time, you can only set options that affect script generation (in the tabs **Generator**, **Generator Filtering**, and **Generator per Protocol**).

For information about recording options, see *Setting Recording Options* on page 3-1.

Virtual User Recording Considerations

When you record a virtual user script that accesses a database, you often need to make sure that when you run the schedule, the underlying database is in the same state as it was when you originally recorded the scripts. There are several ways to accomplish this:

- ▶ At the start of a schedule run, have one user in the run initialize (roll back) the database before the other users do active work. The examples in the previous section use this method.
- ▶ Before each schedule run, you can manually roll back the database to the state it was at the beginning of the recording session.
- ▶ Have the last script in a session perform the necessary operations to restore the database, such as removing inserted records or undoing updates.

As you record the script, it is a good idea to:

- ▶ Split the recording session into multiple scripts. Typically, each script recorded during a session contains a logical set of actions performed by one user. For example, if you record a query transaction against a Web server and an order entry transaction against a database server, you would probably split the session into two different scripts—one for each transaction.

Splitting a session into multiple scripts is also a convenient way to treat sections of the recording session differently when you add the session's scripts to a schedule. For example, suppose you log into a database server and perform three different transactions. If you split the session so that the login and each transaction is a separate script, you can set up the schedule to perform many iterations of the transactions but perform the login script just once.

For more information about sessions, including how to split a virtual user recording session into multiple scripts, see *Splitting a Session into Multiple Scripts* on page 4-15.

- ▶ Add comments to the script. This is because one user action, such as clicking in a window to add a record to a database, can translate into more than one emulation command. Therefore, adding comments to your script while recording makes the script easier to understand and easier to map to the user actions. For more information about adding comments, see *Comments* on page 5-11.
- ▶ Insert blocks into the script. Each virtual user script automatically measures the server response time between one emulation command and the next. However, you may want to measure larger units of work, such as an entire transaction. To do so, add blocks during recording to bracket these units of interest. For more information about adding blocks, see *Blocks* on page 5-4.

After you record the script, play back the script in a schedule to make sure it runs correctly. You will not receive any syntax errors because you have not yet modified the script. However, you may receive playback errors if your client-side environment is set up differently from the way it was set up during recording or if you did not restore the database to its initial recording state.

Modifying Virtual User Scripts

Once you have recorded a series of virtual user scripts, you should modify them so they will run more than once. For example, if your recorded script deletes a record with the key of John Doe, you cannot run that script multiple times. If you run the script with a workload of 100 users, the first virtual user will succeed, but the next 99 virtual users will get an error.

To avoid this problem, you can use a datapool to supply the data values to your script. Typically, you use a datapool so each virtual user that runs the script can send unique data to the server. Also, a single virtual user that performs the same transaction multiple times can send unique data to the server in each transaction. If you do not use a datapool, each virtual user sends the same values to the server (which are the values that are in the recorded script).

In general, you create a datapool immediately after you record the server or user actions. You create a datapool with Robot or TestManager.

You may also want to make the following changes to a recorded script:

- ▶ **Modifying think time distribution** – The default think time distribution is `CONSTANT`. If 1000 users run a script, they will delay the exact same think time with each command. By setting a think time distribution of `NEGEXP`, each user will select a think time at random from a negative exponential distribution. To change the think time distribution, you must modify the value of `Think_dist` in each script.
- ▶ **Suppressing aberrant think times** – You might have been interrupted by a ten-minute phone call during the recording process. When you play back the script, it pauses for ten minutes. You can remove this think time by editing the script or by setting a maximum think time in Robot's Virtual User Record Options. For example, if you set a maximum think time of one minute, when the script runs, it pauses for one minute, not ten.
- ▶ **Modifying start and stop times** – You might want to modify the placement of `start_time` and `stop_time` commands, if you inserted timers or blocks during recording.
- ▶ **Inserting synchronization points** – Inserting a synchronization point enables you to coordinate the activities of a number of virtual users by pausing and resuming activities. A useful technique is to synchronize all users at the beginning of the schedule and to stagger release times so that users executing the same script do not all start the script at the same time.

You can insert a synchronization point into a schedule or into a script:

- **Into a schedule.** The advantages of inserting a synchronization point into a schedule are that the synchronization point is visible in the schedule and that the synchronization point is easy to change without editing the script. For more information, see *Inserting a Synchronization Point* on page 7-25.
- **Into a script.** You can insert a synchronization point into a script in either of these ways:

During recording. For more information, see *Synchronization Points* on page 5-7.

During script editing, by manually typing the `VU sync_point` statement into the script. For more information, see the *VU Language Reference*.

Correcting Errors in Virtual User Scripts

When you run a schedule, you might encounter the following errors in a script:

- ▶ Syntax errors. These will occur only if you have modified the script. LoadTest will not run a schedule unless it can compile all of the scripts successfully. Therefore, you must fix the syntax errors first.

Some of the most common syntax errors involve:

- attempting to use a function without the proper number and type of arguments
- attempting to use default scope variables across scripts
- ▶ Runtime errors. These errors might appear while you monitor a schedule, and are often related to the environment or state of the client or server software. For example, you might need to fix:
 - initialization problems
 - abnormal user termination
 - abnormal loss of connections
 - access to variables no longer in context
 - a variable that you are attempting to use but that has not been initialized

The LogViewer's User Log and User Error files contain the information necessary to fix these runtime errors.

- ▶ Connection failures and initialization timeouts.

Analyzing Performance Results

LoadTest generates a great deal of data about your tests, and at first, the sheer volume of data might be overwhelming. However, if you planned your tests carefully, you should be reasonably certain what data is important to you.

It is a good idea to first check that your sample is statistically valid. To do this, run a Performance report and a Response report on your data.

The Performance report includes two columns: Mean and Standard Deviation. If the mean is less than three times the standard deviation, your data might be too dispersed to get meaningful results.

The Response graph shows the response time versus the elapsed time of the run. The data should reach a steady-state behavior rather than getting progressively better or worse. If the response time trend gets progressively better, you might be including login time in your results rather than measuring the results when they have attained a stable load. Or the data accessed in your database may be smaller than realistic, resulting in all accesses being satisfied in cache.

After you are satisfied that your sample is valid, you can start analyzing the results of your tests. When you are analyzing results, it is a good idea to use a multilevel approach. For example, if you were driving from one city to another, you would use a map of the United States to plan an overall route, and a more detailed city map to get to your destination. Similarly, when you analyze your results, you should first start at a macro level and then move to levels of greater detail.

The following sections outline the different levels of detail that you can use to analyze the results of your tests.

Comparing Results of Multiple Runs

The first level of analysis involves evaluating the graphical summaries of results for individual schedule runs and then comparing the results across multiple runs. For example, you can examine the distribution of response times for individual users or transactions during a single schedule run. You can also compare the mean response times across multiple runs with different numbers of users.

This first-level analysis lets you know whether your performance goals are generally met. It helps identify trends in the data, and can highlight where performance problems occur—for example, performance might degrade significantly at 250 users.

For this type of analysis, you run the Performance and Compare reports.

Comparing Specific Requests and Responses

The second level of analysis involves examining summary statistical and actual data values for specific user requests and system responses. Summary statistics include standard deviations and percentile distributions for response times, which indicate how the system responses vary as seen by individual users. If you are testing a SQL database, you might want to trace specific SQL requests and corresponding responses to analyze what is happening and the potential causes of performance degradation.

For second-level analysis, you could do the following:

1. Identify a stable measurement interval by running the Response report and obtaining two timestamps. The first timestamp occurs when the virtual users exit from the startup tasks. This is the timestamp of the last user who starts to do “real” work: adding records, deleting records, and so on. The second timestamp is the first user who logs off the system. You have now identified a “stable” measurement interval.
2. Create a Performance report using only the interval specified by these two timestamps.
3. Graph the Performance report to verify that the distribution has flattened.
4. Run the Performance, Compare, and Usage reports to examine the summary statistics for this measurement interval.

Determining the Cause of Performance Problems

The third level of analysis helps you understand the causes and significance of performance problems.

Analyzing Results Statistically

This detailed analysis takes the low-level data and uses statistical testing to help draw correct conclusions. Although this analysis provides objective and quantitative criteria, it is more time consuming than first- and second-level analysis and requires a basic understanding of statistics.

When you analyze your data at this level, you use the concept of **statistical significance** to help you understand whether differences in response time are real or are due to some random event associated with the test data collection. On a fundamental level, randomness is associated with any event. Statistical testing determines whether there is a systematic difference that cannot be explained by random events. If the difference was not caused by random error, the difference is statistically significant.

To perform a third-level analysis, you run the Performance and Response reports.

Some of the measurements you need to be concerned about when you do third-level analysis are:

- ▶ **Minimum** – The lowest response time.
- ▶ **Maximum** – The highest response time.
- ▶ **Mean** – The average response time. This average is computed by adding all of the response time values together and then dividing that total by the number of response time values.
- ▶ **Median** – The midpoint of the data. Half of the response time values are less than this point and half of them are greater than this point.
- ▶ **Standard Deviation** – How tightly the data is grouped around the mean.
- ▶ **Percentiles** – The percentages of response times above or below a certain point. The 90th percentile is often measured.
- ▶ **Outlier** – A value that is much higher or lower than the others in the data.

For example, suppose that System A and System B both have a mean response time of 12 milliseconds. This does not mean that the system response is the same. System A might have response times of 11, 12, 13, and 12. System B might have response times of 1, 20, 25, and 2. Although the mean time is the same, the minimum, maximum, and standard deviation are all quite different.

To do an even deeper analysis, you can save the report output in .csv format, and export it into a commercial statistics package.

Monitoring Computer Resources and Tuning Your System

Performance problems can be caused by limited hardware resources on your server rather than software design. For example, your disk job service times could be unacceptably slow due to a concentration of disk transfers all being sent to a single disk rather than being spread across several disk drives. This problem is typically fixed by relocating some of the frequently accessed files (such as swap files or temporary files) to a disk with less activity.

Performance problems can also be caused by overloaded LAN segments or routers, which result in substantial network congestion. Even the simplest round-trip delay from client to server and back can take several seconds to process. This problem is typically fixed by splitting an overloaded LAN segment into two or three segments with routers in between. Sometimes you may also add a second network card to server systems so they can be directly accessible to two LAN segments without going through a router.

Either of these hardware limitations can result in slow response time measurements that cannot be fixed by changing the software design.

LoadTest lets you match CPU, memory, and disk utilization metrics with user response time data. You can monitor your computer resource usage during a schedule playback and then graph this usage data over the corresponding user response times to determine whether imbalance in the hardware resources is causing slow response times.

For this type of analysis, you would run the Response report, and right-click on the report output to overlay the computer resource metrics one at a time to see if any of them are responsible for slow response times.

For more information about running the Response report, see *Mapping Computer Resource Usage onto Response Time* on page 9-27.

Planning Functional Tests

In the simplest sense, a functional test determines whether the software functions as designed. Typically, the ramifications of what has changed can be classified in one of two ways:

- ▶ An unplanned change – for example, a software defect.
- ▶ A planned change – for example, a fix for a software defect or an enhancement.

In either case, if you want to continue with your functional testing, you must make the newly accepted functionality the new baseline standard, and then compare this new standard against tests of subsequent builds of the application.

Distributed Functional Testing

Functional tests that use Robot alone are usually performed on a single computer. With LoadTest, however, you can perform functional tests in distributed mode, meaning that you can have many computers running concurrently. This enables you to:

- ▶ Expand your functional testing efforts to include additional computers that are configured differently – for example different operating systems, screen resolutions, clock speeds, and so on.
- ▶ Speed up the process of stand-alone functional testing by distributing the scripts among different computers and playing them back in the same schedule.

Setting Pass and Fail Criteria for Functional Tests

One of your primary tasks in planning a functional test is to identify the features to be tested and how you will determine whether the features pass or fail. If you are running a functional test, the pass or fail criteria will be in the form of functional defects. For example, you need to determine whether the latest release of the application produces:

- ▶ The same output as the baseline
- ▶ The same error detection as the baseline
- ▶ The same error recovery as the baseline

Identifying Functional Testing Requirements

When planning a functional test, you need to determine the hardware and software that your test requires. For example:

- ▶ Server computers – the server computers that will be accessed
- ▶ Client computers – the characteristics, such as processor speed, memory, and disk space, of the client computers, that run the application
- ▶ Databases that will be accessed
- ▶ Applications to be tested

Scheduling Functional Tests

If you are testing performance, you schedule your tests so that the workload model mirrors the workload at your site. If you are testing functionality, however, you schedule your tests so they can run on different computers, or on a specific computer. The following sections show how you might do this.

Distributing Tests Among Different Computers

With functional testing, you might want to run your tests immediately on any computer that is available. In this case, do two things:

- ▶ When you insert user groups into a schedule, click the Multiple Computers button and add your computers to the computer list that appears. For more information about scheduling scripts to run on different computers, see *Inserting User Groups into a Schedule* on page 7-5.

- ▶ After you have inserted your user groups, insert a **Next Available** selector. The scripts that you insert under the selector will be continuously sent out to the next available computer. Of course, the scripts must be designed so that they are self-contained and do not rely on one another. For more information about the Next Available selector, see *Inserting a Selector* on page 7-18.

Running Tests on a Specific Computer

If, however, you are testing functionality on a group of computers that have different hardware or software, you need to schedule the user groups to run on a particular computer. For more information about scheduling scripts to run on a particular computer, see *Inserting User Groups into a Schedule* on page 7-5.

Designing Functional Tests

Once you have decided on the number of users and the mix of tasks, you are ready to record the user actions and set up the tests. Some questions you might consider are:

- ▶ If one user fails, should the test stop or should it keep running? Generally, if you are testing functionality, the test should stop if a user fails.
- ▶ Should the test wait until all the users are connected, or should the test begin running immediately? Generally, if you are testing several configurations of a system, and the same script is running independently on each configuration, the test can begin running immediately.
- ▶ What type of data will be needed to test each function? Datapools are extremely useful here. They enable you to exhaust every combination and iteration of a field. You can populate a datapool with valid and invalid data. Although you will probably create the datapool later, you need to plan the type of data it will contain at the same time you plan the tests.

You also need to set up a test database, since you will most likely not want to run your tests on the production database.

To enable a test to be reused, you must record modular scripts so that they can start and stop at common, convenient points for even processing. For example, a set of modular scripts for a banking application might include querying an account, updating one of the attributes of the account, and returning to the menu where this function was first invoked. Most functional tests are based on many modular recordings that are placed into script shells in Robot.

Overview of a Distributed Functional Test

Assume that you want to test your Accounting software. You want to distribute your tests so that they can run as quickly as possible.

The following table summarizes how you set up this test:

Scripts	Schedule	Reports
A script to log users in. A modular script for each user task. A script to perform any cleanup work and then shut down the application.	A fixed user group, with one user assigned to each computer in the test, that logs the users in. A fixed user group, with one user assigned to each computer in the test, that contains a Next Available selector and modular scripts that run on any computer. A fixed user group, with one user assigned to each computer in the test, that shuts down the application.	LogViewer report to show whether all users in the schedule successfully ran to completion.

The previous table shows one way to perform a distributed functional test. There are many other ways to use LoadTest to build and run effective distributed functional tests. The most important thing to keep in mind is that all of the scripts should be modular in nature.

GUI Recording Considerations

Before you record a GUI script that accesses a database, you often need to make sure that when you run the schedule, the underlying database is in the same state as it was when you originally recorded the scripts. There are several ways to accomplish this:

- ▶ At the start of a schedule run, have one user in the run initialize (roll back) the database before the other users do active work. The examples in the previous section use this method.
- ▶ Before each schedule run, you can manually roll back the database to the state it was at the beginning of the recording session.
- ▶ Have the last script in a session perform the necessary operations to restore the database, such as removing inserted records or undoing updates.

As you record the user actions, it is a good idea to annotate the script. Annotating your script while recording makes the script easier to understand and to map to the user actions.

After you record the user actions or server traffic, play back the script in Robot. You want to make sure that it runs correctly before you modify it. You will not receive any syntax errors because you have not modified the script.

Modifying GUI Scripts

Once you have recorded a series of scripts, you should modify them so that they will run more than once. For example, if your recorded script deletes a record with the key of John Doe, you cannot run that script multiple times. If you run the script with a workload of 100 users, the first GUI user will succeed, but the next 99 GUI users will get an error.

To avoid this problem, you can use a datapool to supply the data values to your script. Typically, you use a datapool so that each GUI user that runs the script can send realistic data to the server. Also, a single GUI user that performs the same transaction multiple times can send realistic data to the server in each transaction. If you do not use a datapool, each GUI user sends the same values to the server (the values that are in the recorded script).

Another use of datapools is for testing the values of each field. So, for example, if you are testing a numeric field, you can populate your datapool with the minimum and maximum values accepted, one less than the minimum and one more than the maximum values, a null value, and so on.

In general, you create a datapool immediately after you record the server or user actions. You create a datapool with Robot or TestManager.

You will probably want to add a loop to your script to repeatedly test the values of a field.

Correcting Errors in GUI Scripts

When you run the schedule, you might encounter the following errors in a script:

- ▶ Syntax errors. For example, you might need to fix errors like the following:
 - An object did not exist during recording, but it occurs in playback.
 - An object's coordinates have changed between the time you recorded it and the time you run the schedule. For example, the coordinates of a drop-down combo box may have moved out of range.

To fix these problems, comment out the incorrect portion of the script, re-record that portion of the script, and insert the corrected code into the script.

- ▶ Runtime errors. These errors might appear while you monitor a schedule, and are often related to your environment. For example, you might need to fix:
 - Script command failures.
 - Unexpected active windows. For example, assume you recorded a script that creates a new file in Notebook, and then closes that file. When you play back that recording, the file already exists. When the script tries to close the file, it gets a message asking if it wants to overwrite the previous file. This message is an unexpected active window, because it did not appear during recording.

Analyzing Functional Results

The LogViewer shows whether an individual script passed or failed. By default, LoadTest displays the LogViewer if a script fails during a schedule run. If anything in the script failed, the LogViewer displays the script name in red. You can click on the script name to examine the cause of the failure. Scripts that pass are green, and scripts that you terminate are yellow.

For more information about the LogViewer, see the *Using Rational Robot* manual.

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 virtual user 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 Virtual User Recording

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

1. Set the virtual user recording options.

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

- The type of virtual user 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.

Setting Recording Options

- 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.
If you use 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.
For information about splitting a session, see *Splitting a Session into Multiple Scripts* on page 4-15.
 5. Optionally, insert timers, blocks, comments, and other features into the script during recording.
For information about inserting these features into virtual user scripts, see Chapter 5, *Adding Features to Virtual User Scripts*.
 6. Close the client application.
 7. Stop recording
 8. Robot automatically generates scripts.

Setting the Recording Method

A **recording method** is the type of virtual user recording that you want to perform. Your choices are:

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

In general, we recommend that you select network 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 accesses an Oracle 8 database.	Required		
The client application accesses an Oracle 7 database. (For network and proxy recording, you need to supply the name of the Oracle database. For more information, see <i>Providing the Name of an Oracle Database</i> on page 3-23.)	Recommended	First alternate	Second alternate
The client application is not installed on the Master.		Recommended	Alternate
The client application is not running on Windows NT 4, SP 3 or 4.		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, <i>specific</i> client and server computers.			Recommended
Neither the client nor the server computer is on the same network segment as the Master.			Required
An Ethernet switch controls network traffic, and neither the client nor the server application are installed on the same computer as the Master.			Required
The client application accesses a TUXEDO server.	Alternate	Recommended	

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** → **Virtual User 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** → **Virtual User 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 3-16.)

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 3-27.

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 Master 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 **Computer Admin** to define the computer network name and port number of interest.

Selecting a Network Card

If you are using network recording and the Master 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 **PerformanceStudio 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).

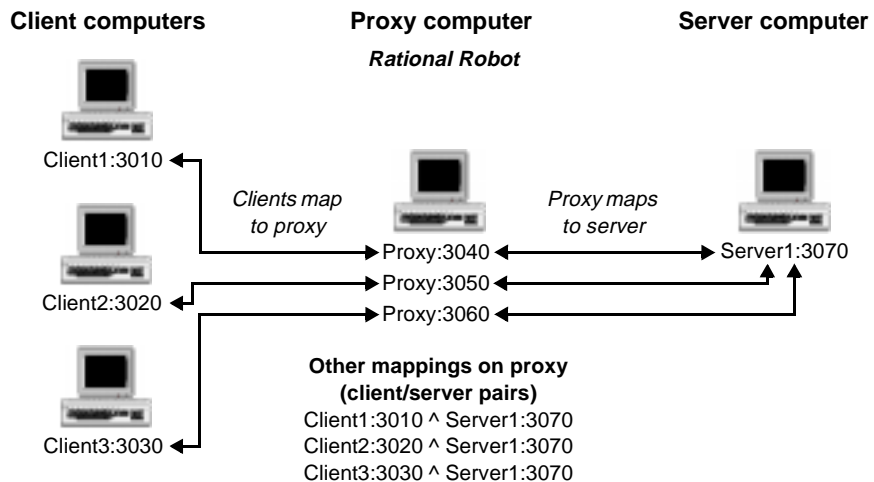
The following examples show when you might need multiple clients and servers:

- ▶ 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** → **Virtual User 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 Virtual User 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 3-8.
3. In the **Method:Proxy** tab of the Virtual User 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 3-8.
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 Virtual User**).

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 **Computer Admin** to display the Virtual User Computer Administration 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 3-27.
- ▶ Proxy service is running. For more information, see *Starting and Stopping Proxy Service* on page 3-32.

To create a proxy computer:

1. Click **Tools** → **Virtual User 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 **Computer Admin** to define them. For information about how to define a client or server computer, see *Defining a Client or Server Computer* on page 3-27.

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

1. Click **Tools** → **Virtual User 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 change a script generation option and regenerate the script with the 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 Oracle or TUXEDO requests—for example, to give emulation commands pertaining to TUXEDO a command ID prefix that reflects the specific emulation command.

For information about regenerating scripts, see *Regenerating the Scripts Recorded in a Session* on page 4-17.

Modifying the Contents of a Script

To modify the contents of a script:

1. Click **Tools** → **Virtual User 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 user running a script to send realistic data to the server. If you do not use a datapool, each virtual user sends the same data to the server (the data that you sent to the server when you recorded the script).

For more information about datapools, see Chapter 6, *Working with Datapools*.

Command ID Prefix

This option specifies an optional prefix for VU 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 LoadTest 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 3-23.

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:

```
sqlxec ["EmulCmd~024"] "select * from table";
sqlnrecv ["EmulCmd~025"] ALL-ROWS;
```

- ▶ Truncation format for multi-script sessions:

```
sqlxec ["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 script playback.

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

Value	Meaning
All	Insert all retrieved data into the script.
First	<p>Insert a specified number of bytes (HTTP and socket protocols) or rows (other protocols) from the beginning of the data retrieved from the server.</p> <p>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:</p> 
Last	<p>Insert a specified number of rows from the end of the set of records retrieved from the server.</p> <p>You cannot use Last with HTTP or socket protocols.</p> <p>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:</p> 
None	Do not insert any retrieved data into the script.

NOTE: **Display recorded rows** is supported only for Sybase, Microsoft SQL Server, HTTP, and socket protocols.

Verify playback row counts

This option specifies whether the number of rows that were affected by an executed 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 a SQL statement or Web request executed during script playback behaves as it did during recording. If the SQL statement or request returns a different number of rows or bytes during playback, LoadTest notes the discrepancy when you run an Analog report.

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 3-13.

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, LoadTest checks whether the return code for a SQL statement executed during script playback matches the return code for the same statement executed during recording. If the SQL statement returns a different code during playback, LoadTest notes the discrepancy in the LoadTest playback report.

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	<pre>sqlexec ["x001"] "INSERT INTO mytable VALUES ('value1', 'value2');"</pre> <p>During recording, Robot does not report the data it collects from the execution of the SQL statement. During playback, LoadTest assumes that any non-zero return code is an error. It pays no attention to the number of affected rows.</p>
Verify playback return codes	<pre>sqlexec ["x001"] EXPECT_ERROR {-212}, "INSERT INTO mytable VALUES ('value1', 'value2');"</pre> <p>Robot records that the error code -212 was returned from the SQL statement. During playback, LoadTest expects the SQL statement to return the error code -212. When you run an Analog report on the log, the report output shows any variance in the error codes.</p>
Verify playback row counts	<pre>sqlexec ["x001"] EXPECT_ROWS 1, "INSERT INTO mytable VALUES ('value1', 'value2');"</pre> <p>Robot records that one row was affected by the SQL statement. During playback, LoadTest expects the SQL statement to affect one row. When you run an Analog report on the log, the report output shows any variance in the number of rows affected.</p>
Both	<pre>sqlexec ["x001"] EXPECT_ERROR {-212}, EXPECT_ROWS 0, "INSERT INTO mytable VALUES ('value1', 'value2');"</pre> <p>Robot records that the SQL statement returned the error code -212, and that no rows were affected. During playback, LoadTest expects that the SQL statement will return error -212 and that no rows will be affected. When you run an Analog report on the log, the report output shows any variance in the error codes and the number of rows affected.</p>

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, LoadTest 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 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	<p>Plays back the script at a rate based on the actual time required to record and process each VU emulation command. For example, if the think time delay for a particular emulation command is 16,703 ms during recording, Robot adds the following line before that emulation command:</p> <pre>push Think_avg 16703;</pre> <p>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.</p>
per script	<p>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.</p> <p>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 script's average think time by editing one <code>Think_avg</code> environment variable.</p>
none	<p>Plays back the script on a per script basis, using the most recently-set value for <code>Think_avg</code> (the value at the top of the <code>Think_avg</code> stack). If there is no value set, the default <code>Think_avg</code> value of 5000 ms (5 seconds) is used.</p> <p>No think time commands are added to the script with this setting.</p>

Setting Recording Options

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 LoadTest 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 you 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 Virtual User Manual Filtering dialog box at the end of the recording session. For more information, see *Choosing the Protocols to Include in a Script* on page 4-11.

How to Filter Protocols

To filter protocols from a recorded script:

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

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	<p>If selected, Robot generates scripts containing requests for the protocols listed in Selected protocols.</p> <p>If cleared, Robot generates scripts containing requests for all scriptable protocols it records.</p>
Manual Filtering	<p>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.</p> <p>When the list appears, select one or more connections to include in the script.</p> <p>If cleared, Robot does not display a list of connections.</p>

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 Virtual User 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 4-11.

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 Virtual User 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, LoadTest 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 userN ame 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.** LoadTest 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.
- **Cltnme** – 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 HTTP, Oracle, TUXEDO, and IIOP 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 virtual user 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** → **Virtual User Record Options**.
2. Click the **Generator per protocol** tab.
3. At the HTTP section, 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 VU 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 VU 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 VU 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 check 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.

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

Setting Recording Options

Before you record a script, you can tell LoadTest 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** → **Virtual User 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_00n`.

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
```

LoadTest 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** → **Virtual User Record Options**.
2. Click the **Generator per protocol** tab.
3. 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 LoadTest 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
<code>tux_bq</code>	<code>tbq</code>	<code>tux_tpinit</code>	<code>tini</code>
<code>tux_tpabort</code>	<code>tabo</code>	<code>tux_tpnotify</code>	<code>tnot</code>
<code>tux_tpacall*</code>	<code>tacaR</code>	<code>tux_tppost</code>	<code>tpos</code>
<code>tux_tpbroadcast</code>	<code>tbro</code>	<code>tux_tprecv</code>	<code>trec</code>
<code>tux_tpcall</code>	<code>tcal</code>	<code>tux_tpresume</code>	<code>tres</code>
<code>tux_tpcommit</code>	<code>tcom</code>	<code>tux_tpsend</code>	<code>tсен</code>
<code>tux_tpconnect</code>	<code>tcon</code>	<code>tux_tpsubscribe</code>	<code>tsub</code>

* `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_tpdequeue	tdeq	tux_tpsuspend	tsus
tux_tpdconnect	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.

Supplying IIOp information

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** → **Virtual User Record Options**.
2. Click the **Generator per protocol** tab.
3. At the IIOp section 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 LoadTest 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.

Setting General Recording Options

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

1. Click **Tools** → **Virtual User 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 Virtual User 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.

Script and session 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.

Setting Recording Options

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 Virtual User 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 Virtual User Recorder window:

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

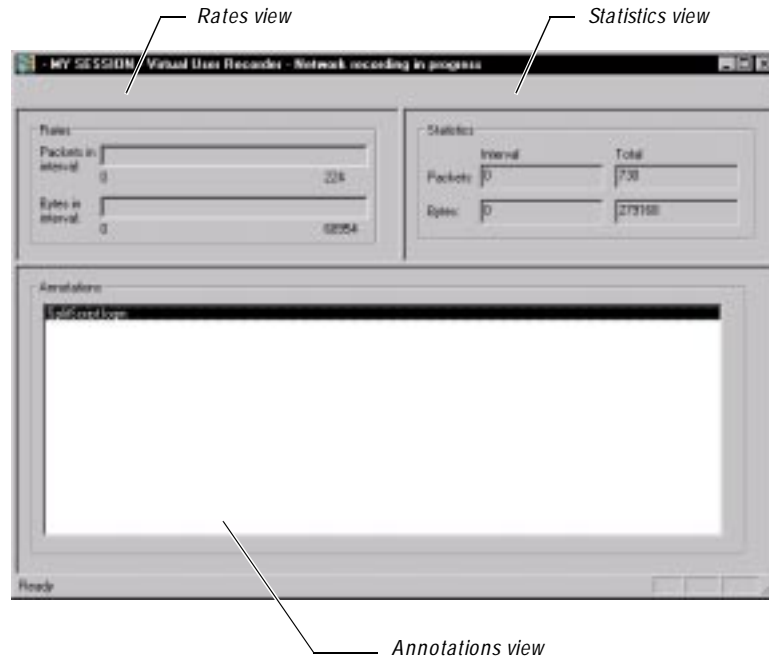


During recording, the Virtual User Recorder icon is displayed in the taskbar, to the left of the clock. The icon blinks when Robot is capturing a request or response. This icon serves as a visual cue that Robot is recording, even when the Virtual User Recorder window is minimized.

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.



For more information about the Virtual User Recorder window, including descriptions of the window's views, see *Getting Feedback During Recording* on page 4-7.

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** → **Virtual User 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 **Computer Admin**.
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.

For information about finding a computer's IP address, see the *Getting Started with Rational PerformanceStudio* manual.

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 repository 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** → **Virtual User Record Options**.
2. Click the **Method:Network** tab.
3. Click **Computer Admin**.
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** → **Virtual User Record Options**.
2. Click the **Method:Network** tab.
3. Click **Computer Admin**.
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 virtual user script. Robot writes the information to the Authentication Datapool. During playback, LoadTest retrieves your user ID and password from the Authentication Datapool and uses the information to log the virtual user 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 **PerformanceStudioAuthentication** (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	<p>Whether the password in this row is ACTIVE or IN ACTIVE. Select one of these choices from the list box.</p> <p>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.</p> <p>If there is more than one active password, Robot uses the first active password that it finds in the Authentication Datapool.</p>
Class	The class is always SQL in this release.
Subclass	<p>One of the following values:</p> <ul style="list-style-type: none"> ▶ Oracle ▶ Sybase ▶ SQL Server
Service	<p>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.</p> <p>This is the same name that Robot inserts into the <i>server</i> argument of the <code>sqlconnect</code> command during recording.</p>
Login	The user ID.
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.

For information about providing login information at the end of the recording process, see *Providing a Missing Password* on page 4-5.

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 `DATAPPOOL_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 PerformanceStudio.
- ▶ Start your system.
- ▶ Open the Virtual User 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** → **Virtual User 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 Virtual User 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 Virtual User 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 3-8.

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 Virtual User 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.
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** → **Virtual User 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** → **Virtual User 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 Virtual User Proxy Administration dialog box appears.
5. In the grid of existing proxies, select the proxy to delete.
6. Click **Delete Proxy**.
7. In the Virtual User 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 Virtual User 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.

Setting Recording Options

When you click the **Method:Proxy** tab of the Virtual User Record Options dialog box, any client/server pairs that are no longer associated with a proxy are displayed in the Virtual User 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.

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.

3. In the **Server:Port** list, select the server to be reassociated with the proxy.
4. Click **Create Proxy**.

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

▶▶▶ C H A P T E R 4

Recording Virtual User Scripts

This chapter describes how to record virtual user 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 virtual user script manually
- ▶ Defining script properties
- ▶ Managing scripts and sessions

Recording a Session

You do not record virtual user scripts 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

Scripts (.s) are stored in the Script directory of your current project. For example, if the current repository is MyRepo and the current project is MyProject, here is what the directory hierarchy looks like:

```
c:\MyRepo\Project\MyProject\Script
```

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

```
c:\MyRepo\Project\MyProject\Session
```

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 VU 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 single virtual user script in a session. For information about splitting a recording session into multiple scripts, see *Splitting a Session into Multiple Scripts* on page 4-15.

To record a virtual user script:



1. In Robot, click the **Record VU Script** button.

Alternatively, click **File** → **Record Virtual User**, 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 virtual user recording options, you can do so now by clicking **Options**. The next step assumes that your options are set.

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

- Robot is minimized (default behavior).
- The floating VU Record 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 4-5.)
- The Virtual U ser Recorder icon appears on the taskbar. The icon blinks as Robot captures requests and responses.
- If the client application is already running, the Virtual User 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 4-7.)
- If the client application is not running, the Virtual U ser Start Application dialog box appears before the Virtual U ser Recorder window appears.



4. If the Virtual U ser 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 Virtual User 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 Virtual User 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 Virtual User Recorder window. Progress bars and request statistics appear in the top of the window.

If there is no activity in the Virtual User Recorder window (or if the Virtual User 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 VU 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 VU 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 4-9.

10. Click **OK**.

The Virtual User Script Generation Feedback 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 Virtual User Missing Password dialog box. For more information about this dialog box, see *Providing a Missing Password* on page 4-5.
- The Virtual User Manual Filtering dialog box. For information about this dialog box, see *Choosing the Protocols to Include in a Script* on page 4-11.

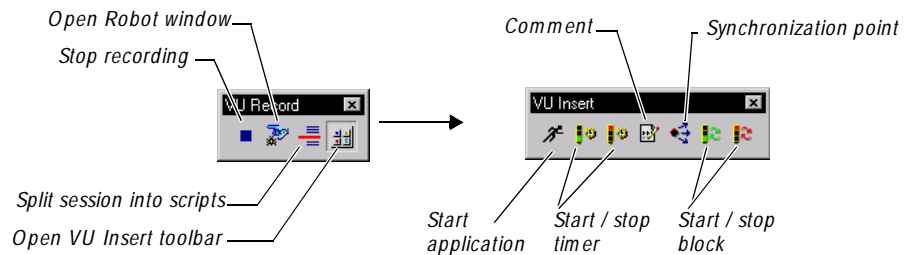
11. Click **OK** in the Virtual User Script Generation Feedback dialog box. The script that you recorded appears in the Robot window.

Using the Floating Toolbars

When you begin to record a script, Robot displays a floating toolbar by default. The VU 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 VU Record toolbar, you display the VU Insert toolbar. This toolbar lets you insert features into the script and start another application during recording.

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



If Script Generation Problems Occur

If problems occur during script generation, the following message appears in the status bar of the Virtual User Script Generation Feedback 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:

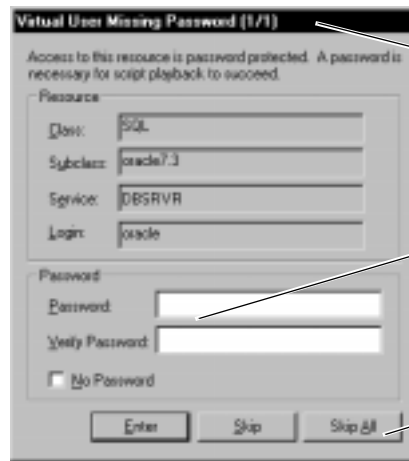
- ▶ Double-click the text to see the entire message.
- ▶ Press CTRL+ C to copy the text to the Clipboard.

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, LoadTest 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 Virtual User 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.



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.*

For more information about the Authentication Datapool, see *Authenticating Login* on page 3-29.

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 virtual user script, the Virtual User Recorder appears, either in a normal or minimized state. You can use this window to monitor client activity during the recording session.

The Virtual User 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 Virtual User Recorder displays its data at three-second intervals. Information in the Virtual User Recorder window is continuously updated as the client/server conversation progresses.

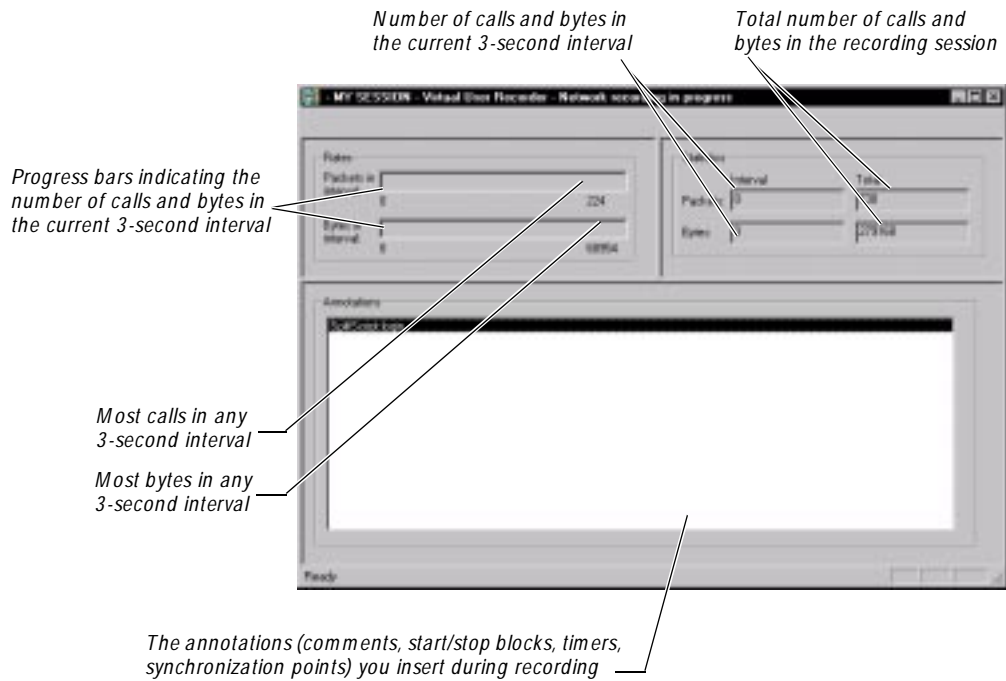
If you do not see any activity in this window as you record, Robot is not capturing client/server traffic. Stop recording and try to determine the cause of the problem.

You can set a variety of options about the appearance of the Virtual User Recorder window. For more information, see *Setting the Recorder Window* on page 3-26.

The Virtual User Recorder During Recording

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

Recording Virtual User Scripts



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 Virtual User Recorder window reports the activity of *all* clients on the network, not just the activity of your computer.

For information about the client/server traffic that you can record, see *Selecting a Client/Server Pair* on page 3-5.

- ▶ With proxy recording, the window displays the number of IP packets and the bytes in these packets.

The Virtual User Recorder Icon



During recording, the icon associated with the Virtual User 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 Virtual User Recorder window is minimized.

The Virtual User Recorder After Recording

The Virtual User 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 VU 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 the Scripts Recorded in a Session* on page 4-17.

Cancelling Scripts During Recording

During virtual user recording, you can cancel scripts that you have recorded. The scripts are deleted.

This feature is useful if you make errors while recording a script, 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 the 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 virtual user recording, click the **Stop** button on the VU 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 virtual user recording, click the **Split Script** button on the VU 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 scripts that you previously recorded in this session, and then stop recording:



1. During virtual user recording, click the **Stop** button on the VU Record floating toolbar.
2. In the Stop Recording dialog box, click **Ignore just-recorded information**.
3. Click **OK**.
4. Click **OK** in the Virtual User Script Generation Feedback 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 virtual user recording, click the **Stop** button on the VU Record floating toolbar.
2. Click **OK** in the Stop Recording dialog box.
3. Immediately click **Cancel** in the Virtual User Script Generation Feedback 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 4-25.

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

For information about setting automatic or manual filtering, see *Setting Filtering Options* on page 3-16.

Manually Filtering Protocols

The Virtual User 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.

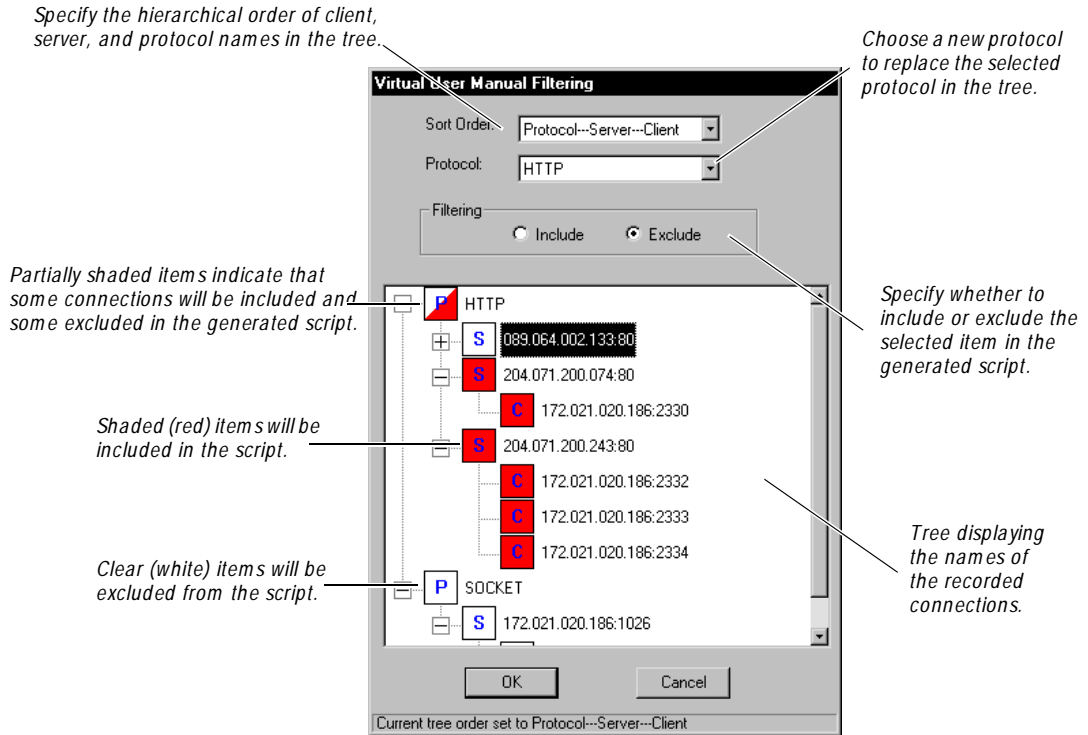
Controls in the Virtual User Manual Filtering Dialog Box

The Virtual User 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.
Tree 	<p>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.</p> <p>Items are marked for inclusion or exclusion as follows:</p> <ul style="list-style-type: none"> ▶ 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. <p>When you click an item in the tree, you select that item and also any items below that item.</p> <p>Click the + icon to expand a branch, and click the - icon to close a branch.</p> <p>To change the hierarchical order of the protocol, client, and server names, select a new order in Sort Order.</p> <p>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.</p>
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 Virtual User Manual Filtering dialog box, with the components of the connection sorted by protocol, then server, then 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 Virtual User 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 Virtual User Manual Filtering dialog box to convert the requests captured during a connection from one protocol type to another.

To convert a protocol in the Virtual User 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 as 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 schedule, 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 virtual user script to play back.
3. Click **OK**.

LoadTest appears, ready to play back the script that you selected.

4. In LoadTest, click **Run** → **Schedule**.
5. Click **OK** in the Run Schedule dialog box.

Working with Sessions

A virtual user 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 VU 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 4-9.

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 repository. For example, assume someone at another site e-mails you a session file. You can import this session file, regenerate scripts, and create a schedule.

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 repository.
3. To regenerate the scripts in the session you imported, click **Tools** → **Regenerate VU Scripts from Session**, and select the session you imported.
4. To regenerate the schedule, click **Tools** → **Rational Test** → **LoadTest**.

5. Click **File** → **New** → **Schedule**. The New Schedule dialog box appears.
6. Select **Existing Session**, and click **OK**.
7. LoadTest displays a list of sessions that are in the repository. Click the name of the session that you imported, and click **OK**.

LoadTest automatically creates a schedule that is ready to run.

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

Regenerating the Scripts Recorded in 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 VU Scripts from Session**.
2. Click the name of the session to use for script regeneration.

You can regenerate scripts that are contained within a session. You cannot regenerate scripts 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 Virtual User Script Generation Feedback 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), script regeneration ends, the message *Completed successfully* appears, and the **OK** button is enabled.

4. Click **OK** to acknowledge that the script 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 Virtual User 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

For information about recording options, see *Setting Recording Options* on page 3-1.

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, remember that you overwrite all of the scripts in the session.

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

1. Click **Tools** → **Virtual User 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 VU 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.

Unless you are regenerating a session's scripts, you should view session properties in TestManager. (Click **View** → **Asset Browser** and open **Session Queries**.)

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

1. Click **Tools** → **Regenerate VU 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 VU Scripts from Session dialog box, click **OK** to regenerate the session's scripts, or click **Cancel** to close the dialog box without regenerating the session's scripts.

NOTE: Once you click **OK** in the Regenerate VU Scripts from Session dialog box, the existing scripts in the session are destroyed. If you then click **Cancel** in the Virtual User Generation Feedback 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.

Unless you are regenerating a session's scripts, you should view session properties in TestManager. (Click **View** → **Asset Browser** and open **Session Queries**.)

To view script properties just before you regenerate a session's scripts in Robot:

1. In Robot, click **Tools** → **Regenerate VU 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.

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

7. When you have finished viewing and editing script 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 VU Scripts from Session dialog box, click **OK** to regenerate the session's scripts, or click **Cancel** to close the dialog box without regenerating the session's scripts.

Coding a Virtual User Script Manually

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

However, you can open an empty virtual user 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 virtual user script and add code to it:

1. In Robot, click **File** → **New** → **Script**.
2. Type a script name and, optionally, a description of the script.
3. Click **Virtual User**.
4. Click **OK**. Robot creates an empty script with the following lines:

```
#include <VU.h>
{
}
```

5. Add the code to the virtual user script.

For information about using the VU scripting language, see the *VU Language Reference*.

Creating Library Files

VU 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** → **Script** to open the Open 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 LoadTest schedules 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** → **Asset Browser**.
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 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** → **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

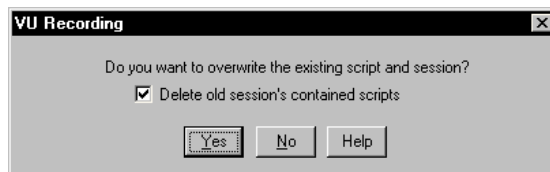
If you remove a script from a session, you can no longer regenerate that script if you regenerate the session.

To remove a script from a session:

1. In Robot, click **File** → **Open** → **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 an existing 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:



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 Virtual User**.
2. In the Record Virtual User - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the VU 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 Virtual User**.
2. In the Record Virtual User - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the VU 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 schedule.

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 Virtual User**.
2. In the Record Virtual User - Enter Session Name dialog box, select the name of the session to re-record, and then click **OK**.
3. In the VU 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 Script or Stop Recording dialog box).

Also, if you plan a script in TestManager, the name of the planned script appears in the list of existing scripts 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: <ul style="list-style-type: none"> ▶ 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** → **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 script. In addition, the new script 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 to delete.

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 TestManager, click **View** → **Asset Browser**.
2. Double-click **All** under **Session Queries**.
3. Right-click the name of the session to delete, and then click **Delete**.
4. Click **Contained Scripts**.

Recording Virtual User Scripts

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

Adding Features to Virtual User Scripts

This chapter describes the features that you can add to a virtual user script while recording the script 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 LoadTest report output.

However, if you want to measure the time it takes a virtual user 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.

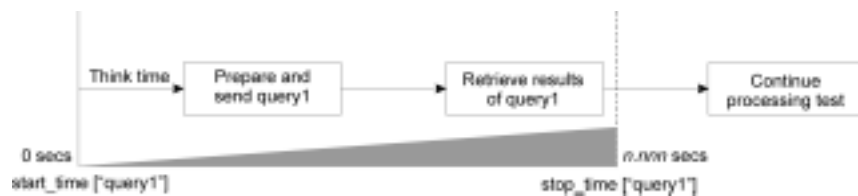
When you stop a timer, you can reuse that timer's name in another timer. There is no practical limit to the number of timers that you can add to a script.

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?

You need to use timers in the following cases:

- ▶ You want to 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).
- ▶ You want to 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 virtual user script as follows:



1. If the VU Insert floating toolbar is not already displayed, click the **Display VU Insert Toolbar** button on the VU 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 VU 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 Virtual User Recorder Annotations window.

Adding a Timer During Editing

The VU 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 virtual user 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 details about using these commands, see the *VU Language Reference*.

NOTE: Inserting timers into GUI scripts is similar to inserting timers into virtual user scripts. However, GUI scripts use a different set of timer commands (`StartTimer` and `StopTimer`).

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 following comment:

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

- ▶ Robot automatically starts a timer at the start of the block:

```
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:

```
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`. (For more information about command IDs, see *Command ID Prefix* on page 3-10.)

- ▶ A block ends with a `stop_time` command plus a comment:

```
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 LoadTest 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 VU Insert floating toolbar is not already displayed, click the **Insert** button on the VU 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 5-6.

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 Virtual User Recorded Annotations window.

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

Nesting Blocks



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

When you nest blocks:

- ▶ Robot inserts the `start_time` command at or near the beginning of the second block.
- ▶ Timing continues on the first block (in other words, `stop_time` 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 the `stop_time` 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 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 users by pausing the execution of each user at a particular point—the synchronization point—until one of the following events occur:

- ▶ All users associated with the synchronization point arrive at the synchronization point.
- ▶ A timeout period is reached before all users arrive at the synchronization point. You specify the timeout period in the LoadTest Synchronization Point dialog box.
- ▶ You manually release the users while monitoring a schedule run in LoadTest.

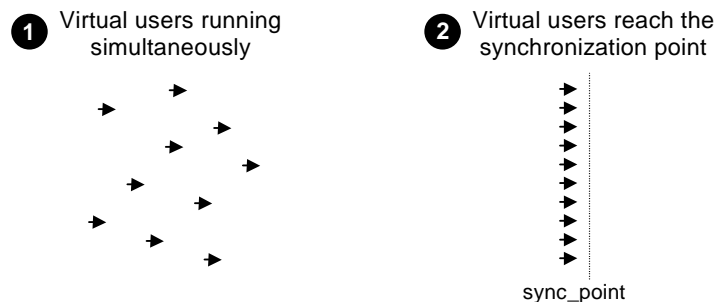
When one of the above events occurs, LoadTest releases the users, allowing them to continue performing the transaction.

Both virtual users and GUI users support synchronization points.

How Synchronization Points Work

At the start of a test, all users 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** (virtual user scripts) or **SQASyncPointWait** (GUI scripts). In a schedule, a synchronization point is a time specified in the schedule.

The following figure illustrates a synchronization point in a virtual user script:



The users pause at the synchronization point until LoadTest releases them. Typically, LoadTest releases synchronized users when they all arrive at the synchronization point.

Why Use Synchronization Points?

By synchronizing users 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 user load on the timed activity. For example, to the effect of user load on data retrieval, you could take the following general steps:

1. While recording the virtual user script (named VU 1 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 5-5).

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 user that will provide the user load, 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 VU 2.
3. Add VU 1 and VU 2 to a schedule.
4. Run the schedule a number of times, each time using a different number of the VU 2 users. However, you only need one VU 1 user in each test.

Theoretically, as the number of synchronized VU 2 users increases, the time reported by the VU 1 timer should also increase.

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

- ▶ All VU 2 virtual users submit their forms at the same time—thereby providing maximum concurrent user load.
- ▶ The VU 1 user submits its query at the same time that the VU 2 users are loading the server—thereby providing maximum user load at a critical time.

Inserting Synchronization Points

You can insert a synchronization point into a script (through Robot) or into a schedule (through LoadTest).

- ▶ **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 schedule** – You can insert a synchronization point into a schedule through the LoadTest Synchronization Point dialog box.

Insert a synchronization point into the schedule to pause execution before or between scripts rather than within a script. In addition, inserting a synchronization point into a schedule 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 schedule rather than hidden within a script.

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

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

Inserting a Synchronization Point During Recording

To insert a synchronization point into a virtual user script during recording:



1. If the VU Insert floating toolbar is not already displayed, click the **Insert** button on the VU 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 users 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 Virtual User Recorded Annotations window.

Inserting a Synchronization Point During Editing

You can only use the VU 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 the VU command `sync_point` into the script.

For details about using this command, see the *VU Language Reference*.

NOTE: To insert a synchronization point into a GUI script, enter the synchronization point command `SQASyncPointWait` into the script. You cannot insert a synchronization point into a GUI script during recording.

Inserting a Synchronization Point During Scheduling

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

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

If the users are to be released at different times (that is, in a staggered fashion), you can specify the minimum and maximum times within which all users must be released.
- ▶ You can specify a timeout period.

For more information about inserting a synchronization point in a schedule, see *Inserting a Synchronization Point* on page 7-25

Release Times and Timeouts for Synchronization Points in 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 schedules). By default:

- ▶ Virtual users held at a script-based synchronization point are released simultaneously.

- ▶ There is no time limit to how long virtual users can be held at the synchronization point.

However, if a synchronization point in a schedule 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 that reference a particular synchronization point name plus all user groups that reference that name.

For example, suppose you have the following user groups in a schedule:

- ▶ A Data Entry user group of 75 virtual users. This user group runs a script containing the synchronization point Before Query.
- ▶ An Engineering user group of 10 virtual users. 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 users. 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 schedule runtime, LoadTest releases the users held at the Before Query synchronization point when all 110 users arrive at their respective synchronization points.

Comments

A **comment** is a line of text in a script that begins with the characters `/*` and ends with the characters `*/`—for example:

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

Comments are ignored at script compile time and during script playback.

Use comments to document the script and to help you find your way around the script if you later need to edit it.

Adding Comments During Recording

To insert a comment into a virtual user script during recording:



1. If the VU Insert floating toolbar is not already displayed, click the **Insert** button on the VU 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 Virtual User Recorded Annotations window.

Adding Comments During Editing

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

Be sure to begin the comment with the `/*` characters, and end the comment with the `*/` characters.

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 VU Insert floating toolbar to add start timers and stop timers, synchronization points, start blocks and stop 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 VU Record floating toolbar. This button restores the Robot window, letting you access the Insert menu.

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 virtual user scripts
- ▶ Using datapools with GUI scripts
- ▶ Managing datapools with TestManager
- ▶ Managing user-defined data types
- ▶ Generating and retrieving unique datapool rows
- ▶ Creating a datapool outside Rational Test
- ▶ Creating a column of values outside Rational Test

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 virtual user and GUI scripts played back in a LoadTest schedule. For information about datapool access from GUI scripts played back in Robot, see the datapools chapter in the *Using Rational Robot* manual.

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 users and GUI users under high-volume conditions that potentially involve hundreds of users performing thousands of transactions.

Typically, you use a datapool so that:

- ▶ Each user that runs the script can send realistic data (which can include unique data) to the server.
- ▶ A single user 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 user sends the same literal values to the server (the values that were captured when you recorded the script).

For example, suppose you record a virtual user script that sends order number 53328 to a database server. If 100 virtual users run this script, order number 53328 is sent to the server 100 times. If you use a datapool, each virtual user 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 virtual user script.	•	
Create a datapool and automatically generate datapool values.	•	•
Edit the DATAPOOL_CONFIG section of a virtual user script.	•	
Edit datapool column definitions and datapool values.	•	•

(Continued)

Activity	Robot	TestManager
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 *Importing a Datapool* on page 6-35 and *Creating a Datapool Outside Rational Test* on page 6-44.

NOTE: LoadTest automatically copies a .csv file to each Agent computer that uses it. If an Agent's .csv file becomes out-of-date, LoadTest automatically updates it.

Datapool Cursor

The datapool **cursor**, or row-pointer, 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 schedule runs. For example, suppose the last datapool row accessed in a schedule run is row 100:

- ▶ If the cursor is persistent across schedule runs, datapool row access resumes with row 101 the first time the datapool is accessed in a new schedule run.
- ▶ If the cursor is not persistent, datapool row access resumes with row 1 the first time the datapool is accessed in a new schedule run.

NOTE: GUI users can share a cursor when playback occurs in a LoadTest schedule, 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 6-16.

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 virtual user scripts, you can control row access order through the **Access Order** setting in the Robot Configure Datapool in Script dialog box. (See page 6-17.)

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 user 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 user 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 virtual user 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 virtual user 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 virtual user 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 6-10.

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 virtual user 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 *virtual user 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 Virtual User 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 6-24.

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 virtual user 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 *Creating a Datapool with TestManager* on page 6-25.

Creating and populating a datapool for a virtual user 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 virtual user script.

For information about editing the `DATAPOOL_CONFIG` section of a script, see *Step 1. Editing Datapool Configuration* on page 6-14.

- 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 6-19.

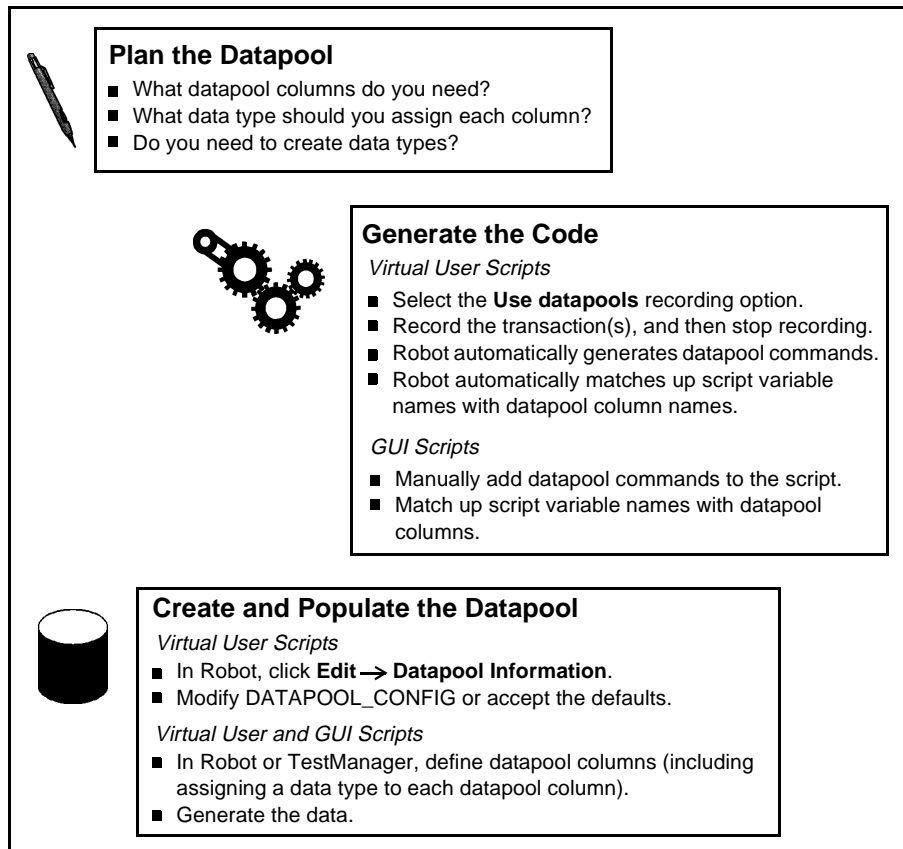
You also assign a data type to each datapool column. Data types supply a datapool column with its values. For information about data types, see *Data Types* on page 6-9.

- 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 repository. For more information, see *Creating a Datapool Outside Rational Test* on page 6-44.

The following figure illustrates the three stages of datapool creation:

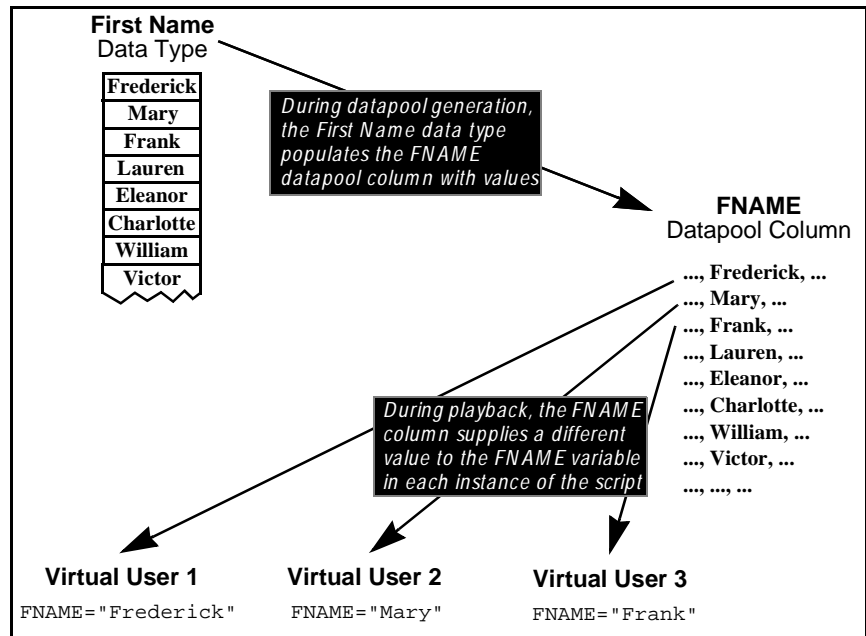


Data Types

A datapool **data type** is a source of data for one datapool column.

For example, the **Names - First** data type (shipped with Rational Test as a standard data type) contains a list of persons' first names. Suppose you assign this data type to the datapool column **FN AME**. When Robot automatically generates the datapool, it populates the **FN AME** column with all of the values in the **Names - First** data type.

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 6-13.

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 virtual user scripts, you can view the DATAPOOL_CONFIG section that Robot automatically adds to the end of the script. (Robot adds this information to a virtual user script when you select **Use datapools** in the **Generator** tab of the Virtual User 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 6-14.

- ▶ 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"
ComboBox Click, "ObjectIndex=1", "Text=Discover"

'Credit Card Expiration Date
EditBox Left_Drag, "ObjectIndex=4", "Coords=19,13,16,12"
InputKeys "12/31/99"
```

To 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.

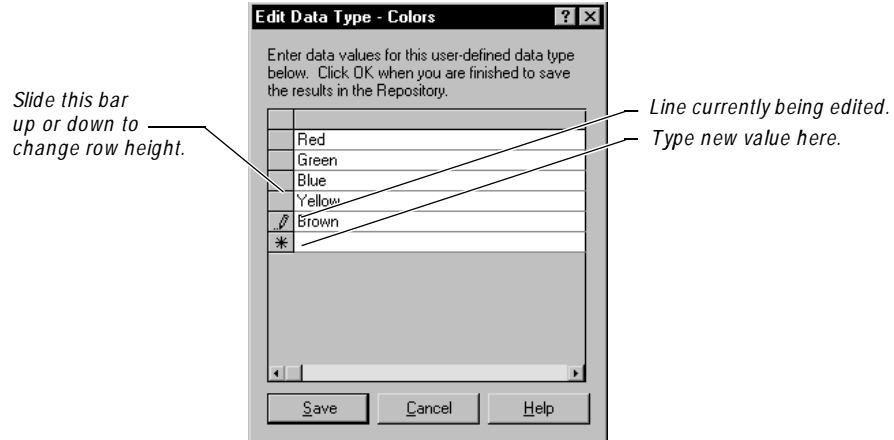


7. To type a new value, place the insertion point on the blank line next to the asterisk icon, and then type the value.

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:



When you create a user-defined data type, it is listed in the **Type** column of the Datapool Specification dialog box (where you define datapool columns). **Type** also includes the names of all the standard data types. User-defined data types are flagged in this list with an asterisk (*).

NOTE: You can assign data from a standard data type to a user-defined data type. For information, see *Editing User-Defined Data Type Definitions* on page 6-39.

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 *Defining Datapool Columns* on page 6-27.

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 6-11.

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 *Creating a Column of Values Outside Rational Test* on page 6-48.

Using Datapools with Virtual User Scripts

Robot can insert datapool commands into a virtual user script automatically. If you want Robot to do so, take these steps before you begin to record a virtual user script:

1. In Robot, click **Tools** → **Virtual User 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 virtual user 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 6-19.)

You cannot automatically generate data for a datapool that has more than 150 columns.

NOTE: `DATAPOOL_CONFIG` is included only in virtual user 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 6-24.

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 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** → **Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in 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 Script dialog box, or make any appropriate changes.

Use the table on page 6-16 to help you modify the settings in this dialog box.

NOTE: By default, the **Usage** column (see page 6-18) 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 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 6-19.

Here is how the Configure Datapool in Script dialog box maps to the DATAPOOL_CONFIG section of a script:

Because *Obey Usage* is selected and the *Persistent* check box is not checked, no other flag is used.

Usage	Name	Type	Script Data
INCLUDE	CUSTOMER_ID	string	329781
INCLUDE	PRODUCTS_COMPOSER	string	Bech
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	VOBS	string	33822
INCLUDE	VOBS_1	string	84
INCLUDE	VOBS_2	string	2
INCLUDE	VOBS_3	string	25-APR-1998
INCLUDE	VOBS_4	string	MasterCard
INCLUDE	VOBS_5	string	1234567890010
INCLUDE	VOBS_6	string	112250
INCLUDE	VOBS_7	string	20
INCLUDE	VOBS_8	string	9379.80
INCLUDE	VOBS_9	string	Order Initiated
EXCLUDE	VOBS_10	string	Order Initiated

These tables match row for row and column for column.

```

DATAPOOL_CONFIG "Classics Orders" DP_SHUFFLE DP_SHARED DP_WRAP
INCLUDE "CUSTOMER_ID", "string", "329781";
INCLUDE "PRODUCTS_COMPOSER", "string", "Bech";
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 "VOBS1", "string", "33822";
INCLUDE "VOBS1_1", "string", "84";
INCLUDE "VOBS2", "string", "2";
OVERWRITE "VOBS3", "string", "25-APR-1998";
INCLUDE "VOBS4", "string", "MasterCard";
INCLUDE "VOBS5", "string", "1234567890010";
INCLUDE "VOBS6", "string", "112250";
INCLUDE "VOBS7", "string", "20";
INCLUDE "VOBS8", "string", "9379.80";
EXCLUDE "VOBS10", "string", "Order Initiated";
  
```

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 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 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?	<p>Sets the action to take after the last row in the access order is reached:</p> <ul style="list-style-type: none"> ▶ Yes – Resume at the beginning of the access order. ▶ No – End access to the datapool. <p>If you attempt to retrieve a datapool value after the end of the datapool is reached, a runtime error occurs.</p> <p>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.</p>
Cursor	<p>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 schedule runs:</p> <ul style="list-style-type: none"> ▶ 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 schedule 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 schedule run, the first row accessed in the next schedule run is 101. <p>A persistent cursor resumes row access based on the last time the cursor was accessed as a <i>persistent</i> cursor. For example, suppose a cursor is persistent, and the last row accessed for that cursor in a schedule run is 100. Then, the same schedule 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 schedule is run, row access resumes with row 101, not 51.</p> <p>With persistent cursors, you can use the Row box to set the row to be accessed first in the next test run.</p> <p>Persistent cursors are only valid with shared cursors, and when Access Order is set to either Sequential or Shuffle.</p> <ul style="list-style-type: none"> ▶ 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	<p>Determines the sequence in which datapool rows are accessed:</p> <ul style="list-style-type: none"> ▶ 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 LoadTest rearranges, or “shuffles,” the access order of all datapool rows, a unique sequence results. Each row is referenced in a shuffled sequence only once. <p>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.</p> <p>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).</p> <p>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).</p>
Use Script Data	<p>Specifies the source of the values that script variables are assigned during schedule runtime, as follows:</p> <ul style="list-style-type: none"> ▶ Always – Script variables are assigned the values provided during recording rather than values from the datapool. Recorded values are listed in the Script Data column. <ul style="list-style-type: none"> This option overrides the runtime meaning of the <code>INCLUDE</code> directive in the Usage column. It also adds the flag <code>OVERRIDE</code> to the <code>DATAPOOL_CONFIG</code> 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 <code>INCLUDE</code> directive in the Usage column are assigned datapool values. Script variables not associated with the <code>INCLUDE</code> directive are assigned values in the Script Data column. <ul style="list-style-type: none"> No flag is added to <code>DATAPOOL_CONFIG</code> with this option.

(Continued)

Field or column	Description
Datapool	<p>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:</p> <ul style="list-style-type: none"> ▶ 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. <p>After you specify a starting row number, click Set Cursor.</p> <p>Any changes you make in the Datapool group box do not affect the DATAPPOOL_CONFIG section of the script.</p>
Usage	<p>Specifies one of the following directives to apply during database creation and during schedule runtime. To change an individual directive, right-click the directive name:</p> <ul style="list-style-type: none"> ▶ INCLUDE <ul style="list-style-type: none"> - During datapool creation, creates a column for the script variable in Name. The column is assigned the same name. - During schedule runtime, assigns a value to the script variable in Name from the corresponding datapool column. <p>You can override the runtime meaning of all INCLUDE directives by selecting Always in the Use Script Data group box. With Always selected, all script variables are assigned the associated values in the Script Data column.</p> ▶ EXCLUDE <ul style="list-style-type: none"> - During datapool creation, does not create a column for the script variable in Name. - During schedule runtime, assigns the value in Script Data to the script variable in Name. Datapool values are not used. ▶ OVERRIDE <ul style="list-style-type: none"> - During datapool creation, creates a column for the script variable in Name. The column is assigned the same name. - During schedule runtime, assigns the value in Script Data to the script variable in Name. Datapool values are not used. <p>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.</p>

(Continued)

Field or column	Description
Name	<p>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 <code>INCLUDE</code> or <code>OVERRIDE</code>), the datapool column is assigned the same name.</p> <p>This value can only be modified in the script.</p>
Type	<p>The data type of the value in Script Data. The data type is always <code>string</code>.</p> <p>This value can only be modified in the script.</p>
Script Data	<p>A value that was provided during recording. The value was assigned to the script variable in Name.</p> <p>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.</p> <p>This value can only be modified in the script.</p>

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 6-14, 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 6-14 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 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 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 *Defining Datapool Columns* on page 6-27.

To see an example of datapool columns defined in the Datapool Specification dialog box, see *Example of Datapool Column Definition* on page 6-30.

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 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 Script dialog box.

This section provides the basic steps for editing datapool column definitions while in Robot. For information about the Configure Datapool in Script dialog box, see *Step 1. Editing Datapool Configuration* on page 6-14.

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** → **Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in Script dialog box.
3. Either accept the defaults in the Configure Datapool in Script dialog box, or make any appropriate changes.

Use the table on page 6-16 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 *Defining Datapool Columns* on page 6-27.

To see an example of datapool columns defined in the Datapool Specification dialog box, see *Example of Datapool Column Definition* on page 6-30.

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 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 Script dialog box.

This section provides the basic steps for editing datapool values while in Robot. For information about the Configure Datapool in Script dialog box, see *Step 1. Editing Datapool Configuration* on page 6-14.

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** → **Script** to open it.
2. Click **Edit** → **Datapool Information** to open the Configure Datapool in Script dialog box.
3. Either accept the defaults in the Configure Datapool in Script dialog box, or make any appropriate changes.

Use the table on page 6-16 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 *Editing Datapool Values with TestManager* on page 6-34.

7. When finished editing datapool values, click **Save**, and then click **Close**.

For an example of the datapool values that TestManager generates, see *Example of Datapool Value Generation* on page 6-32.

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 LoadTest schedule, the GUI script can access the same datapool as other GUI scripts and/or virtual user scripts.

There are differences in the way GUI scripts and virtual user 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 virtual user scripts automatically.
- ▶ There is no `DATAPPOOL_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 virtual user 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 the datapools chapter in the *Using Rational Robot* manual.

For information about creating a datapool with `TestManager`, see *Creating a Datapool with TestManager* on page 6-25.

Accessing a Datapool from GUI and Virtual User Scripts

If a GUI script and a virtual user script provide the same set of values to the client application during recording, the scripts can access the same datapool during playback in a LoadTest schedule.

Here is the suggested order of steps for creating a datapool and setting up access to it from GUI scripts and virtual user scripts:

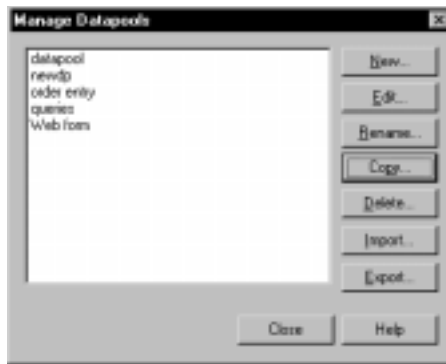
1. Record the virtual user procedure and create the datapool as described in *Using Datapools with Virtual User Scripts* on page 6-13.
2. Record the GUI script.
3. Edit the GUI script and add datapool commands to it.

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 GUI users and virtual users each retrieve a unique row of data from the datapool, follow the guidelines listed in *Generating and Retrieving Unique Datapool Rows* on page 6-42.

Managing Datapools with TestManager

You manage datapools in the TestManager Manage Datapools dialog box:



The activities you perform in this dialog box affect datapools stored in the repository. For information about where datapools are stored, see *Datapool Location* on page 6-36.

Creating a Datapool with TestManager

To create and automatically populate a datapool with TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Click **New**.
3. Type a name for the datapool (40 characters maximum) and optionally, a description (255 characters maximum).
4. Click **OK**.
5. Click **Yes** to acknowledge that you want to define the datapool now.

The Datapool Specification dialog box appears. This dialog box lets you define the columns in the datapool file. Datapool column definitions are listed as rows in this dialog box. Datapool columns are also called *fields*.

6. Click **Insert before** or **Insert after** to add a datapool column to the datapool.

7. Type a name for the new datapool column (40 characters maximum).
With virtual user scripts, datapool column names must match the names of the script variables that they supply values to. Names are case-sensitive.
8. Assign a data type to the datapool column, and define any other properties as necessary.
For information about the properties you can define for a datapool column, see *Defining Datapool Columns* on page 6-27.
9. Repeat steps 6 through 8 until you have defined all the columns in the datapool.
To see an example of datapool columns defined in the Datapool Specification dialog box, see *Example of Datapool Column Definition* on page 6-30.
10. Type a number in **No. of records to generate**.
Alternatively, if you do not want to generate any data now, click **Save** to save your datapool column definitions, and then click **Close**.
11. When finished defining datapool columns, click **Generate Data**.
You cannot generate data for a datapool that has more than 150 columns.
12. Optionally, click **Yes** to see a brief summary of the generated data.

If There Are Errors

If the datapool values are not successfully generated, you are prompted to see an error report rather than a summary of the generated data. To correct the errors:

1. Click **Yes** to see the error report.
2. After viewing the cause of the errors, click **Cancel**.
3. Correct the errors in the **Datapool Fields** grid.

Viewing Datapool Values

To see the generated values, close the Datapool Specification dialog box. In the Manage Datapools dialog box, select the datapool you just created, click **Edit**, and then click **Edit Datapool Data**.

If a datapool includes complex values (for example, embedded strings, or field separator characters included in datapool values), you should view the datapool values to make sure the contents of the datapool are as you expect.

Making the Datapool Available to a Script

For a script to be able to access the datapool you create with TestManager, the script must contain datapool commands, such as commands for opening the datapool and fetching values. Virtual user scripts must also contain `DATAPPOOL_CONFIG`.

You can add datapool commands and `DATAPPOOL_CONFIG` to a script either before or after you create the datapool with TestManager:

- ▶ For information about automatically adding datapool commands and `DATAPPOOL_CONFIG` to a virtual user script during recording, see *Using Datapools with Virtual User Scripts* on page 6-13.
- ▶ For information about adding datapool commands to a GUI script, see *Using Datapools with GUI Scripts* on page 6-24.

Defining Datapool Columns

Use the following table to help you define datapool columns in the Datapool Specification dialog box:

Grid column	Description
Name	<p>The name of a datapool column (and its corresponding script variable).</p> <p>If you change the name of a datapool column, be sure the new name matches all instances of its corresponding script variable.</p> <p>If you create a datapool outside of the Rational Test environment and then import it, TestManager automatically assigns default names to the datapool columns. Use Name to match the imported datapool column names with their corresponding script variables. Names are case-sensitive.</p> <p>You can use an IME to type multi-byte characters in datapool field names.</p> <p>(With GUI scripts, you can associate datapool column names and script variables through column position rather than column name. For more information, see the description of the <code>SQADatapoolValue</code> command in the <i>SQABasic Language Reference</i>.)</p>

(Continued)

Grid column	Description
<p>Type</p>	<p>The standard or user-defined data type that supplies values to the datapool column in Name. User-defined data types are marked with an asterisk (*).</p> <p>Specify the data type to assign to the datapool column, as follows:</p> <ul style="list-style-type: none"> ▶ To select a standard data type or an existing user-defined data type, click the currently displayed data type name, and then select the new data type from the drop-down list: <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 5px 0;"> Random alphanumeric stri ▼ </div> <p>See Appendix C for a description of the standard data types.</p> <p>If you type rather than select the name of a user-defined data type, enter an asterisk before the user-defined data type name. For example, to specify the user-defined data type MyData, type:</p> <p style="padding-left: 20px;">*MyData</p> <ul style="list-style-type: none"> ▶ To create a new user-defined data type, enter the data type name (without the asterisk) in the field, and then press RETURN. After you click Yes to confirm that you want to create a user-defined data type, the Data Type Properties - Edit dialog box appears. <p>For information about creating a data type, see <i>Creating User-Defined Data Types</i> on page 6-11.</p>
<p>Sequence</p>	<p>The order in which the values in the data type specified in Type are written to the datapool column. Select one of these options from the drop-down list:</p> <ul style="list-style-type: none"> ▶ Random – Writes numeric and alphanumeric values to the datapool column in any order. ▶ Sequential – Writes numeric values sequentially (for example, 0, 1, 2...). With decimal numbers, the sequence is based on the lowest possible decimal increment (for example, with a Decimals value of 2, the sequential values are 0.00, 0.01, 0.02, ...). <p>Sequential is only supported for numeric values (including date and time values) and values generated from user-defined data types.</p> <p>When you select Sequential with numeric data types, and you specify a Minimum and Maximum range, Interval must be greater than 0.</p> <ul style="list-style-type: none"> ▶ Unique – With data type Integers - Signed, ensures that numbers written to the datapool column are unique. Also, set Repeat to 1, and define a Minimum and Maximum range. <p>Do not confuse the Random and Sequential settings in this grid with Random and Sequential access order in the Configure Datapool in Script dialog box. The Random and Sequential settings in this grid determine the order in which values are written to an individual datapool column at datapool creation time. Random and Sequential access order determine the order in which users access datapool rows at schedule runtime.</p>

(Continued)

Grid column	Description
Repeat	<p>The number of times a given value can appear in a datapool column. Repeat cannot be set to 0.</p> <p>To make values unique with Integers - Signed data types and user-defined data types, set Repeat to 1. For unique Integers - Signed values, also set Sequence to either Sequential or Unique.</p> <p>When defining unique values, make sure the number of rows you are generating is not higher than the range of possible unique values.</p>
Length	<p>The maximum number of characters that a value in the datapool column can have. If the datapool column contains numeric values, Length specifies the maximum number of characters a number can have, <i>including</i> a decimal point and minus sign, if any.</p> <p>For example, for decimal numbers as high as 999.99, set Length to 6. For decimal numbers as low as -999.99, set Length to 7.</p> <p>Length cannot be 0.</p>
Decimals	<p>Specifies the maximum number of decimal places that floating point values can have. Maximum setting is 6 decimal places.</p>
Interval	<p>Writes a sequence of numeric values to the datapool column. The sequence increments by the Interval you set. For example, if Interval is 10, the datapool column contains 0, 10, 20, and so on. If Interval is 10 and Decimal is 2, the datapool column contains 0.00, 0.10, 0.20, and so on.</p> <p>Minimum interval is 1. Maximum interval is 999999.</p> <p>With numeric data types (including dates and times), when Sequence is set to Sequential and you specify a Minimum and Maximum range, Interval must be greater than 0.</p> <p>Use Interval only with numeric values (including dates and times).</p>
Minimum	<p>Specifies the lowest in a range of numeric values. For example, if the datapool column supplies order number values, and the lowest possible order number is 10000, set Type to Integer - Signed, Minimum to 10000, and Maximum to the highest possible order number.</p> <p>Use Minimum only with numeric values (including dates and times).</p>
Maximum	<p>Specifies the highest in a range of numeric values. For example, if the datapool column supplies values to a variable named <i>ounces</i>, set Type to Integer - Signed, Minimum to 0, and Maximum to 16.</p> <p>Use Maximum only with numeric values (including dates and times).</p>

(Continued)

Grid column	Description
Seed	<p>The number that Rational Test uses to compute random values. The same seed number always results in the same random sequence. To change the random sequence, change the seed number.</p> <p>When Type is String Constant, use Seed to provide the alphanumeric value of the constant. For example, if you want to insert the constant “Rational Software” into every row of a datapool column, set Type to String Constant and type Rational Software into Seed.</p>
Data File	<p>The path to the user-defined data type file. The path is automatically inserted for you. This field is not modifiable.</p> <p>Data type files are stored in the Datatype directory of your project. You never have to modify these files directly.</p>

Some items might not be modifiable, depending on the data type you select. For example, if you select the Names - First data type, you cannot modify **Decimals**, **Interval**, **Minimum**, or **Maximum**.

If you are generating unique values for an Integers - Signed data type, **Length**, **Minimum**, **Maximum**, and **No. of records to generate** must be consistent. For example, if you want unique numbers from 0 through 999, errors may result if you set **Length** to 1, **Maximum** to 5000, and/or **No. of records to generate** to a number greater than 1000.

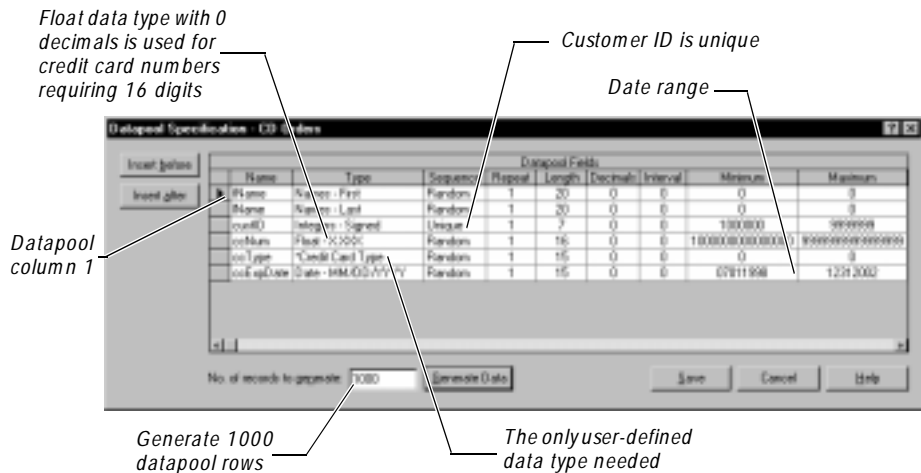
NOTE: You can use an IME to type multi-byte characters into the **Name** column only. The IME is automatically disabled when you are editing any other column.

Example of Datapool Column Definition

Suppose you want to record a transaction in which a customer purchase is entered into a database. During recording, you supply the client application with the following information about the customer:

- ▶ Customer name
- ▶ Customer ID
- ▶ Credit card number
- ▶ Credit card type
- ▶ Credit card expiration date

After you record the script, you are ready to create the datapool. Following the instructions on page 6-25, you define the datapool's columns in the Datapool Specification dialog box, as illustrated below:



Note the following datapool column definition highlights:

- ▶ **fName** column. The standard data type Names - First supplies this datapool column with masculine and feminine first names.
- ▶ **iName** column. The standard data type Names - Last supplies this datapool column with surnames.
- ▶ **custID** column. The standard data type Integer - Signed supplies ID numbers to this datapool column. Because all customer IDs in this example consist of seven digits, the **Minimum** and **Maximum** range is set from 1000000 through 9999999. Also, because all IDs must be unique, **Sequence** is set to Unique.

NOTE: **Sequence** can only be set to Unique for Integer - Signed data types.

- ▶ **ccNum** column. The Integer - Signed data type generates numbers up to nine digits. Because credit card numbers contain more than nine digits, the standard data type Float X.XXX is used to supply credit card numbers to this datapool column. **Decimals** is set to 0 so that only whole numbers are generated. **Sequence** is set to Random to generate random card numbers. To generate unique numbers, **Repeat** is set to 1.

- ▶ **ccType** column. This is the only datapool column that needs to have values supplied from a user-defined data type. The user-defined data type Credit Card Type contains just four values — American Express, Discover, MasterCard, and Visa.
- ▶ **ccExpDate** column. The standard data type Date - MM/DD/YYYY supplies credit card expiration dates to this datapool column. The range of valid expiration dates is set from July 1, 1998 through December 31, 2002. **Sequence** is set to Random to generate random dates.

Example of Datapool Value Generation

After you define datapool columns in the Datapool Specification dialog box, click **Generate Data** to generate the datapool values. To see the values you generated:

1. Click **Close**.
2. Click **Edit Datapool Data**.

This is what you see:



Editing Datapool Column Definitions with TestManager

To edit the definitions of the columns in an existing datapool with TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Select the datapool to edit, and then click **Edit**.
3. Click **Define Datapool Fields**.

The Datapool Specification dialog box appears. This dialog box lets you define the columns in the datapool file. Datapool column definitions are listed as rows in this dialog box. Datapool columns are also called *fields*.

4. Edit one or more datapool column definitions, as described in the table in *Defining Datapool Columns* on page 6-27.
5. When finished editing datapool column definitions, take either of these actions:
 - To save the datapool column definitions but not generate any data, click **Save**, and then click **Close**.
 - To save the datapool column definitions and generate data, type a number in the **No. of records to generate** field, and then click **Generate Data**. Optionally, click **Yes** to see a brief summary of the generated data.

NOTE: To see the generated values, close the Datapool Specification dialog box. In the Datapool Properties - Edit dialog box, click **Edit Datapool Data**.

If There Are Errors

If the datapool values are not successfully generated, you are prompted to see an error report rather than a summary of the generated data. To correct the errors:

1. Click **Yes** to see the error report.
2. After viewing the cause of the errors, click **Cancel**.
3. Correct the errors in the **Datapool Fields** grid.

Deleting a Datapool Column

Datapool column definitions are listed as rows in the Datapool Specification dialog box.

To delete a datapool column definition from the list:

1. Click anywhere in the row to be deleted.
2. Click the gray box to the left of the datapool column name. This action selects the entire row.
3. Press the DELETE key.

You are not prompted to confirm the deletion.

Editing Datapool Values with TestManager

To view or edit the values in an existing datapool with the TestManager editor:

1. Click **Tools** → **Manage Datapools**.
2. Select the datapool to edit, and then click **Edit**.
3. Click **Edit Datapool Data** in the Datapool Properties - Edit dialog box.

The Edit Datapool dialog box appears.

4. Modify the datapool values as necessary. Note that:



- When you click a value to edit it, an arrow icon appears to the left of the row you are editing.



- When you begin to edit the value, a pencil icon appears to the left of the row, indicating editing mode.
- To undo the changes you just made to a value, press CTRL + Z *before* you move the insertion point out of the field.
- To see the editing menu, select the text to edit, and then right-click the mouse.
- To increase the width of a column, move the bar that separates column names. To increase the height of a row, move the bar that separates rows:

Slide up or down to change row height.

IName	IName	custID
Karen	Conway	2445267
Koganti	Pistora	7327429

Slide left or right to change column width.

5. To delete an entire datapool row:
 - a. Click the gray box to the left of the first value in the row you are deleting. This action selects the entire row.
 - b. Press the DELETE key.
6. When finished modifying datapool values, click **Save**, and then click **Close**.

For an example of the datapool values that TestManager generates, see *Example of Datapool Value Generation* on page 6-32.

Canceling 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.

Renaming a Datapool

To rename a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to rename.
3. Click **Rename**.
4. Type the datapool's new name (40 characters maximum).
5. Click **OK**, and then click **Close**.

Copying a Datapool

To copy a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to copy.
3. Click **Copy**.
4. Type a name for the new datapool (40 characters maximum).
5. Click **OK**, and then click **Close**.

Deleting a Datapool

To delete a datapool:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the name of the datapool to delete.
3. Click **Delete**.
4. Click **Yes** to confirm the deletion, and then click **Close**.

Deleting a datapool removes the datapool .csv and .spc files plus all references to the datapool from the repository.

Importing a Datapool

Robot automatically creates and populates datapools for you. However, it is possible for you to create and populate a datapool yourself, using a tool such as Microsoft Excel. For example, you might want to export data from your database into a .csv file, and use that file as your datapool.

If you create a datapool yourself, you need to import it into the same repository as the scripts that will access it. Use TestManager to import a datapool .csv file.

To import a datapool .csv file using TestManager:

1. Click **Tools** → **Manage Datapools**.
2. Click **Import**.
3. In **Look in**, specify the directory where the datapool you created is located.
4. In **File name**, specify the name of the datapool .csv file.
5. Click **Open**. The Datapool Properties - Import dialog box appears, containing the datapool name.
6. Accept the default datapool name or type a new one (40 characters maximum).
7. In **Field Separator**, make sure the character(s) displayed are the same as the field separator used in the .csv file you are importing as a datapool.

For information about field separators in datapools, see *Datapool Structure* on page 6-45.

8. Optionally, type a description of the datapool (255 characters maximum).
9. Click **OK**, and then click **Close**.

When you import a datapool, you often have to change the names of the datapool columns to match the names of the corresponding script variables. For more information, see *Matching Datapool Columns with Script Variables* on page 6-48.

Datapool Location

When you import a datapool, TestManager copies the datapool's .csv file to the Datapool directory associated with the current repository and project.

For example, if the current repository is MyRepo, and the current project directory is MyProject, the datapool is stored in the following directory:

```
C:\MyRepo\Project\MyProject\Datapool
```

This directory also includes the datapool's specification (.spc) file. When you create and then import a .csv file, TestManager automatically creates the .spc file for you. You should never edit the .spc file directly.

NOTE: After you import a datapool, the original file you used to populate the datapool remains in the directory you specified when you saved it. The Rational Test software has no further need for this file.

Importing a Datapool from Another Project

Use the TestManager Import feature to copy a datapool you created for one project into another. When you import a datapool into a new project, the source datapool is still available to the original project.

NOTE: If the datapool you are importing includes user-defined data types, import the data types *before* you import the datapool. For information, see *Importing a User-Defined Data Type* on page 6-41.

To import a datapool into a new project:

1. Run TestManager. (By default, click **Start** → **Programs** → *Rational product name* → **Rational Test 7** → **TestManager**.)
2. In the Rational Repository Login dialog box, select the repository and project that you are importing the datapool to.
3. Click **File** → **Import Test Assets**. The Import Test Assets wizard appears.
4. Select **Datapool** in the **Asset Type** list.
5. Click the appropriate overwrite option. This option determines whether to overwrite an existing datapool with the same name as the one you are importing.
6. Click **Next**.
7. In the Login dialog box, provide the path, project name, and your login information for the repository that contains the datapool you are importing.

If you are importing between projects in the same repository, you do not need to provide a user ID and password.
8. Click **Next**.
9. Select one or more datapools to import. (To select multiple datapools, hold down the CTRL key while clicking each datapool to import.)
10. Click **Next**.
11. Click **Finish** after reading the information summarizing the import operation.

Exporting a Datapool

Use the TestManager Export feature to copy a datapool to any directory on your computer's directory structure. When you export a datapool, the original datapool remains in its repository and project.

Do not attempt to export a datapool to another Rational Test project. Instead, use the import feature to import the datapool into the new project. For more information, see *Importing a Datapool* on page 6-35.

To export a datapool to a location on your computer's directory structure:

1. In TestManager, click **Tools** → **Manage Datapools**.
2. Click the datapool to export.
3. Click **Export**.
4. In **Save In**, specify the directory where you want to copy the datapool.
5. Click **Save**.
6. Click **OK** to acknowledge that the datapool was exported to the correct location.

Managing User-Defined Data Types

Use TestManager to manage user-defined data types. You can edit data type values and data type definitions. You can also rename, copy, and delete data types.


For information about creating user-defined data types, see *Data Types* on page 6-9.

Editing User-Defined Data Type Values

If you want to add, remove or modify data type values, or if you just want to modify the optional description, edit the data type.

You can only edit user-defined data types, not standard data types.

To edit a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Select the data type to edit, and then click **Edit**.
3. Optionally, modify the data type description.
4. Click **Edit Data Type Data**. A dialog box appears with the name of the data type you are editing.
5.  The arrow icon points to the line that is currently available for editing. Edit that line, or place the insertion point in another line.

You can use the keyboard's up and down arrow keys to move the line pointer.

6. To see the editing menu, select the text to edit, and then right-click the mouse.
7. To delete a value, place the insertion point anywhere in the value, click the gray box to the left of the value, and then press the DELETE key.
8. When finished making changes, click **Save**.
9. Click **OK** in the Data Type Properties - Edit dialog box, and then close the Manage Data Types dialog box.

Editing User-Defined Data Type Definitions

Like all data types, a user-defined data type is essentially a one-column datapool. The single column contains the values that you type into the user-defined data type.

You can edit the default definition of the data type column in the Datapool Specification dialog box, just as you edit the default definition of datapool columns.

If you edit the definition of a user-defined data type, and then generate values for the data type, you overwrite any existing values for the data type.

How To Edit User-Defined Data Type Definitions

To edit the definition of a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Select the data type to edit, and then click **Edit**.
3. Click **Define Data Type Field**.

The Datapool Specification dialog box appears. Note that the **Insert Before** and **Insert After** buttons are not present. That is because a user-defined data type file can have only one column of values.

4. Define the fields in the data type. Use the table in *Defining Datapool Columns* on page 6-27 to help you.
5. When finished defining the data type, click **Generate Data**.
Optionally, click **Yes** to see a brief summary of the generated data.
6. Click **Close** in the Datapool Specification dialog box, and then click **Edit Data Type Data**.
7. Make any necessary modifications to the generated values.
8. When finished modifying values, click **Save**.
9. Click **OK** in the Data Type Properties - Edit dialog box, and then close the Manage Data Types dialog box.

Automatically Generating Values for a User-Defined Data Type

You can add values to a user-defined data type by supplying it with values from a standard data type. Doing so can reduce the typing that you need to perform when adding values to the user-defined data type.

For example, suppose you want to create a user-defined data type containing a list of valid product IDs. The valid ID numbers range from 1000001 through 1000100. However, there is a dash between the fourth and fifth digits (such as 1000-001).

Rather than type in all 100 numbers, with dashes, you can have TestManager generate the numbers and assign them to a user-defined data type. All you then have to do is edit the data type values and add the dash to each ID. The following steps guide you through the process:

1. In TestManager, click **Tools** → **Manage Data Types**.
2. Click **New**.
3. Type a name for your user-defined data type (for example, Item ID).
4. Click **OK**.
5. Click **No** when prompted to enter data type values now.
6. Click the name of your new data type, and then click **Edit**.
7. Click **Define Data Type Field**.
8. Set the following column values in the grid (or accept the defaults):
 - **Type** = Integers - Signed
 - **Sequence** = Sequential
 - **Repeat** = 1
 - **Length** = 7
 - **Interval** = 1
 - **Minimum** = 1000001
 - **Maximum** = 1000100
9. Type 100 in **No. of records to generate**.
10. Click **Generate Data**.
11. Click **No** to decline to see data generation details, and then click **Close**.
12. Click **Edit Data Type Data** in the Data Type Properties dialog box.
13. Type a dash character between the fourth and fifth characters of each value.

NOTE: You can also assign the standard data type Read From File to a user-defined data type. For information about using the Read From File data type, see *Creating a Column of Values Outside Rational Test* on page 6-48.

Importing a User-Defined Data Type

You can import a user-defined data type from one project into another. When you import a user-defined data type into a new project, the source data type is still available to the original project.

To import a user-defined data type into a new project:

1. Run TestManager. (By default, click **Start** → **Programs** → *Rational product name* → **Rational Test 7** → **TestM anager**.)
2. In the Rational Repository Login dialog box, select the repository and project that you are importing the user-defined data type to.
3. Click **File** → **Import Test Assets**. The Import Test Assets wizard appears.
4. Select **Data type** in the **Asset Type** list.
5. Click the appropriate overwrite option. This option determines whether to overwrite an existing datapool with the same name as the one you are importing.
6. Click **Next**.
7. In the Login dialog box, provide the path, project name, and your login information for the repository that contains the data type you are importing.

If you are importing between projects in the same repository, you do not need to provide a user ID and password.
8. Click **Next**.
9. Select one or more data types to import. (To select multiple data types, hold down the CTRL key while clicking each data type to import.)
10. Click **Next**.
11. Click **Finish** after reading the information summarizing the import operation.

Renaming a User-Defined Data Type

To rename a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to rename.
3. Click **Rename**.
4. Type the data type's new name (40 characters maximum).
5. Click **OK**, and then click **Close**.

Copying a User-Defined Data Type

To copy a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to copy.
3. Click **Copy**.
4. Type the name of the new data type (40 characters maximum).
5. Click **OK**, and then click **Close**.

Deleting a User-Defined Data Type

To delete a user-defined data type in TestManager:

1. Click **Tools** → **Manage Data Types**.
2. Click the name of the data type to delete.
3. Click **Delete**.
4. Click **Yes** to confirm the deletion, and then click **Close**.

Generating and Retrieving Unique Datapool Rows

Many database tests work best when each row of test data is unique. For example, if a test involves virtual users adding customer orders to a database, each new order has to be unique — in other words, at least one field in the new record has to be a “key” field containing unique data.

When you are defining datapool columns in the Datapool Specification dialog box, you specify whether a given datapool column should contain unique data. If you specify that one or more columns should contain unique data, the datapool that the Rational Test software generates is guaranteed to contain unique rows.

However, even when a datapool contains all unique rows, it is possible for duplicate rows to be supplied to a script at test runtime.

To generate and retrieve unique datapool rows, you need to perform a few simple tasks when you define the datapool.

Use the following guidelines whether the datapool is being accessed by a single script or by multiple scripts, including both virtual user and GUI scripts.

What You Can Do to Guarantee Unique Row Retrieval

To ensure that a datapool supplies only unique rows to scripts at runtime, follow these guidelines:

What to do	How to do it
Specify at least one column of unique data.	<p>In the Datapool Specification dialog box, specify that at least one datapool column should contain unique data. Unique data can be supplied through the Integers - Signed data type, through the Read From File data type, and through user-defined data types.</p> <p>With the Integers - Signed data type, take all of these actions:</p> <ul style="list-style-type: none"> ▶ Set Sequence to Unique or Sequential. ▶ Set Repeat to 1. ▶ If Sequence= Unique, set an appropriate range in Minimum and Maximum. ▶ Make sure the values of Length and No. of records to generate are appropriate for the set of numbers to generate. <p>With the Read From File data type, see <i>Generating Unique Values</i> on page 6-50 for information.</p> <p>With user-defined data types, see <i>Generating Unique Values from User-Defined Data Types</i> on page 6-12 for information.</p>
Generate enough datapool rows.	<p>Generate at least as many unique datapool rows as the number of times the datapool will be accessed during a test.</p> <p>For example, if 50 users will access a datapool during a test, and each user is set for 3 iterations each, the datapool must contain at least 150 rows.</p> <p>You specify the number of rows to generate in the No. of records to generate field of the Datapool Specification dialog box.</p>
Disable cursor wrapping.	<p>If the datapool cursor wraps after the last row in the datapool has been accessed, previously fetched rows are fetched again.</p> <p>Disable cursor wrapping in any of these ways:</p> <ul style="list-style-type: none"> ▶ When editing the <code>DATAPOOL_CONFIG</code> section of a virtual user script in the Configure Datapool in Script dialog box, set Wrap at end of file? to No. ▶ When editing a virtual user script in Robot, add <code>DP_NOWRAP</code> to the list of flags in the <code>flags</code> argument of the <code>DATAPOOL_CONFIG</code> statement or the <code>datapool_open</code> function. ▶ When editing a GUI script in Robot, set the <code>wrap</code> argument of the <code>SQADatapoolOpen</code> command to False.

(Continued)

What to do	How to do it
Use sequential or shuffle access order.	<p>With sequential or shuffle access, each datapool row is referenced in the row access order just once. When the last row is retrieved, the datapool cursor either wraps or datapool access ends.</p> <p>With random access, rows can be referenced in the access order multiple times. So, a given row can be retrieved multiple times.</p> <p>Set row access order in any of these ways:</p> <ul style="list-style-type: none"> ▶ When editing the <code>DATAPOOL_CONFIG</code> section of a virtual user script in the Configure Datapool in Script dialog box, set Access Order to Sequential or Shuffle. ▶ When editing a virtual user script in Robot, add <code>DP_SEQUENTIAL</code> or <code>DP_SHUFFLE</code> to the list of flags in the <i>Flags</i> argument of the <code>DATAPOOL_CONFIG</code> statement or the <code>datapool_open</code> function. ▶ When editing a GUI script in Robot, set the <i>sequence</i> argument of the <code>SQADatapoolOpen</code> command to <code>SQA_DP_SEQUENTIAL</code> or <code>SQA_DP_SHUFFLE</code>.
Do not rewind the cursor during a test.	If you rewind the datapool cursor during a test (through the <code>VU datapool_rewind</code> function or the <code>SQABasic SQADatapoolRewind</code> command), previously accessed rows will be fetched again.

NOTE: Rational Test can only guarantee that a datapool contains unique rows when you generate datapool data through Robot or TestManager.

Creating a Datapool Outside Rational Test

To create a datapool file and populate it with data, you can use any text editor, such as Windows Notepad, or any application, such as Microsoft Excel or Microsoft Access, that can save data in .csv format.

For example, you can create a datapool file and type in the data, row by row and value by value. Or, you can export data from your database into a .csv file that you create with a tool such as Excel.

After you create and populate a datapool, use TestManager to import the datapool into the repository. For information about importing a datapool, see *Importing a Datapool* on page 6-35.

Datapool Structure

A datapool is stored in a text file with a .csv extension. The file has these characteristics:

- ▶ Each row contains one record.
- ▶ Each record contains datapool field values delimited by a field separator. Any character can be used for the field separator. Some common field separators are:
 - Comma (,). This is typically the default in the US and the UK.
 - Semi-colon (;). This is typically the default in most other countries.
 - Colon (:).
 - Pipe (|).
 - Slash (/).

The field separator can consist of up to three single-byte ASCII characters or one multi-byte character.

NOTE: To view or change the field separator, click **Start** → **Settings** → **Control Panel**, double-click the **Regional Settings** icon, and then click the **Number** tab. **List separator** contains the separator character(s).

- ▶ Each column in a datapool file contains a list of datapool field values.
- ▶ Field values can contain spaces.
- ▶ A single value can contain a separator character if the value is enclosed in double quotes. For example, “Jones, Robert” is a single value in a record, not two.

The quotes are used only when the value is stored in the datapool file. The quotes are not part of the value that is supplied to your application.
- ▶ A single value can contain embedded strings. For example, “Jones, Robert “Bob”” is a single value in a record, not two.
- ▶ Each record ends with a line feed.
- ▶ Datapool column names are stored in a .spc file. (Robot and TestManager edit the .spc file. Never edit the .spc file directly.)
- ▶ The datapool name that is stored in the repository is the same as the root datapool file name (without the .csv extension). The maximum length of a datapool name is 40 characters.

Example Datapool

This is an example of a datapool file with three rows of data. In this example, field values are separated by commas:

```
John,Sullivan,238 Tuckerman St,Andover,MA,01810  
Peter,Hahn,512 Lewiston Rd,Malden,MA,02148  
Sally,Sutherland,8 Upper Woodland Highway,Revere,MA,02151
```

Example Using Microsoft Excel

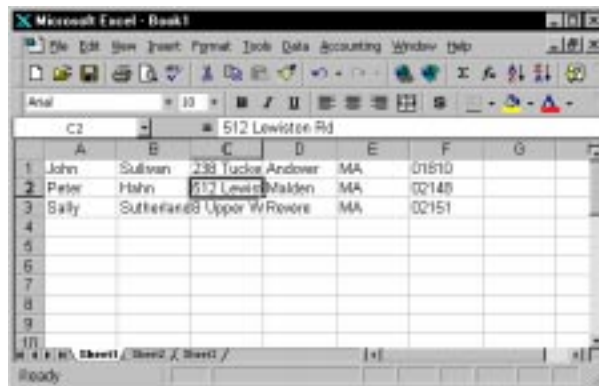
To create and populate a datapool using Microsoft Excel:

1. Run Microsoft Excel.
2. Click **File** → **New** to create a new Excel workbook.
3. Click the **Workbook** icon, and then click **OK**.
4. Type a datapool record into row 1. To do so, type each value in the record into separate columns, beginning with column A.

When using Microsoft Excel to populate a datapool, do not separate values with the Windows separator character (see page 6-45). Excel automatically inserts the separator character when you save the datapool in .csv format.

5. Continue populating the datapool by typing records into the subsequent rows.

Here is an example of how a datapool might look as it is being populated with data in Microsoft Excel:



Note that:

- ▶ Each column represents a datapool field.
- ▶ Each row is an individual datapool record containing datapool field values.

Saving the Datapool in Excel

When you finish adding rows of values to the datapool, save the datapool to .csv format. To do so using Microsoft Excel:

1. Click **File** → **Save As**.
2. In the **Save in** field, specify the directory where you want to save the datapool.

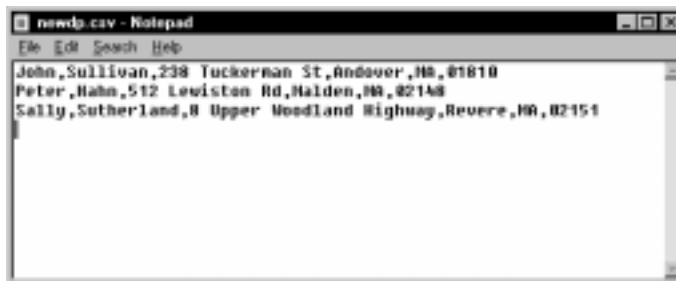
Do not specify the Datapool directory in the repository. When you later import the datapool using the TestManager Import feature, TestManager automatically copies the datapool to the Datapool directory in the current repository and project.

3. In the **File name** field, type the datapool name (maximum 40 characters). This is the name that the script and the repository use to identify the datapool.
4. In the **Save as type** list, select the entry **CSV (Comma delimited) (*.csv)**.
5. Click **Save**.

If you are prompted that the .csv file format does not support multiple workbook sheets, click **OK** to save the datapool and abandon the other (empty) worksheets.

Upon closing Excel, if you are prompted to decide whether to save the .csv file in Microsoft Excel format, click **No**.

If you use Windows Notepad to open the datapool file you just created and saved, this is how it looks:



Matching Datapool Columns with Script Variables

When you create a .csv file and then import it as a datapool, TestManager automatically assigns column names (that is, datapool field names) to each datapool column.

Datapool column names must match the names of the script variables that they supply with data (including a case match). But most likely, when you create and import a datapool, the column names that TestManager assigns will not match the names of the associated script variables. As a result, you need to edit the column names that TestManager automatically assigns during the import. You do so by modifying a column's **Name** value in the Datapool Specification dialog box.

For information about how to open the Datapool Specification dialog box during datapool editing, see *Editing Datapool Column Definitions with TestManager* on page 6-32.

NOTE: With GUI scripts, you can associate datapool column names and script variables through column position rather than column name. For more information, see the description of the `SQADatapoolValue` command in the *SQABasic Language Reference*.

Maximum Number of Imported Columns

You can import a datapool that contains up to 32,768 columns. If you open an imported datapool in the Datapool Specification dialog box, you can view and edit all datapool column definitions up to that limit.

A datapool is subject to a 150-column limit only if you generate data for the datapool from the Datapool Specification dialog box.

Creating a Column of Values Outside Rational Test

A datapool that you create with Rational Test can include a column of values supplied by an ASCII text file. You could use this feature, for example, if you want the datapool to include a column of values from a database.

Populating a datapool column with values from an external file requires two basic steps:

1. Create the file containing the values.
2. Assign the values in the file to a datapool column through the standard data type Read From File.

Step 1. Create the File

To use a file as a source of values for a datapool column, the file must be a standard ASCII text file. The file must contain a single column of values, with each value terminated by a carriage return.

You can create this text file any way you like — for example, you can use either of these methods:

- ▶ Type the list of values in Microsoft Notepad.
- ▶ Export a column of values from a database to a text file.

Step 2. Assign the File's Values to the Datapool Column

Once the file of values exists, you assign the values to a datapool column just as you assign any set of values to a datapool column — through a data type. In this case, you assign the values through the Read From File data type. To do so:

1. Open the Datapool Specification dialog box:
 - To open this dialog box during datapool creation, see *Creating a Datapool with TestManager* on page 6-25.
 - To open this dialog box during datapool editing, see *Editing Datapool Column Definitions with TestManager* on page 6-32.
2. In the **Type** column, select the data type Read From File for the datapool column being supplied the values from the external text file.
3. Tab out of the column. The Open dialog box appears.
4. In **Look in**, specify the directory where the text file you created is located.
5. In **File name**, specify the name of the text file.
6. Click **Open**.

You can use the Read From File data type to assign values to multiple columns in the same datapool.

Generating Unique Values

You can use the Read From File data type to generate unique values to a datapool column you create outside Rational Test.

To generate unique values through the Read From File data type, the file that the data type accesses must contain unique values.

In addition, when you are defining the datapool in the Datapool Specification dialog box, make the following settings for the datapool column associated with the Read From File data type:

- ▶ Set **Sequence** to Sequential.
- ▶ Set **Repeat** to 1.
- ▶ Make sure the **No. of records to generate** value does not exceed the number of unique values you are accessing through the Read From File data type.

For information about the values you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 6-27.

▶▶▶ C H A P T E R 7

Designing Schedules

This chapter describes how to design schedules. It includes the following topics:

- ▶ About schedules
- ▶ Creating a schedule
- ▶ Inserting user groups into a schedule
- ▶ Inserting scripts into a schedule
- ▶ Inserting other items into a schedule
- ▶ Opening a schedule
- ▶ Editing a script
- ▶ Editing a schedule
- ▶ Using events and to coordinate execution
- ▶ Setting shared variables
- ▶ Saving a schedule
- ▶ Printing and exporting a schedule
- ▶ Checking a schedule
- ▶ Checking Agent computers
- ▶ Controlling runtime information of a schedule
- ▶ Controlling how a schedule terminates
- ▶ Running a schedule

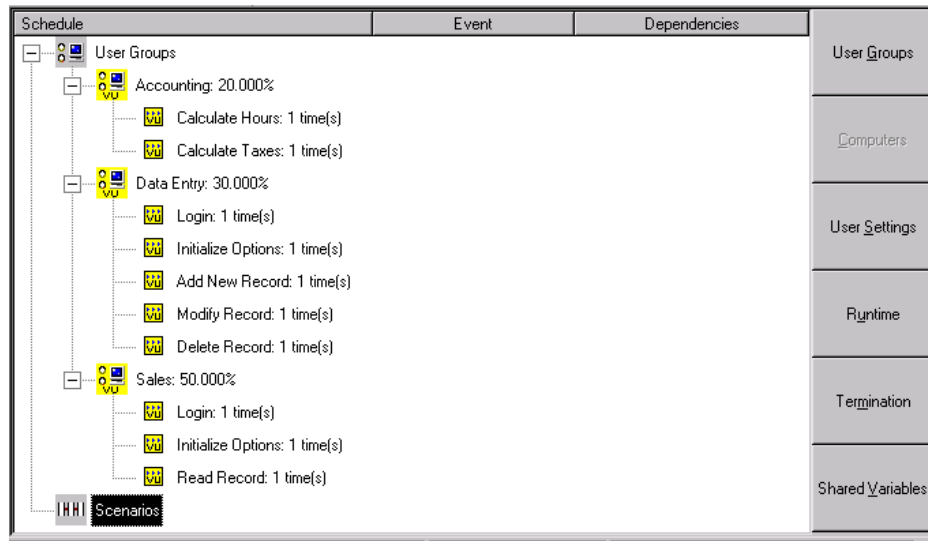
About Schedules

A schedule shows a hierarchical representation of the workload that you want to run. It shows such items as the user groups, the number of users in each user group, which scripts the user groups run, and how many times each script runs.

Through a schedule, you can:

- ▶ Run virtual user scripts and GUI scripts.
- ▶ Group scripts to emulate the actions of different types of users.
- ▶ Set the order in which scripts run.
- ▶ Synchronize users.

The following simple schedule shows three user groups: Accounting, Data Entry, and Sales.



In this schedule:

- ▶ The Accounting user group runs two scripts: one calculates payroll hours and one calculates payroll taxes.
- ▶ The Data Entry user group runs five scripts: one logs in, one initializes database options, and three change database records.
- ▶ The Sales user group runs three scripts: one logs in, one initializes database options, and one reads database records.

The examples in this chapter show virtual user scripts. A schedule, however, can contain GUI scripts, or a mixture of GUI and virtual user scripts.

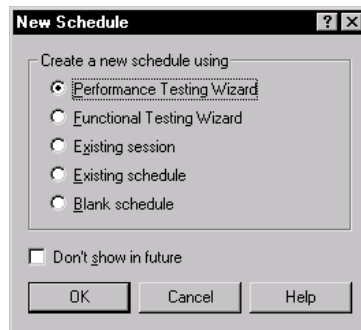
Creating a Schedule

A **schedule** enables you to not only run scripts, but, more importantly, to emulate the actions of users adding load on a system. A schedule can be as simple as one virtual user executing one script, or as complex as hundreds of virtual users in different groups, with each group executing different scripts at different times.

Creating a Schedule from a Blank Schedule

To create a schedule from a blank schedule:

1. Click **File** → **New** → **Schedule**.



2. Click **Blank Schedule**.
3. Click **OK**.

A schedule appears, which contains a User Groups icon and a Scenarios icon.



The following sections explain how to insert user groups, scripts, and other items into a schedule so you can run it.

Creating a Schedule from a Session

If you have recorded a session in Robot, you can play back the scripts in the session through LoadTest.

When you add a session to a schedule and then run the schedule, you execute all of the client/server requests that you recorded during the session in the order in which you recorded them.

Adding a session to a schedule saves you from having to add individual scripts to the schedule. For example, suppose you record the scripts *Connect*, *Query*, and *Disconnect* in a recording session named *DBQuery*. To run the scripts in a schedule, you can either add each script to the schedule in the order in which you recorded them, or you can simply add the session *DBQuery*.

To create a schedule directly from a Robot session:

1. Click **File** → **New** → **Schedule**. The New Schedule dialog box appears.
2. Select **Existing Session**, and click **OK**.
3. LoadTest displays a list of sessions that you have recorded. Click the name of a session, and click **OK**.

LoadTest automatically creates a schedule that is ready to run.

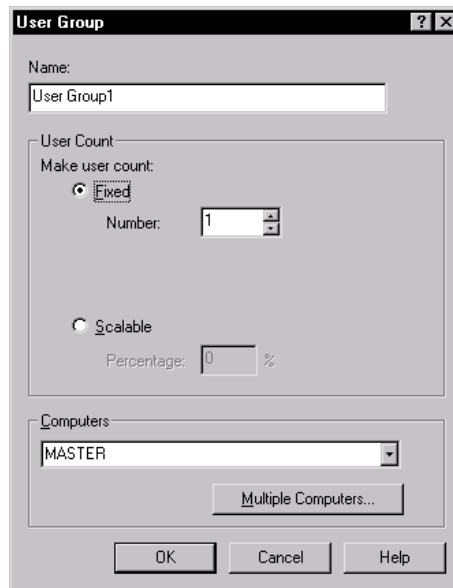
NOTE: You can also create a schedule using a wizard. For more information, see *Schedules, creating*, in the LoadTest Help index.

Inserting User Groups into a Schedule

A **user group** is the basic building block for all schedules. A user group is a collection of users that perform the same activity. For example, the schedule on page 7-2 contains three user groups: Accounting, Data Entry, and Sales.

To insert a user group into a schedule:

1. Click **Insert** → **User Group**.



The screenshot shows a dialog box titled "User Group" with a standard Windows window border (minimize, maximize, close buttons). The dialog is divided into three sections:

- Name:** A text input field containing "User Group1".
- User Count:** A section titled "Make user count:" with two radio button options:
 - Fixed:** Selected. Below it is a "Number:" label and a spin box containing the value "1".
 - Scalable:** Unselected. Below it is a "Percentage:" label and a spin box containing the value "0" followed by a percent sign.
- Computers:** A dropdown menu showing "MASTER". Below the dropdown is a button labeled "Multiple Computers...".

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

2. Type the name of the user group in the **Name** box. Although you can use the default name, it is better to use a name that describes the user group.

NOTE: The name of a user group cannot be identical to the name of a shared variable, a script, or the following reserved words: MASTER, ALL, ASSIGN, TO, THRU, END, UNION, DELAY, delay, shared, SHARED, SYC, DLB_FREQ, DLB_TIME, or PERMUTE.

3. Decide whether the group will contain a fixed or scalable number of users.
 - **Fixed** – Specifies a fixed number of users. Enter the maximum number of users you want to be able to run. For example, if you enter **50** users, you can run up to 50 users in the Sales group each time you run a schedule.

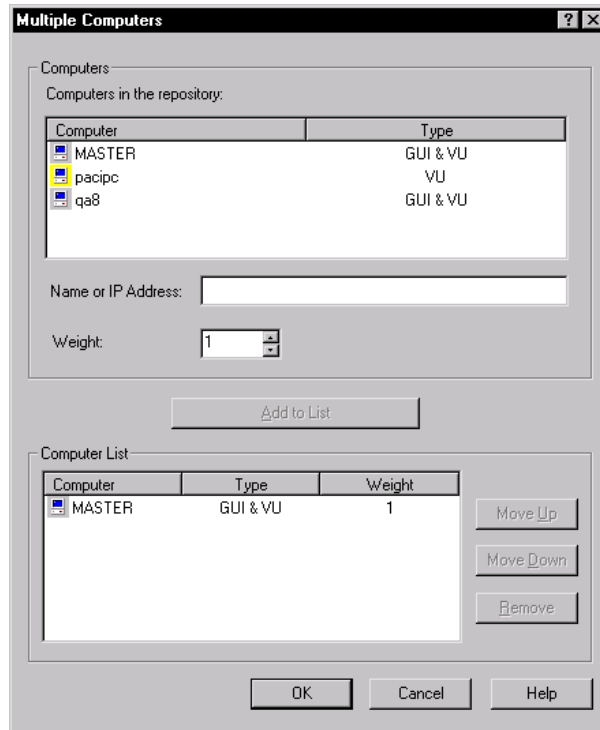
Typically, you assign a fixed number of users to user groups that do not add a workload. For example, one user could run a `warmup` script to open a database for the users, and another user could run a `shutdown` script to restore and close the database.
 - **Scalable** – Specifies a scalable number of users. Type the percentage of the workload that the user group represents. For example, the Accounting group might represent 20 percent of the users, the Data Entry group might represent 30 percent of the users, and the Sales group might represent 50 percent of the users. Each time you run a schedule, you specify the total number of users that will run.
4. Select the computer where the user group will run. The default computer is the LoadTest Master. To run a user group on an Agent computer, select its name in the **Computers** box.

Typically, you run the user group on an Agent computer if:

- A functional test is designed for a particular computer.
- A performance test requires specific client libraries, or a functional test requires specific software. The user group must run on the computer that has the libraries or software installed.

NOTE: Copy any custom-created external C libraries necessary for the test to the Agent computer.

5. To distribute the users in a user group among multiple computers, click the **Multiple computers** button.



Typically, you run a user group on multiple computers if you have:

- A functional test that must execute as quickly as possible. You can save time by running your users simultaneously on different computers.
- A large number of virtual users, and the Master computer does not have enough CPU or memory resources to support this workload. You can conserve resources by running the users on different computers so that fewer users run on each computer.

The top section of the Multiple Computers dialog box displays the computers that are in the repository.

To run a user group on a computer that is not in the repository, simply type its name or IP address in the **Name or IP Address** box, and then click **Add to List**. LoadTest adds the computer to the list, but not to the repository.

If you assign more than one computer to a user group, the users are distributed according to the **Weight** that you set for the computer. If one computer can handle more users, you should give it a higher weight than the other computers. For example, if AgentOne can handle twice as many users as the other computers, you may want to give it a weight of 2, and assign the others a weight of 1.

If you do not assign a weight, the users are distributed evenly in round-robin fashion. The first user runs on the first computer on the list, the second user runs on the second computer, and so on.

Click the **Move Up** and **Move Down** buttons to change the order in which the computers become available to run users. If you have many computers and many users, the order of the computers is unimportant. However, if you are running only a few users, you might want one user to start on a specific computer. For example, if your schedule contained only one user, the user would run on the top computer in the list.

Click the **Delete** button to remove a computer from the list. When you are satisfied with the order of the computers, click **OK**.

6. In the User Group dialog box, click **OK**. The user group appears in the schedule.

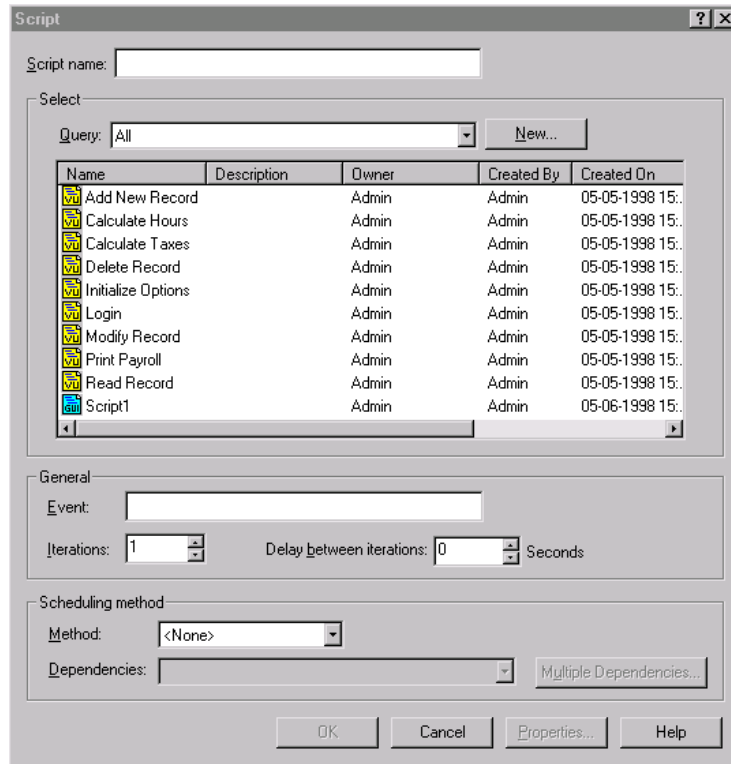
Inserting Scripts into a Schedule

After you insert user groups into a schedule, you add the scripts that the user groups run. The schedule on page 7-2 shows the scripts that each user group runs. The Accounting group runs two scripts, the Data Entry group runs five scripts, and the Sales group runs three scripts.

NOTE: User groups can contain either virtual user or GUI scripts. You cannot mix two kinds of scripts within one user group.

To insert a script into a schedule:

1. Open the schedule, and select the user group that will run the script. For example, the Sales user group might run a script that reads a record.
2. Click **Insert** → **Script**. The Script dialog box appears, which lists the scripts in the repository:



3. Select the script from the **Select** section of the window, or type the name of the script in the **Script name** box.

NOTE: If you type the name of a script that is not in the repository, LoadTest adds the name to the repository. However, the script file itself is empty, and you must record it before you run the schedule.

4. To make the script set an event, type the name of the event in the **Event** box. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
5. To run the script more than once, set the number of times you want to repeat it in the **Iterations** box.
6. If you set multiple iterations of the script and you want to delay between each execution of the script, enter the delay time in the **Delay between iterations** box.

NOTE: Another way to set iterations and delays between iterations is to insert a sequential selector, a delay, and a script directly into a schedule. The advantage of this method is that the selector and the delay are visible in the schedule. The advantage of adding iterations and a delay between iterations when you insert a script is that less space is taken up in the schedule.

7. Typically, a script runs immediately after the preceding item in the schedule completes. To customize when the script runs, select one of the following options from the **Scheduling method** list:
 - **After Delay of...**
 - **After Dependencies...**
 - **After Start of Schedule**
8. To create a dependency on another item in the schedule, type the name of the event in the **Dependencies** box. For information about dependencies, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
9. Click **OK**.

Inserting Other Items into a Schedule

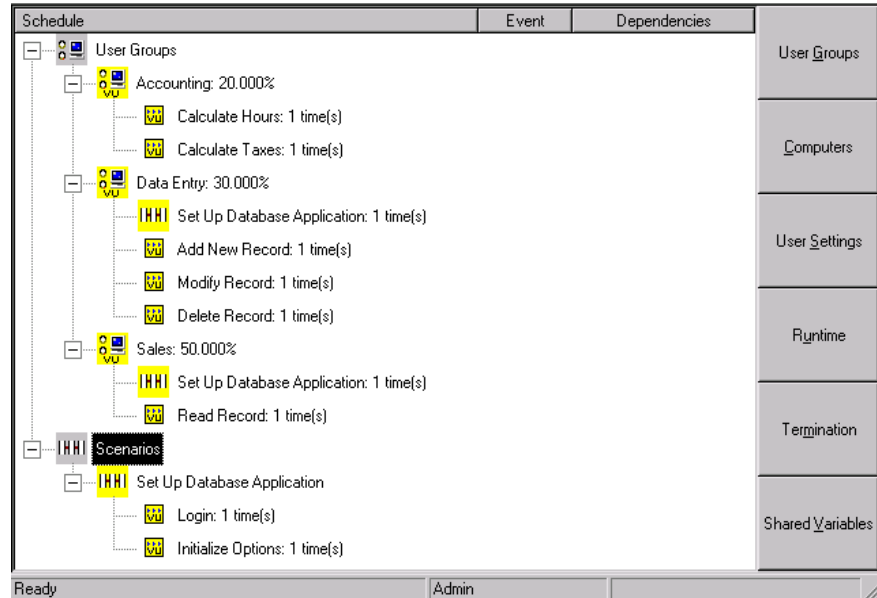
A schedule requires only user groups and scripts to run. However, a schedule that realistically models the work that actual users perform is likely to contain more than user groups and scripts. It might also contain scenarios, selectors, delays, and synchronization points.

Inserting a Scenario

A **scenario** lets you group scripts together so they can be shared by more than one user group. If you have a complicated schedule that uses many scripts, grouping the scripts under a scenario has the added advantage of making your schedule easier to read and maintain.

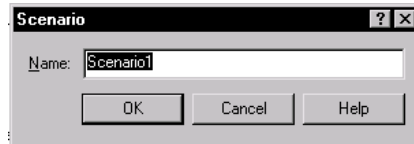
You define a scenario in the **Scenarios** section of the schedule by inserting a scenario and then inserting items within it. To make a user group execute a scenario, you insert the scenario name in a user group. Otherwise, the scenario is not executed.

In the schedule on page 7-2, both the Data Entry and the Sales user groups run the scripts `Login` and `Initialize Options`. You can simplify this schedule by storing both scripts in a scenario. The following schedule shows the scripts `Login` and `Initialize Options` grouped under the `Set Up Database Application` scenario:



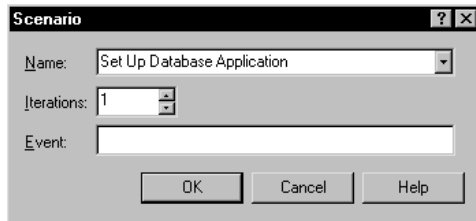
To insert a scenario into a schedule:

1. Click the **Scenarios** section in the schedule.
2. Click **Insert** → **Scenario**.



3. Type a name in the **Name** box. You can use the default name, but it is better to supply a name that describes the scenario.

4. Click **OK**. The name of the scenario appears in the **Scenarios** section of the schedule.
5. Click the user group that will run the scenario.
6. Click **Insert** → **Scenario**. The Scenario dialog box appears:



7. Type the name of the scenario in the **Name** box.
8. To run the scenario more than once, set the number of times you want to repeat it in the **Iterations** box.
9. To make the scenario set an event, type the name of the event in the **Event** box. For details about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.

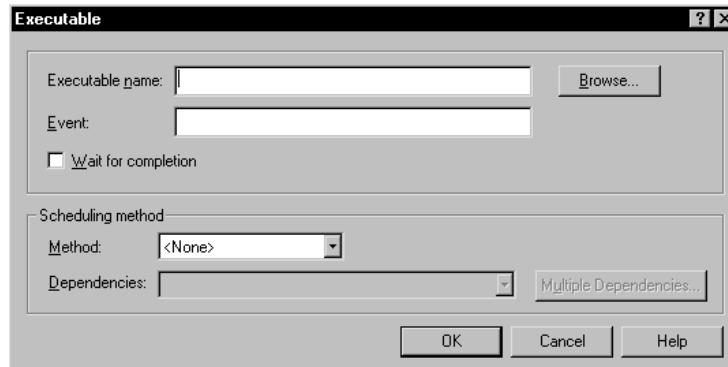
After you have created the scenario and the user group that runs the scenario, it is a good idea to populate the scenario. A scenario requires only scripts to run. However, like a user group, a realistic scenario may also contain selectors, delays, and synchronization points. A scenario can even contain other scenarios.

Inserting an Executable

An **executable** is a program, such as Notepad or Excel, that runs on Windows NT. You can insert an executable into a GUI user group only. When you run the schedule, the executable runs as well.

To insert an executable into a schedule:

1. Click **Insert** → **Executable**.

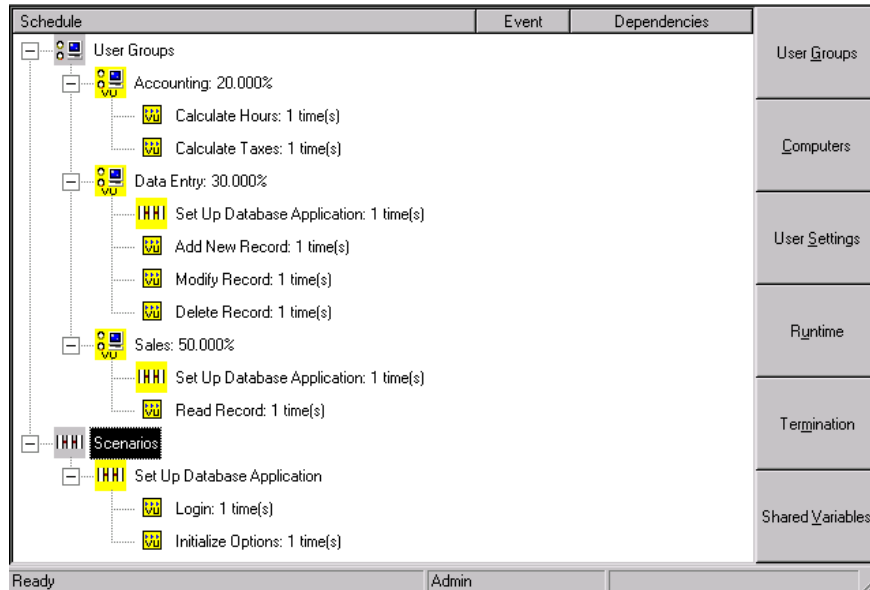


2. Type a name in the **Executable name** box. To see a list of executables, click the **Browse** button.
3. To make the executable set an event, type the name of the event in the **Event** box. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
4. Select the **Wait for completion** check box to if you want the executable to finish running before the next item in the schedule begins running. Clear this box if you want the schedule to run the next item before the executable finishes running.
5. Normally, an executable runs immediately after the preceding item in the schedule completes. To customize when the executable runs, select an option from the **Scheduling method** list:
 - **After Delay of...**
 - **After Dependencies...**
 - **After Start of Schedule**
6. To create a dependency on another item in the schedule, type the name of the dependency in the **Dependencies** box. For information about dependencies, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
7. Click **OK**.

Setting Schedule Items to Run in Different Sequences

A **selector** provides more sophisticated control than running a simple sequence of consecutive items in a schedule. A selector tells LoadTest which items each user will execute, and in what sequence. For example, you might want to repeatedly select a script at random from a group of scripts.

Consider the following schedule, which does not contain any selectors:

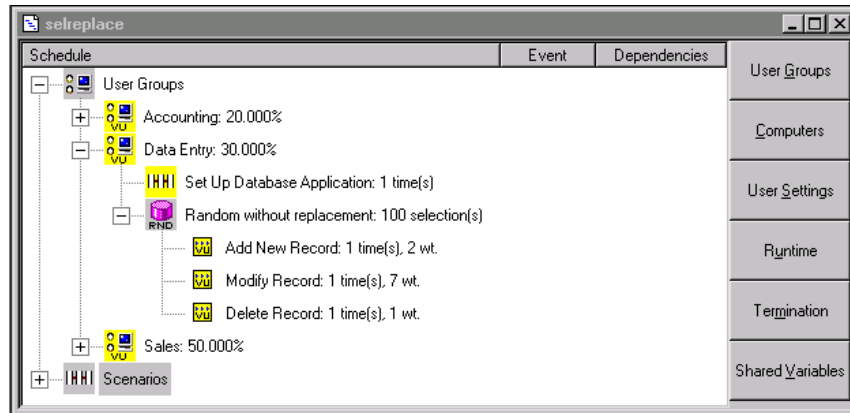


When you run the schedule with 50 virtual users, LoadTest assigns 10 users to Accounting, 15 users to Data Entry, and 25 users to Sales. All 50 users start executing scripts at the same time.

- ▶ The 10 Accounting users run each script in the order in which the script appears in the schedule: first `Calculate Hours` and then `Calculate Taxes`.
- ▶ The 15 Data Entry users run the `Set Up Database Application` scenario and then run the `Add New Record`, `Modify Record`, and `Delete Record` scripts in the order in which the scripts appear in the schedule.
- ▶ The 25 Sales users run the `Set Up Database Application` scenario and then run the `Read Record` script.

However, suppose your Data Entry users actually add records, delete records, and modify records randomly. Furthermore, they do not perform these tasks with the same frequency. For every record they delete, they modify seven records and add two records.

To make your user group reflect this behavior, insert a Random selector into the Data Entry user group. The following schedule shows the Data Entry user group, which selects scripts randomly without replacement.



When you run the schedule with 50 virtual users, LoadTest assigns 10 users to Accounting, 15 users to Data Entry, and 25 users to Sales. Each Data Entry user:

- ▶ Runs the Set Up Database Application scenario.
- ▶ Picks one script per iteration: Add New Record, Modify Record, or Delete Record. Since there are 100 iterations, each Data Entry user adds a record 20 times, modifies a record 70 times, and deletes a record 10 times. The adding, modifying, and deleting are done in any order.

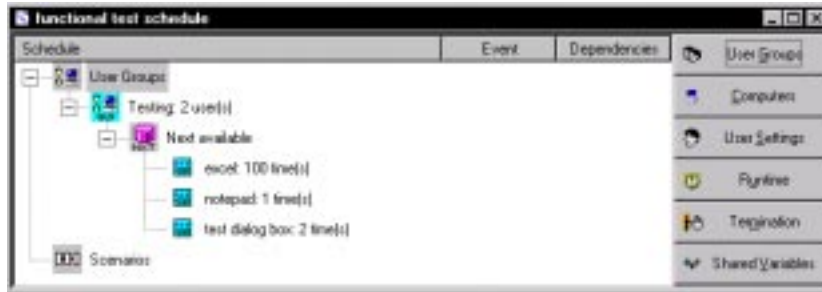
Types of Selectors

LoadTest provides the following types of selectors:

- ▶ **Sequential** – Runs each script or scenario in the order in which it appears in the schedule. This is the default.
- ▶ **Next available** – Distributes its scripts or scenarios to an available user (one user per computer). Typically, you use this selector in a GUI user group. The items are parceled out in order, based on which users are available to run another script. Once an item runs, it does not run again.

A next available selector distributes each script without regard to its iterations.

In the following schedule, the Testing user group has two users (and therefore two computers) assigned to it. The schedule runs three GUI scripts under a next available selector.



When you run this schedule:

- The `excel` script runs on the first computer for 100 iterations.
 - The `notepad` script runs on the second computer for one iteration.
 - The `test dialog box` script runs on the next available computer (most likely the second computer, because its script had only one iteration), and runs for two iterations.
- **Random with replacement** – The selector runs the items under it in random order, and each time an item is selected, the odds of it being selected again remain the same. Think, for example, of a bucket that contains 10 red balls and 10 green balls. You have a 50% chance of picking a red ball and a 50% chance of picking a green ball. The first ball selected is red. The ball is then replaced in the bucket. Every time you pick a ball, you have a 50% chance of getting a red ball.
- Since the ball is replaced after each selection, the bucket always contains 10 red and 10 green balls. It is even possible (but unlikely) that you pick a red ball every time—you never pick a green ball. Similarly, the **Random with replacement** selector is not guaranteed to run every item in it, particularly if you have set one script to run more frequently than another. In other words, if your bucket contains 19 red balls and one green ball, the green ball might not be selected at all.
- **Random without replacement** – The selector runs the items under it in random order, but each time an item is selected, the odds change. For example, think of the same bucket that contains 10 red balls and 10 green balls. Again, the first ball selected is red. However, the ball is *not* replaced in the bucket. Therefore, the next time you have a slightly greater chance of picking a green ball. Each time you select a ball, your odds change.

And, of course, if the first 10 balls selected are red, the odds of the next 10 balls being green are 100 percent. Similarly, the **Random without replacement** selector will run every item in it, as long as the number of iterations of the selector is greater than or equal to the number of items in the selector.

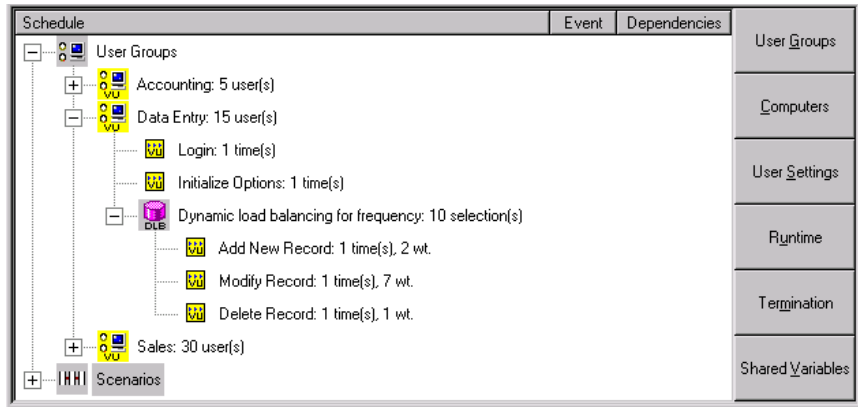
- ▶ **Dynamic load balancing** – With dynamic load balancing, items are not selected randomly. Think again of the bucket that contains red and green balls, and assume that you have assigned an equal “weight” to each ball. Therefore, if the first ball selected is red, the second ball selected is always green. This is because with each ball, or script, selected, the system “dynamically balances” the workload to approach the 50-50 weight that you set. Of course, you can set other weights that are not 50-50. The important fact to remember, however, is that the next script to run is not selected randomly; it is selected to balance the workload according to the weight that you have set.

You can balance the workload either for time or for frequency. For example, assume you are dynamically balancing `scripta` and `scriptb`, and using equal weights. `Scripta`, however, takes twice as long to run as `scriptb`.

If you check **Dynamic load balancing for time**, the load is balanced by the runtime of each script. Because `scripta` takes twice as long to run, it is actually selected only half as often as `scriptb`.

If you check **Dynamic load balancing for frequency**, both scripts will run an equal number of times. If `scripta` runs 500 times, `scriptb` also runs 500 times. The fact that `scripta` takes longer to run is not factored into the balance.

Dynamic load balancing is done across all users in a user group. For example, the following schedule shows the Data Entry user group, which contains 15 users. Three scripts, `Add New Record`, `Modify Record`, and `Delete Record`, are contained in a dynamic load balancing selector.



When you run the schedule, the first Data Entry user selects the `Modify Record` script, because it has the largest weight. But because the workload is balanced across *all* Data Entry users, after the first user exits, LoadTest recalculates the weights to reflect the fact that the script with the largest weight, 7, has already been selected. By the time later users are ready to select a script, the weights have changed so they have a greater chance of selecting the `Add New Record` script.

Inserting a Selector

To insert a selector into a schedule:

1. Click the user group or a scenario that will contain the selector. For example, to add a selector to the scripts in a user group, click that user group.
2. Click **Insert** → **Selector**.



3. Select the type of selector you want to add.
4. The Selector dialog box differs slightly, depending on the type of selector that you add and whether it is within another selector.
 - For a sequential selector, enter the number of times you want the selector to repeat in the **Number to repeat** box.
 - For a random or dynamic selector, type the number of iterations in the **Number to Select** box.

If you are inserting a selector *within* another random or dynamic load balancing selector, enter the weight of the selector in the **Weight** box.

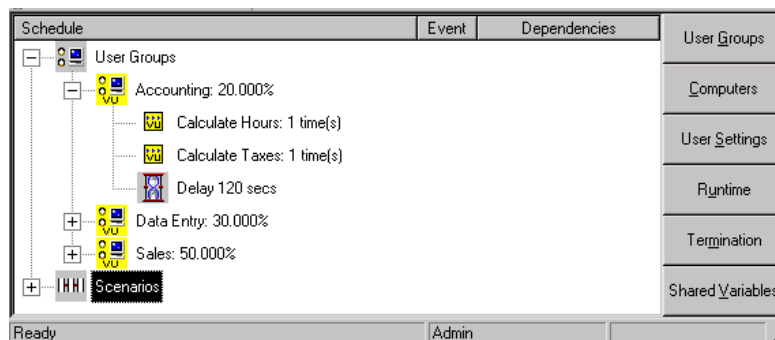
5. To make the selector set an event, type the event in the **Event** box. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
6. Click **OK**. LoadTest displays the selector in your schedule.

Inserting a Delay

A **delay** tells LoadTest how long to pause before it runs the next item in the schedule.

In functional testing, you use delays to cause scripts to wait before executing. For example, if one user updates a record, you can insert a delay to give the application-under-test time to display the correct information. By providing a delay, you can ensure that the application-under-test has enough time to complete displaying the information, in case another user must work with the information displayed.

In performance testing, you use delays to model user behavior. For example, if your Accounting user group calculates the hours and taxes, and then pauses for two minutes, you would add a delay after the `Calculate Taxes` script, as shown in the following schedule.



You can insert a delay into a schedule or a script.

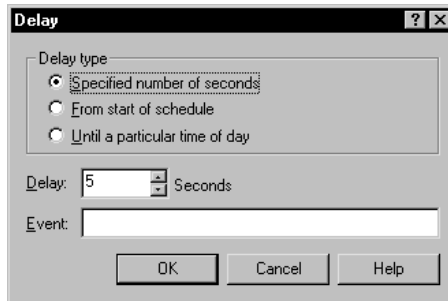
- ▶ **Into a schedule** – The advantages of inserting a delay into a schedule are that the delay is visible in the schedule and the delay is easy to change without editing the script.
- ▶ **Into a script** – The method that you use to insert a delay into a script depends on whether the script is a virtual user script or a GUI script.

You insert a delay into a virtual user script by editing the script to include a `delay()` library routine or by modifying the think time VU environment variables. Use this method to make the delay before script execution different each time. For more information about the `delay()` library routine and the think time environment variables, see the *VU Language Reference*.

You insert a delay into a GUI script when you record the script. For more information, see the *Using Rational Robot* manual.

To insert a delay into a schedule:

1. Click the user group, scenario, or selector that you want to add a delay to.
2. Click **Insert** → **Delay**.



3. Select a delay:
 - **Specified number of seconds** – Begin counting the delay when the previous item finishes executing. Enter the number of seconds to delay in the **Delay** box.
 - **From start of schedule** – Begin counting the delay from the time the schedule begins executing. Enter the number of seconds to delay in the **Delay** box.
 - **Until a particular time of day** – Delay until the time of day you specify. If you select this, the **Delay** box changes to a time-of-day spin box. Set the time of day in that box.

4. Enter how long to delay in the **Delay** box.
5. To make the delay set an event, type the event in the **Event** box. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
6. Click **OK**.

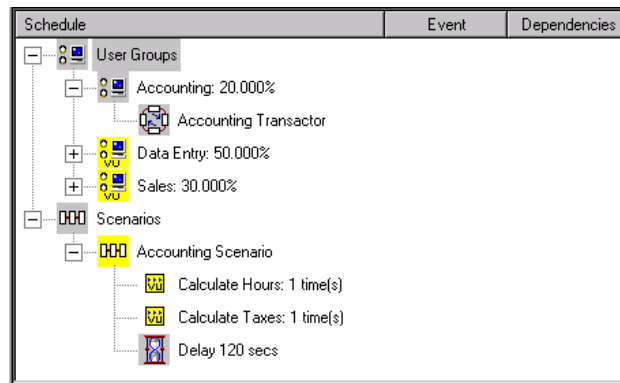
NOTE: The default delay is 5 seconds. To change this default, click **Tools** → **Options**, click the **Create Schedule** tab, and enter a delay in the **Delay Options** section.

Setting Schedule Items to Run at Certain Rates

A **transaction** tells LoadTest the number of user-defined tasks that each user will run in a given time period. For example, you might be testing an Order Entry group that completes 10 forms per hour. Or you might be testing a Web server, and you want the server to be able to support 100 hits per minute. To model this time-based behavior, you use a transaction.

In the previous section, you added a delay to the Accounting user group. This delay made the virtual users pause for two minutes after they calculated the hours and taxes, as shown in the schedule on page 7-19.

However, suppose that the Accounting group instead calculates the hours and the taxes at a certain rate—say, 10 calculations per hour. To make your schedule reflect this rate, you need to replace the selector and delay with a transaction. The following schedule shows the Accounting user group after you have added a transaction:



This schedule is identical to the one on page 7-19, except that it contains:

- ▶ A transactor, which tells LoadTest the rate that you want to maintain, and how long you want to maintain this rate.
- ▶ A scenario, which contains the items that the transactor will run.

The following section explains how to insert a transactor and set its options to reflect the rate that you want to maintain.

Inserting a Transactor

To insert a transactor into a schedule:

1. Click the user group or selector that will contain the transactor.

NOTE: This example shows how to insert a transactor in a user group. However, you can also insert an independent transactor in a sequential or random selector, which will affect the number of times LoadTest runs the transactor. For information about selectors, see *Types of Selectors* on page 7-15.

2. Click **Insert** → **Transactor**.

The screenshot shows the 'Transactor' dialog box with the following settings:

- Name: (empty dropdown)
- Type: Coordinated, Independent
- Rate: Total rate, User rate. Value: 1 per minute.
- Distribution: Negative Exponential
- Range: 20 % of interval
- Iterations: 1
- Scenario: (empty dropdown)
- Event: (empty text field)

3. Select the name of the transactor that you want to add. The default name is **Transactor*n***, but it is a good idea to make the transactor name similar to the scenario that it will run.
4. Under **Type**, select whether you want a coordinated or an independent transactor.
 - A **Coordinated** transactor, which has a built-in synchronization point, lets you specify the total rate that you want to achieve. The virtual users work together to generate the workload. For example, if you run a schedule with 10 users and then run the same schedule with 20 users, the total transaction rate will stay the same.

Use a coordinated transactor when you are emulating the total transaction rate applied to a server, rather than the rate of specific times a user runs a task. For example, to emulate the number of hits per minute that a Web server can handle, use a coordinated transactor.
 - An **Independent** transactor lets each user operate independently. It does not coordinate the users under it with a built-in synchronization point. For example, if you run a schedule with 10 users and then run the same schedule with 20 users, the total transaction rate will double—because the number of users have doubled.

Use an independent transactor if different user groups will run the transaction at different times, or you are emulating user behavior rather than group behavior. For example, to emulate an Accounting user group that performs 10 calculations per hour but not all at the same time, use an independent transactor.
5. Under **Rate**, select the rate of the transactor.
 - For a coordinated transactor, you generally select **Total rate**. This is because whether 100 users or 50 users are participating, it has no effect on the rate that LoadTest submits transactions.

Select **User rate** for a coordinated transactor, however, if you expect to change the rate frequently and want the convenience of not having to edit the schedule. For example, suppose you have inserted a coordinated transactor, and you want to compare a workload at 100 hits per minute, 200 hits per minute, and 300 hits per minute—increasing the workload with each schedule run. If you select **User rate**, you do not have to change the rate in the transactor's properties. Instead, when you run the schedule, you can run the schedule at 100 users, 200 users, and 300 users, and the rate will scale proportionally.

- For an independent transactor, you must select **User rate**. Because each transactor operates independently, **User rate** is meaningful only with this type of transactor.
- 6. Enter the **Distribution**. Although each distribution will approach the rate you have chosen (if you run the transactions over a long enough period of time), the negative exponential distribution most closely emulates typical user behavior.
 - A **Constant** distribution means that each transaction occurs exactly at the rate you specify. For example, if your rate is four per minute, a transaction will start at 15 seconds, 30 seconds, 45 seconds, and 60 seconds—exactly four per minute, evenly spaced, with a 15-second interval. Although this distribution is simple conceptually, it does not accurately emulate the randomness of user behavior.

Use a Constant distribution to emulate an automated process. For example, you might want to emulate an environment where users are uploading data to a database every half hour.

- A **Uniform** distribution means that over time, the transactions will average out to the rate you specify, although the time between each transaction will not be constant. The time between the start of each transaction will be chosen randomly with a uniform distribution within the range that you select. Think of this range as a “window” through which the transaction can run.

For example, assume that your rate is 4 per minute (that is, 1 transaction per 15-second interval). If you select a range of 20%, your transaction will have a 3-second window on each side, because 20% of 15 seconds is 3 seconds.

Therefore, the first transaction will start at 12 - 18 seconds (15 plus or minus 3). The second transaction will start 15 seconds plus or minus 3 seconds after the first transaction starts. So, for example, if the first transaction started at 12 seconds, the second transaction would start at 24 to 30 seconds. However, if the first transaction started at 18 seconds, the second transaction would start at 30 to 36 seconds.

Because each transaction starts *randomly* within the range that you specify, it is normal for transactions to run at a rate that is faster or slower than the rate that you selected for short periods of time. For example, if a transaction started every 12 seconds for a minute (recall that the window is 12-18 seconds), the rate for that interval would be 5 per minute—not the 4 per minute that you selected. Over time, however, the transaction rate will average out to 4 per minute.

With a Uniform distribution, a transaction has the same probability of running within the range that you specify. The transaction will start anywhere within this window. In our example, the probability of the first transaction starting at 12 seconds, 18 seconds—or anywhere in between—is equal.

- A **Negative Exponential** distribution, in contrast, changes the *probability* of when a transaction will start. This distribution most closely emulates the “burstiness” of typical user behavior. Using the same example of 4 transactions per minute, the probability that a transaction will start immediately is high, but decreases over time, so that the desired average rate is maintained.

Imagine that you have called a meeting at two o’clock. Most people arrive at two, a few people arrive at five minutes past two, and fewer still at ten past two. Perhaps the last straggler arrives at two-thirty. This arrival time approximates a negative exponential distribution. Most people arrive on time, and then the arrival rate will decline. Mathematically speaking, the interval is chosen randomly from a negative exponential distribution with the average interval = $1/\text{rate}$.

7. At **Iterations**, enter the number of times that you want the transaction to run. For example, you might want the transaction to run five times.
8. Enter the name of the **Scenario** that the transactor will run. It is a good idea to make the scenario name similar to the name of the transactor.
9. To make the transactor set an event, type the event in the **Event** box. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 7-52.
10. If you are inserting an independent transactor *within* a random selector, enter the weight of the selector in the **Weight** box.
11. Click **OK**. LoadTest displays the transactor in your schedule.

Inserting a Synchronization Point

A **synchronization point** lets you coordinate the activities of a number of users by pausing the execution of each user at a particular point (the synchronization point) until one of the following events occurs:

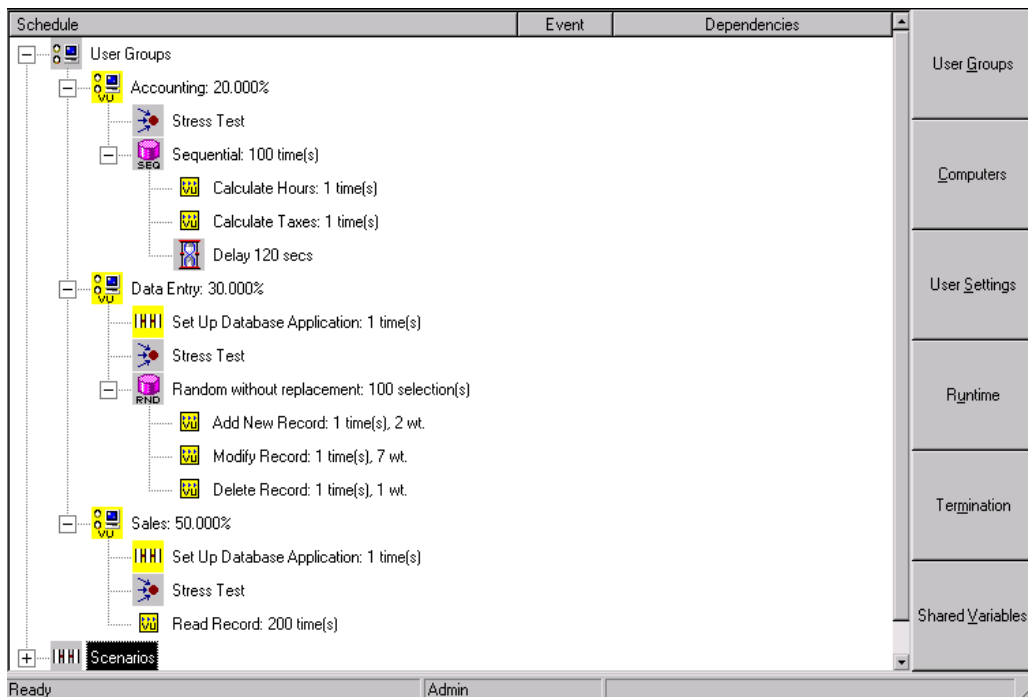
- ▶ All users associated with the synchronization point arrive at the synchronization point.
- ▶ A timeout period is reached before all users arrive at the synchronization point.
- ▶ You manually release the users while monitoring a schedule.

Designing Schedules

When one user encounters a synchronization point, the user stops and waits for other users to arrive. When the set number of users reach the synchronization point, LoadTest releases the users and allows them to continue executing.

For example, assume that you are running a stress test, which is an attempt to run your applications under extreme conditions to see if they or the server “break.” Your schedule might contain virtual users that perform the certain operations continuously and repeatedly for hours on end. To run a stress test, you need to synchronize your virtual users so that they perform the operations at the same time to stress the system. You insert a synchronization point to synchronize these virtual users.

The following schedule shows a stress test:



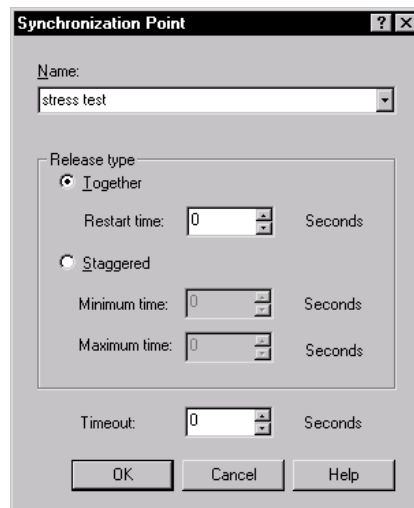
The users in the Accounting user group immediately wait at the synchronization point. The users in the Data Entry and Sales user groups perform the Set Up Database Application scenario and then wait at the synchronization point. When all the users reach the synchronization point, they are released.

If you run the test with 10,000 virtual users, when all the users reach the stress test synchronization point, they are released:

- ▶ Each of the 2000 users in the Accounting group calculates the hours and taxes, pauses for two minutes, and then calculates the hours and taxes again. Each user repeats this 100 times.
- ▶ Each of the 3000 users in the Data Entry group adds, deletes, or modifies a record. Each user repeats this 100 times.
- ▶ Each of the 5000 users in the Sales group reads a record. Each user repeats this 200 times.

To insert a synchronization point into a schedule:

1. Click **Insert** → **Synchronization Point**.



2. Type the name of the synchronization point in the **Name** box.
3. To release all users at once from a synchronization point, click **Together**. The default restart time is 0, which means that when the last user reaches the synchronization point, all users are released immediately.

To delay the users, enter a number in the **Restart time** box. For example, if you set the **Restart time** to 4 seconds, after the users all reach the synchronization point (or the timeout occurs), they wait 4 seconds, and then they are all released.

4. To release the users one by one from a synchronization point, click **Staggered**.

The amount of time that each user waits to be released is chosen at random and is uniformly distributed within the range that you set in the **Minimum time** and **Maximum time** boxes. For example, if the **Minimum time** is 1 second and the **Maximum time** is 4 seconds, after the users reach the synchronization point (or the timeout occurs) each user waits between 1 and 4 seconds before being released. All users are distributed randomly between 1 and 4 seconds.

5. In the **Timeout** box, enter the timeout period for this synchronization point. If all the users associated with a synchronization point do not reach the synchronization point when the timeout period ends, LoadTest releases any users waiting there. The timeout period begins when the first user arrives at the synchronization point.

Although a user who reaches a synchronization point after a timeout is not *held*, the user is *delayed* at that synchronization point. So, for example, if the timeout period is reached, and the **Restart time** is 1 second and the **Maximum time** is 4 seconds, a user is delayed between 1 and 4 seconds.

The default **Timeout** is 0, which means that there is no timeout. Setting a timeout is useful, because one virtual user might encounter a problem, and might never reach the synchronization point. You do not want to hold up other virtual users because of a problem with one user.

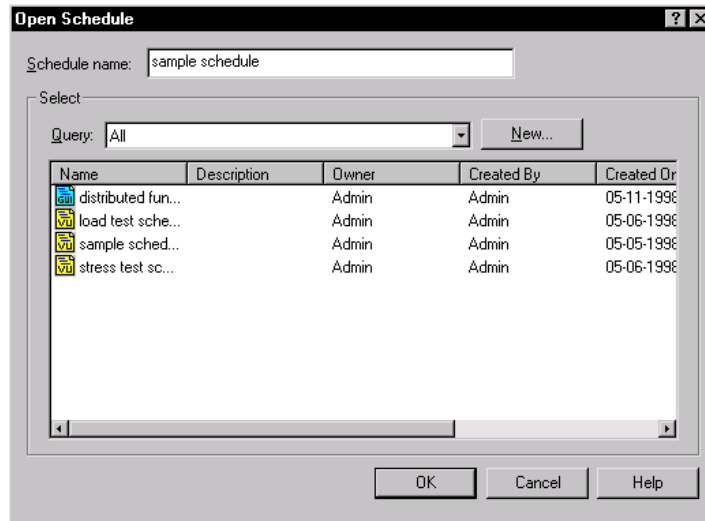
6. Click **OK**.

Opening a Schedule

To work with an existing schedule, you must open it. You can open a schedule from a menu or from the Asset Browser.

To open a schedule from a menu:

1. Click **File** → **Open** → **Schedule**. The Open Schedule dialog box appears, which lists your schedules.



2. Select a schedule.
3. Click **OK**.

To open a schedule using the Asset Browser:

1. Click **View** → **Asset Browser**.
2. Under **Schedules**, double-click the schedule, or select **Open** from its shortcut menu.

Editing a Script

While you are working with schedule, you may want to want to edit a script. Through LoadTest, you can:

- ▶ Edit the properties of a script.
- ▶ Edit the text of a script.

Editing the Properties of a Script

A script can have properties associated with it in addition to the script name. Examples of script properties include a description of the script and the purpose of 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 edit the script's properties after you record the script.

To edit the properties of a script:

1. Click the script whose properties you want to edit.
2. Click **Edit** → **Script Properties**.

3. Click the tab that you want to edit.

NOTE: The **Related Assets** tab pertains to GUI scripts, and the **VU Compilation** tab pertains to VU scripts.

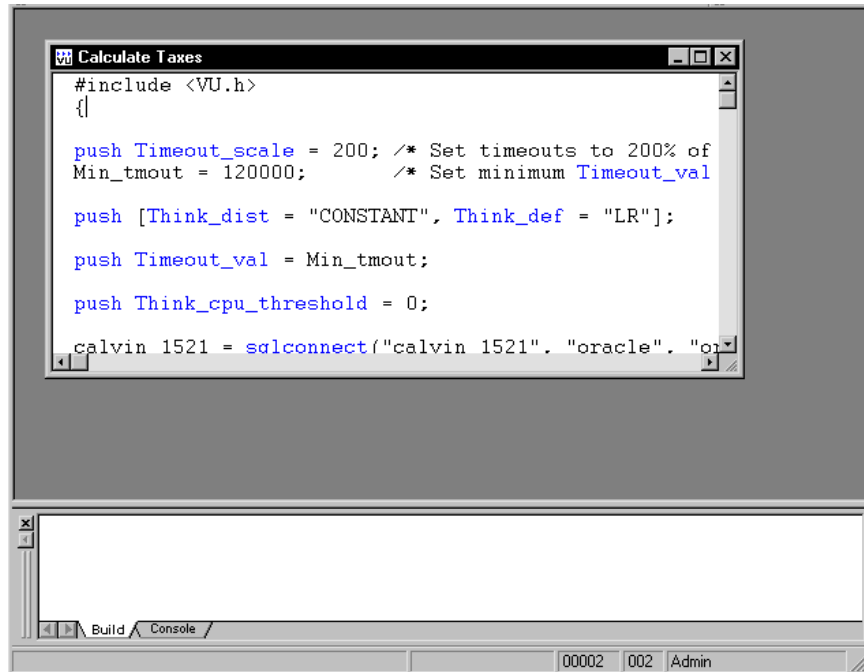
4. Edit the script's properties, and then click **OK**.

For information about the properties you can define, see the section on customizing scripts and LoadTest schedules in the *Using Rational Robot* manual.

Editing the Text of a Script

To edit the text of a script:

1. Click the script whose text you want to edit.
2. Click **Edit** → **Open Script**. Robot appears, ready to edit the script you have selected:



3. To edit the script, use Robot's **Edit** menu commands.
4. When you have finished editing the script, click **File** → **Close**.

Editing a Schedule

While you are working with a schedule, you might want to rearrange certain items. For example, you might move a script from one user group into another user group or gather a few scripts together into a scenario. You might also want to change the properties of an item. For example, you might change a user group from Fixed to Scalable, or change the number of users in a user group. The following sections tell you how to do these tasks.

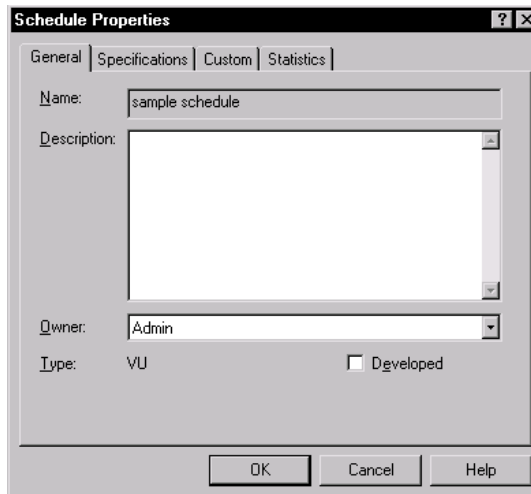
Editing the Properties of a Schedule

A schedule can have properties associated with it. Examples of schedule properties include a description of the schedule and the owner of the schedule.

Defining schedule properties is an important part of the test planning process. For that reason, you typically define a schedule's properties in TestManager before you design the actual schedule. But you can also edit the schedule's properties within LoadTest.

To edit the properties of a schedule:

1. Click **File** → **Open** → **Schedule**, and open the schedule whose properties you want to edit.
2. Click **File** → **Properties**.



3. Click the tab that you want to edit.
4. Edit the schedule's properties, and then click **OK**.

For information about the properties you can define, see the section on customizing scripts and LoadTest schedules in the *Using Rational Robot* manual.

Cutting and Pasting Items

To cut and paste an item in a schedule:

1. Click the item. To select more than one item, hold down the CTRL key while clicking.
2. Click **Edit** → **Cut**.
3. Click where you want to paste the item. You can paste the item in the same schedule or in another schedule.
4. Click **Edit** → **Paste**.

LoadTest does not let you paste any item that would create an incorrect schedule. For example, if you copy the Accountants user group and then paste it into another portion of the schedule, LoadTest renames the pasted group Accountants1. This is because you cannot have two user groups with the same name in one schedule.

Similarly, if you copy a GUI and virtual user script and then paste both scripts into a virtual user group, LoadTest pastes only the virtual user script. This is because a user group cannot mix GUI and virtual user scripts.

Deleting Items

To delete an item in a schedule:

1. Click the item. To select more than one item, hold down the CTRL key while clicking.
2. Click **Edit** → **Delete**.

All items, except for scripts and executables, are completely deleted. Scripts are deleted from the schedule but remain in the repository, so you can reuse them. Executables are deleted from the schedule but remain on your system.

For information about deleting scripts, see the *Using Rational Robot* manual.

Replacing Items

You can use in-line editing to replace any item in a schedule except delays and selectors. Replacing an item—especially an item high in the schedule structure—is often easier than deleting the item and adding another one. For example, your schedule may contain a complex structure of user groups, scripts, and scenarios. Rather than having to delete an item and recreate the schedule structure underneath, you can replace the item.

To replace an item in a schedule:

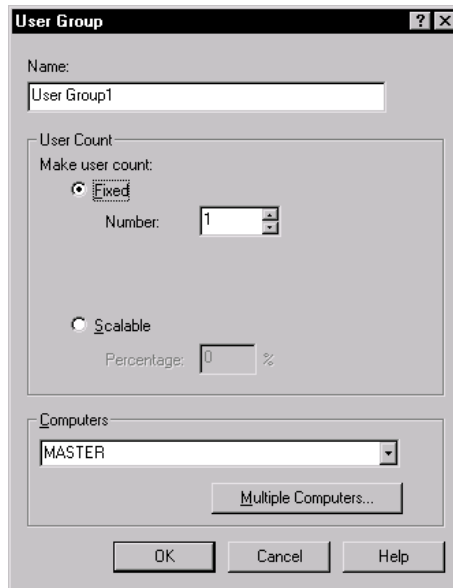
1. Click **Tools** → **Options** → **Create Schedule**, and clear the **Show numeric values** check box. You must turn this option off to replace an item.
2. Open the schedule that contains the item you want to replace.
3. Click the item, wait a few seconds, and click it again.
4. Type the name of the new item. If the item is a script, it must already be in the repository.

Editing Items

To edit the properties of an individual item in a schedule, select that item and click **Edit** → **Properties**. LoadTest displays the same dialog box that appeared when you created the item. You can edit the values in each box.

For example, to edit the properties of a user group:

1. Click the user group.
2. Click **Edit** → **Properties**.



3. To change the name of the user group, type a new name in the **Name** box.

4. To change whether the user count is fixed or scalable, click the appropriate choice under **Make user count**.
5. To change the maximum number of users emulated by the user group, or to change the percentage of the load the user group should emulate, select a different number in the **Number** or **Percentage** box.
6. To change which computer this user group is runs on, select a new computer from the **Computers** list.
7. If you are running the schedule on multiple computers and want to change the order of computers, click the **Multiple computers** button. The Multiple Computers dialog box appears.

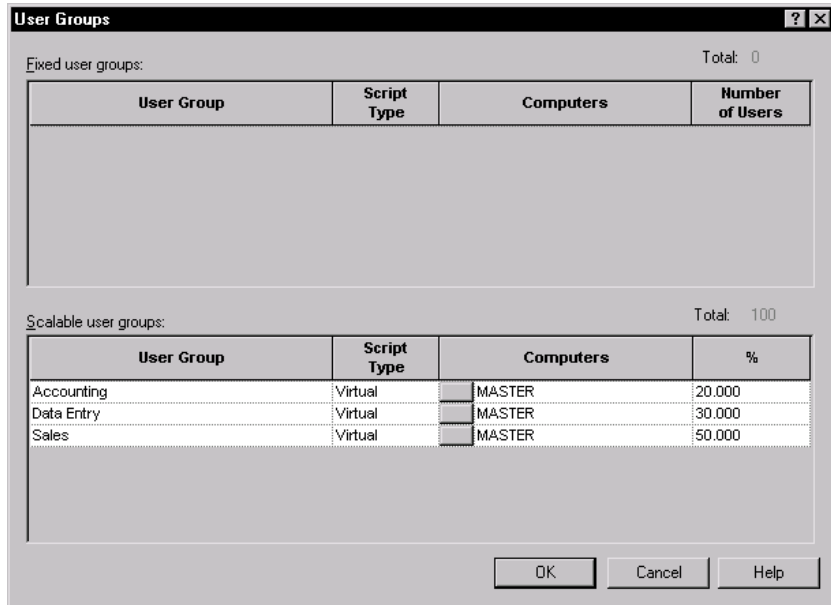
Click the **Move Up** and **Move Down** buttons to change the order of the computers in the list. Click the **Delete** button to remove a computer from the list. When you are satisfied with the order of the computers, click **OK**.
8. In the User Group dialog box, click **OK**.

Editing Information for All User Groups

At times, you may want to edit information for more than one user group. For example, you might want to change the scaling proportion of the user groups. Although you can edit each user group individually, it is much easier to edit the information for all of the user groups at the same time.

To edit information for all user groups:

1. Click the **User Groups** button to the right of the schedule.



The columns of the dialog box display the same information that you set when you created each user group.

2. To change which computer the user group runs on, click the button in the **Computers** column. The Multiple Computers dialog box appears, which lets you change the information.
3. To change the number of users assigned to a fixed user group, type the number in the **Number of Users** column.
4. To change the percentage of users assigned to a scalable user group, type the number in the % column.
5. Click **OK**.

Editing the Settings of an Agent Computer

You may want to change the default settings associated with an Agent computer. To change the computer settings:

1. Click the **Computers** button to the right of the schedule. The Computer Settings dialog box appears, which lists the Agent computers in the schedule and the operating systems that these computers run on:



If a computer has not been added to the repository, the **Operating System** column displays the message `Not in the repository`.

2. The **Update VU Scripts** column controls whether LoadTest copies compiled VU scripts to the Agent computer. Select this check box for LoadTest to copy over compiled scripts if they are out of date. Clear the box only if you are sure the compiled scripts are up to date and you want to avoid the overhead of checking each script.
3. Click the arrow next to the **Return Files** column, and check the types of files you want copied to the Master computer after the schedule runs. You can return Output, Result, Error, and Log files to the Master computer.
4. Click the arrow next to the **Remove Files** column, and check the types of files you want removed from the Agent computer after the schedule runs. You can remove Output, Result, Error, and Log files from the Agent computer.

NOTE: If you select both the **Return Files** and the **Remove Files** boxes for a file, the file is first copied to the Master computer, and then removed from the Agent computer.

5. In the **Local Directory** box, type the name of the local working directory for the Agent computer to store compiled scripts and any datapool files. The Output, Result, Error, and Log files are also stored in a subdirectory under this directory. It is a good idea to supply a name to ensure that there is adequate space to store these files and to avoid conflicts with other users who running tests on that Agent.

If you do not supply a name, LoadTest uses the environment settings of the user who started the Agent computer to determine where to store scripts and files. LoadTest checks the following environment variables, and uses the value of the first one that is set:

- a. TMPDIR
- b. TMP
- c. TEMP
- d. (UNIX Agents only) /tmp

NOTE: For GUI scripts, only Result and Error files have meaning, and thus they are the only files returned to the Master computer, removed from an Agent computer, or stored in a local directory.

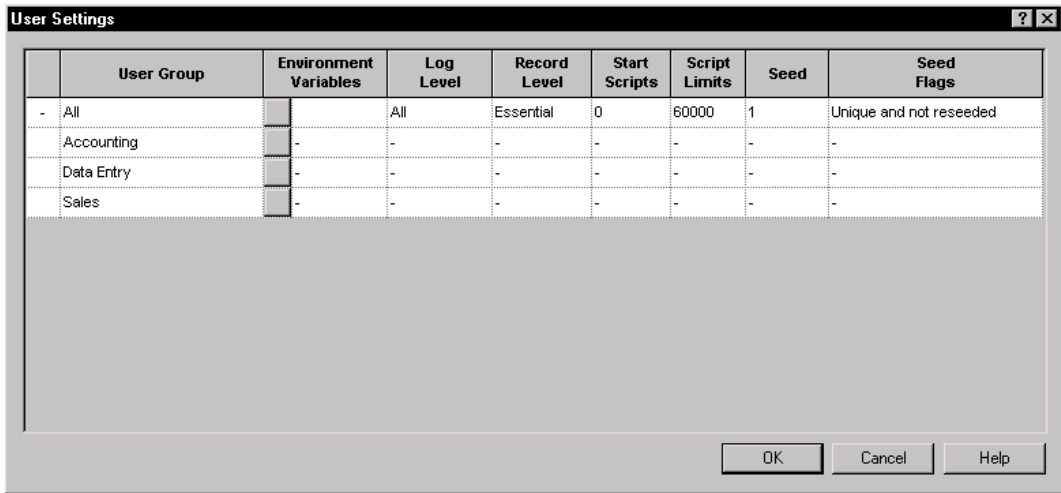
6. Click **OK**.

Editing the User Settings

You may want to change the default settings associated with users. In particular, it is often useful to change what information is logged when you run a schedule. For example, if you are having problems running a schedule, you may want to set one virtual user's log level to **All** and the other users' log levels to **Error** so that you can investigate the problem more thoroughly.

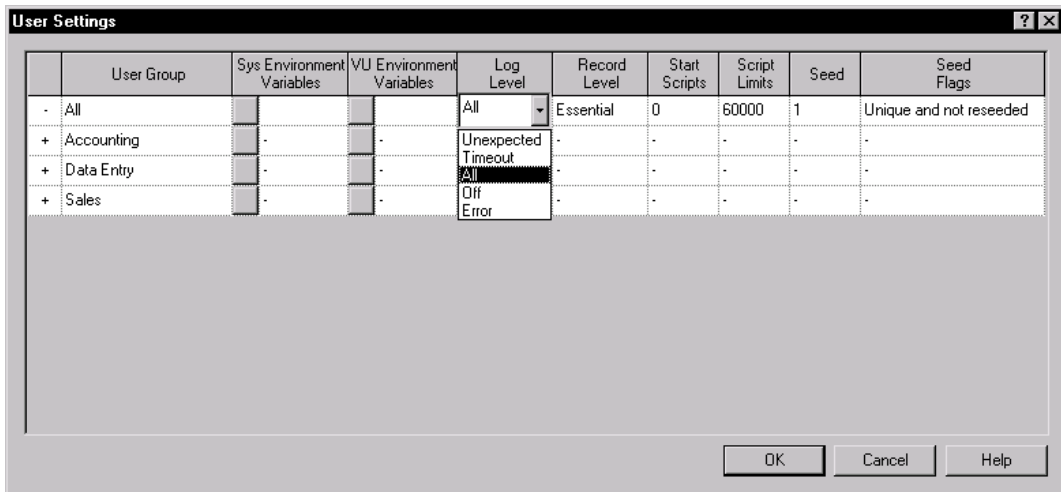
To edit user settings:

1. Click the **User Settings** button to the right of the schedule.



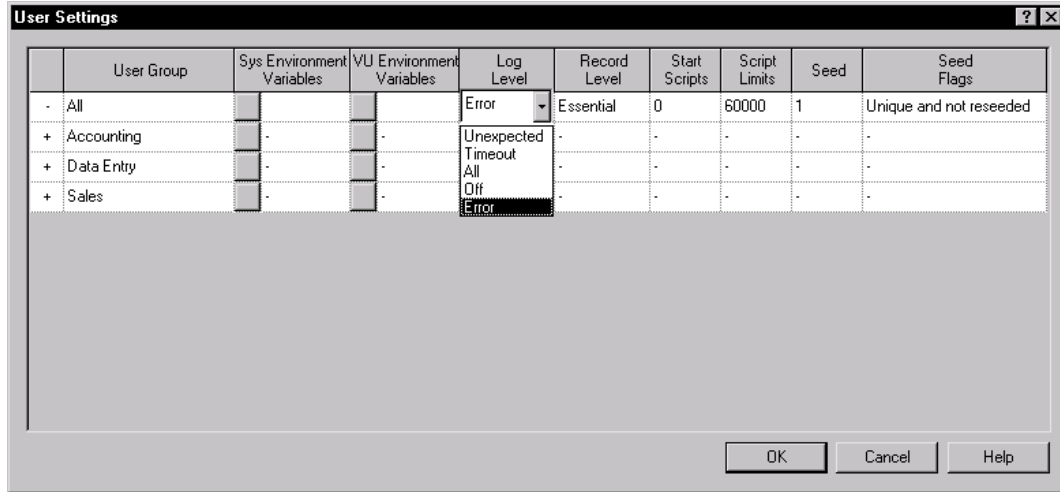
In this dialog box, all of the user groups have inherited the same values, as you can see by the minus sign in each column.

To change the value of a user setting for all user groups, double-click a value in the first row. For example, to change the log level of all virtual user groups, double-click **All** in the **Log Level** column and change it to the value you want:

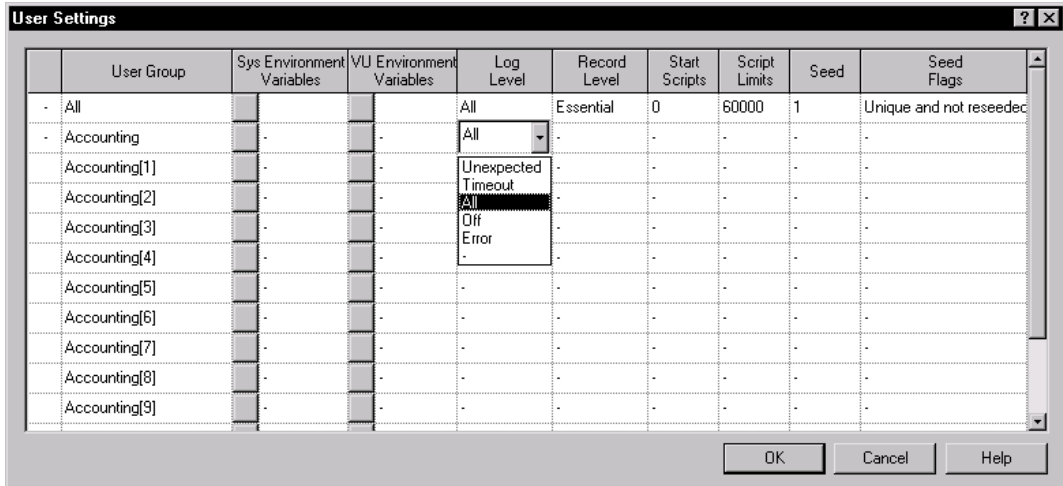


Designing Schedules

To change the value of a user setting for one virtual user group, double-click the individual value that you want to change. For example, to change the log level of the Accounting user group to **Error**, double-click the minus sign in the Accounting row of the **Log Level** column, and then select **Error**:



If a virtual user group has a fixed number of users, you can change the value of a user setting for an individual user. To do so, double-click the plus sign in the first column of the user group that contains that user. For example, assume that the Accounting user group is fixed, with 20 users. To change the log level of the first user in the Accounting user group to **All**, double-click the minus sign in the first column of the Accounting row, and select **All**:



Assigning Values to System Environment Variables

LoadTest passes the system environment variables set on an Agent computer to each user. If you are using virtual users to test a database server or application, you can override these system environment variables as follows:

If you are testing	LoadTest passes these environment variables to the Agent
Oracle on a UNIX Agent	<p>The directory that contains the client software to ORACLE_HOME.</p> <p>Example:</p> <pre>ORACLE_HOME = /ora/app/oracle/product/8.0.5</pre> <p>If /var/opt/oracle does not contain tnsnames.ora, assign the pathname of the file to the variable TNS_ADMIN.</p> <p>Example: TNS_ADMIN = /home/uname/oracletest</p>
Sybase on a UNIX Agent	<ul style="list-style-type: none"> ▶ The directory that contains the client software to SYBASE. Example: SYBASE = /usr/local/sybasec ▶ The directory that contains the Sybase client libraries to the path of one of the following system environment variables: PATH (Windows NT) LD_LIBRARY_PATH (Solaris Agents) SHLIB_PATH (HP-UX Agents) LIBPATH (AIX Agents)

(Continued)

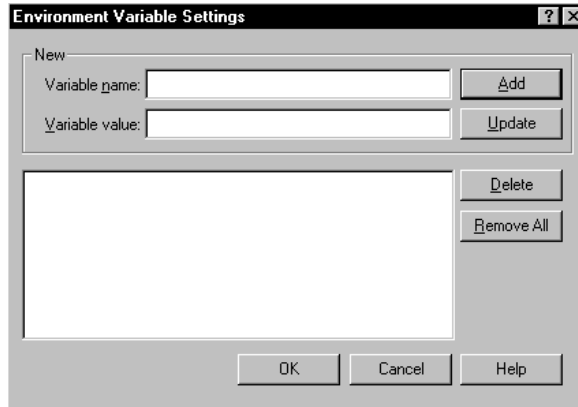
If you are testing	LoadTest passes these environment variables to the Agent
<p>Master or Agents running TUXEDO scripts</p>	<ul style="list-style-type: none"> ▶ The directory that contains the client software to TUXDIR. ▶ The NLSPATH environment variable to the path of the directory that contains the TUXEDO message file. ▶ The value of \$TUXDIR/lib to one of the following system environment variables: LD_LIBRARY_PATH (Solaris Agents) SHLIB_PATH (HP-UX Agents) LIBPATH (AIX Agents) For Windows NT Masters, these need to be defined only for TUXEDO client-only installations. The TUXEDO full run-time installation process sets them automatically. For more information, see the TUXEDO installation instructions. ▶ One of the following: The Workstation Listener's address to WSNADDR. Example: WSNADDR=//sparky:36001 WSNADDR=00028CA1C0A8F0D6 The Workstation Listener's host name and port to WSLHOST and WSLPORT. These variables override WSNADDR, if set. Example: WSLHOST=sparky WSLPORT=36001
<p>Master or Agents running TUXEDO scripts that use FML typed buffer field names</p>	<ul style="list-style-type: none"> ▶ A list of FML field table file names to FIELDTBLS. This variable is used by Agents running scripts that contain FML typed buffer field name references. If not set, VU functions that use FML typed buffer field names that are defined in files not in this list fail, causing dependent VU commands to fail. Example: FIELDTBLS=ct.fl dtbl , inv . fl dtbl ▶ The absolute pathname of the directory containing the FML field table file to FLDTBLDIR. This variable is used by Agents running scripts that contain FML typed buffer field name references. If not set, VU functions that use FML typed buffer field names (for example, tux_setbuf_int () fail, causing dependent VU commands to fail. Example: FLDTBLDIR=/u1/tuxapp/dat

(Continued)

If you are testing	LoadTest passes these environment variables to the Agent
Master or Agents running TUXEDO scripts that use FML32 typed buffer field names	<ul style="list-style-type: none"> ▶ A list of FML32 field table file names to FIELDTBLS32. This variable is used by Agents that run scripts that contain FML32 typed buffer field name references. If not set, VU functions that use FML32 typed buffer field names that are defined in files not in this list fail, causing dependent VU commands to fail. Example: FIELDTBLS32=ct32.fltdtbl,inv32.fltdtbl ▶ The absolute pathname of the directory containing the FML32 field table files to FLDTBLDIR32. This variable is used by Agents running scripts that contain FML32 typed buffer field name references. If not set, VU functions that use FML32 typed buffer field names (such as tux_setbuf_int()) fail, causing dependent VU commands to fail. Example: FLDTBLDIR32=/u1/tuxapp/dat
Master or Agents running TUXEDO scripts that use VIEW, X_COMMON, or X_C_TYPE typed buffers	<ul style="list-style-type: none"> ▶ A list of view description file names to VIEWFILES. This variable is used by used by Agents running scripts that use VIEW, X_COMMON or X_C_TYPE typed buffers. If not set, VU functions that use these typed buffers that are defined in view description files not in this list fail, causing dependent VU commands to fail. Example: VIEWFILES=ct.V,inv.V ▶ The absolute pathname of the directory containing the view description files to VIEWDIR. If not set, tux_tpalloc() or tux_alloc_buf() calls that try to allocate a buffer of type VIEW, X_COMMON, or X_C_TYPE fail, causing dependent VU commands or functions to fail. Example: VIEWDIR=/u1/tuxapp/dat:/u1/tuxapp/dat2
Master or Agents running TUXEDO scripts that use VIEW32 typed buffers	<ul style="list-style-type: none"> ▶ A list of view description file names to VIEWFILES32. This variable is used by Agents running scripts that use VIEW32 typed buffers. If not set, VU functions that use VIEW32 typed buffers which are defined in view description files not in this list fail, causing dependent VU commands to fail. Example: VIEWFILES32=ct32.V,inv32.V ▶ The absolute pathname of the directory containing the view description files to VIEWDIR32. This variable is used by Agents running scripts that use VIEW32 typed buffers. If not set, tux_tpalloc() or tux_alloc_buf() calls that try to allocate a buffer of type VIEW32 fail, causing dependent VU commands or functions to fail. Example: VIEWDIR32=/u1/tuxapp/dat
Solaris Agents running TUXEDO scripts	<ul style="list-style-type: none"> ▶ The TLI network service provider pathname to WSDEVICE. This value is typically /dev/tcp. If not set, playback terminates with an error message. Example: WSDEVICE=/dev/tcp

To override the value of a system environment variable:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. Click the button in the **Sys Environment Variables** column. The Environment Variable Settings dialog box appears:



3. Type the name of the environment variable in the **Variable name** box.
4. Type the value of the environment variable in the **Variable value** box.
5. Click **Add**.

To change the value of a system environment variable that you have already set:

1. Click the name that you want to change.
2. Change the value in the **Variable value** box.
3. Click **Update**.

Initializing VU Environment Variables

You can initialize the value of most VU environment variable within LoadTest. To initialize the value of a VU environment variable:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. Click the button in the **VU Environment Variables** column. The VU Environment variables dialog box appears.

3. Click the tab that corresponds to the type of environment variable you want to initialize, and enter the initial value for the variable.

NOTE: For more information on VU environment variables and their meanings, see the *VU Language Reference*.

Changing the Log Level Value

The **Log level** column in the User Settings dialog box affects virtual users only. It sets the level of information written to each User Log file. You can set the log level to the following values:

- ▶ **Unexpected** – Log timeouts and unexpected responses from SQL emulation commands. For other emulation commands, **Unexpected** is equivalent to **Timeout**.
- ▶ **Timeout** – Log emulation commands that time out. If no commands time out while you are executing the schedule, no User Log file is created.
- ▶ **All** – Perform complete logging. A log entry is made for every emulation command.
- ▶ **Off** – Nothing is logged, and no User Log file is created.
- ▶ **Error** – Log all SQL emulation commands that are in error as well as those that time out. For other emulation commands, **Error** is equivalent to **Timeout**.

If you are running many virtual users, the recommended log level is **Error**. The default log level of **All** will work, but running the schedule requires more CPU, memory, and especially more disk resources.

However, if you have problems with the schedule and are debugging it, it is sometimes useful for one member of each user group to have a log level of **All**.

For more information about the log level, see the `Log_level` environment variable in the *VU Language Reference*.

To change the value of the log level:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. To change the log level for all users in a group, click the cell in the **Log Level** column for that group.

To change the log level for an individual user in a fixed user group, click the plus sign in the first column of the user group that contains that user. The list of users expands to display individual users. Click the cell in the **Log Level** column for that user.

3. Select a value for the log level.
4. Click **OK**.

Changing the Record Level Value

The **Record level** column in the User Settings dialog box affects virtual users only. It sets the level of information written to the result file. You cannot read the result file directly; LoadTest uses it to generate reports. You can set the record level to the following values:

- ▶ **MINIMAL** – Record only items necessary for reports to run. However, the reports will contain no real data. Use this value when you do not want the user's activity included in the reports.
- ▶ **TIMER** – **MINIMAL** plus `start_time` and `stop_time` emulation commands. Your reports will not contain response times for each emulation command, and an emulation command failure will not show up as a failure. In addition, the result file for each virtual user will be small. A small result file means that disk consumption and CPU overhead for each virtual user is less, results are retrieved quickly from Agent computers, and you can run reports in a relatively short time. Use this option if you are not concerned with the response times or pass/fail status of an individual emulation command.
- ▶ **FAILURE** – **TIMER** plus emulation command failures and some environment variable changes. Use this option if you want the advantages of a small result file but you also want to make sure that no emulation command failed.
- ▶ **COMMAND** – **FAILURE** plus emulation command successes and some environment variable changes (default).
- ▶ **ALL** – **COMMAND** plus all environment variable changes. Complete recording is done. This option is essentially the same as **COMMAND**, except it produces more detailed data in Trace report output.

For more information about the VU record level, see the `Record_level` environment variable in the *VU Language Reference*.

To change the record level:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. To change the record level for all users in a group, click the cell in the **Record Level** column for that group.

To change the record level for an individual user in a fixed user group, click the plus sign in the first column of the user group that contains that user. The list of users expands to display individual users. Click the cell in the **Record Level** column for that user.

3. Select a value for the record level.
4. Click **OK**.

Changing the Number of Start Scripts

Set this option in the User Settings dialog box only if you have changed the runtime settings so that you are starting users in groups. You may need to do this to control the maximum number of users that log on to a database server at the same time.

If you are starting users in groups, the **Start Scripts** column sets the number of scripts that the group of users must complete before the next group starts.

For example, assume the Data Entry user group contains 100 virtual users. Each virtual user runs a `Login` script and then selects three scripts in a random order: `Add New Record`, `Modify Record`, and `Delete Record`. You have changed the runtime settings so that the 100 users are not starting all at once; instead, they are starting in groups of 25.

If you enter a value of 1 in the **Start Scripts** column, the second group of 25 starts when each user in the first group of 25 completes the `Login` script. The third group of 25 starts when each user in the second group has completed the `Login` script, and so on.

To change the number of start scripts:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. To change the number of scripts for all users in a group, click the cell in the **Start Scripts** column for that group.

To change the number of scripts for an individual user in a fixed user group, click the plus sign in the first column of the user group that contains that user. The list of users expands to display individual users. Click the cell in the **Start Scripts** column for that user.

To restore the number of scripts to the inherited value, double-click in the cell.

3. Enter the number of scripts that the user must complete before the next group starts. Generally this is one script—a login script.
4. Click **OK**.

Limiting the Number of Scripts

The **Script Limits** column in the User Settings dialog box lets you limit the number of scripts that GUI or virtual users can run without having to remove any scripts from the user group.

For example, you can limit the number of scripts to:

- ▶ Test if the user (or user group, or all users) can complete the initial logon/setup scripts. By limiting the number of scripts, you don't have to delete the remainder of the scripts assigned to the user group and add them later, or create a new schedule just to run a simple test.
- ▶ Temporarily disable a user group without deleting it from the schedule. By setting **Script Limits** to 0 for the user group, you disable it. (You can also disable a fixed user group by setting the number of users to 0.)
- ▶ Vary the length of a schedule run. If your schedule contains nested scenarios with varying numbers of scripts, and random selection of those scenarios, it will be very complicated to use repetition counts to vary the length of the schedule run. A simpler way is to limit the number of scripts.

To limit the number of scripts:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. To limit the scripts for all users in a group, click the cell in the **Script Limits** column for that group.

To limit the scripts for an individual user, click the plus sign in the first column of the user group that contains that user. The list of users expands to display individual users. Click the cell in the **Script Limits** column for that user.

To restore the script limits to the inherited value, double-click in the cell.

3. Enter the maximum number of scripts.
4. Click **OK**.

Changing the Way Random Numbers Are Generated

Each user in a user group has a **user seed**, which generates random numbers in a script. These random numbers affect:

- ▶ A virtual user's think time, random number VU library routines, and random access in datapools.
- ▶ A GUI user's random access in datapools.

You can set user seeds to be unique or to be the same.

With a unique seed, each user who runs the same script will have a slightly different behavior. For example, if one virtual user thinks for 1.3 seconds before executing the first command, the second virtual user might think for 2.4 seconds. Although the individual think times will vary, they will have the same distribution around a mean value. The seeds also affect the VU library routines that have to do with random numbers. For example, if the first user calls the VU `uniform` routine twice and receives the numbers 5 and 3, other users in that group will probably receive different numbers, bounded only by the minimum and maximum values that are set in the script.

With the same seed, each virtual user who runs the same script will have exactly the same behavior. For example:

- ▶ If the first user thinks for 1.3 seconds before executing the first command, the second user (and all subsequent users) will also think for 1.3 seconds before executing that command.
- ▶ If the first user calls the VU `uniform` routine twice and receives the numbers 5 and 3, all other users in that group will also receive 5 and 3.

You can also set whether or not the random number generator is reseeded at the beginning of each script. In general, it is more desirable not to reseed, because one long pseudorandom sequence is better than many short ones.

To change the behavior of the default random number generator:

1. Click the **User Settings** button to the right of the schedule. The User Settings dialog box appears.
2. To change the seed flags for all users in a group, click the cell in the **Seed Flags** column for that group.

To change the seed flags for an individual user, click the plus sign in the first column of the user group that contains that user.

You can select one of the following options:

- **Unique and not reseeded** – Generates a unique seed for each user and does not reseed the random number generator at the beginning of each script. Each user in a user group will behave slightly differently. This is the most commonly used option in performance testing.
- **Unique and reseeded** – Generates a unique seed for each user and reseeds the random number generator at the beginning of each script. Each user in a user group will behave slightly differently, but the numbers are reseeded at the beginning of each script. You might require this option if you are doing government LTD (Live Test Demonstration) testing.

- **Same and not reseeded** – Generates the same seed for each user and does not reseed the random number generator at the beginning of each script. This is generally not a desirable option to select when modeling a realistic workload, because each user who runs the same script will behave in the same way. But this option may be useful for certain types of stress testing.
 - **Same and reseeded** – Generates the same seed for each user and reseeds the random number generator at the beginning of each script. This is generally not a desirable option to select when modeling a realistic workload, because each user who runs the same script will behave in the same way, and short pseudorandom sequences are not desirable. However, this option may be useful for certain types of stress testing.
3. To change the seed behavior for all users in a group, click the cell in the **Seed** column for that group.

To restore a seed to its inherited value, double-click in the cell.

To change the seed for an individual user, click the plus sign in the first column of the user group that contains that user.
 4. Click **OK**.

NOTE: The VU library routines that generate random numbers are `negexp`, `rand`, and `uniform`. For more information about these routines, see the *VU Language Reference*.

Viewing Schedules with the Asset Browser

The Asset Browser is a Rational Test window that displays your testing resources. When you are running LoadTest, the Asset Browser gives you quick access to all the schedules, reports, and results of schedule runs that are in your project.

To see the schedules in your project from the Asset Browser:

1. Click **View** → **Asset Browser**.
2. Double-click on the schedule that you want to view.

NOTE: For more information about the Asset Browser and how to structure queries, see the *Using Rational Robot* manual.

Deleting a Schedule

You may occasionally want to delete a schedule. For example, if you plan to change a schedule extensively, it may be easier to delete it and start with a new one.

When you delete a schedule:

- ▶ All of the relationships you have created in the schedule are deleted.
- ▶ All items defined in the schedule, such as selectors, synchronization points, scenarios, and delays, are deleted.
- ▶ Scripts are not deleted from the repository. You can reuse them in another schedule. For information about deleting a script from the repository, see the *Using Rational Robot* manual.
- ▶ Executables remain on your system. You can reuse them in another schedule.

To delete a schedule:

1. Click **View** → **Asset Browser**.
2. Click the schedule that you want to delete.
3. Click **Edit** → **Delete**.

Renaming a Schedule

To rename a schedule:

1. Click **View** → **Asset Browser**.
2. Click the schedule that you want to rename.
3. Click **Edit** → **Rename**.

Using Events and Dependencies to Coordinate Execution

An **event** is a mechanism that coordinates the way items are run in a schedule. For example, assume you are running a schedule that contains 100 virtual users that access a database. You want the first user to initialize the database, and the other 99 users to wait until the initialization is complete. To do this, you set a **dependency** on an event, which blocks the 99 users until the event—the first user initializes the database—occurs.

You can have multiple events in a schedule, but only one item in a schedule can *set* an event. However, many items can *depend* on the event.

The following schedule shows 99 users waiting until the first user initializes a database:



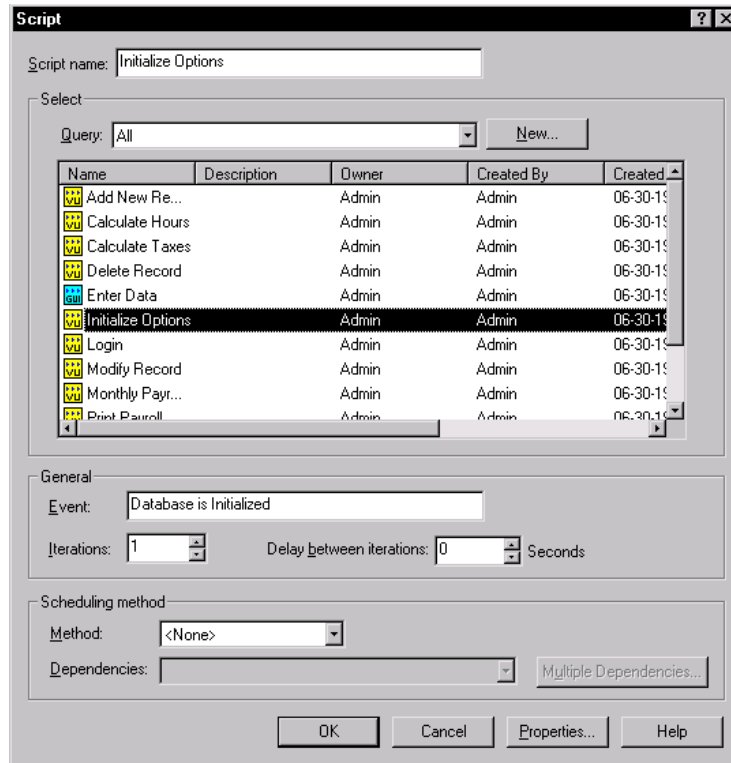
The second column in the schedule lists the events, and the third column lists the dependencies. In this schedule, as soon as the `Initialize Database` script completes, it sets the event `Database Is Initialized`. The `Add New Record`, `Modify Record`, and `Delete Record` scripts depend on this event and can run only after it is set.

NOTE: In the previous example, the `Data Entry` users run scripts randomly. Therefore, you must add a dependency to each script in the selector, because you do not know which script will run first. However, if the `Data Entry` users ran the scripts sequentially, you would need to add a dependency to the first script only.

Setting an Event

To add a script that sets an event:

1. Click **Insert** → **Script**.

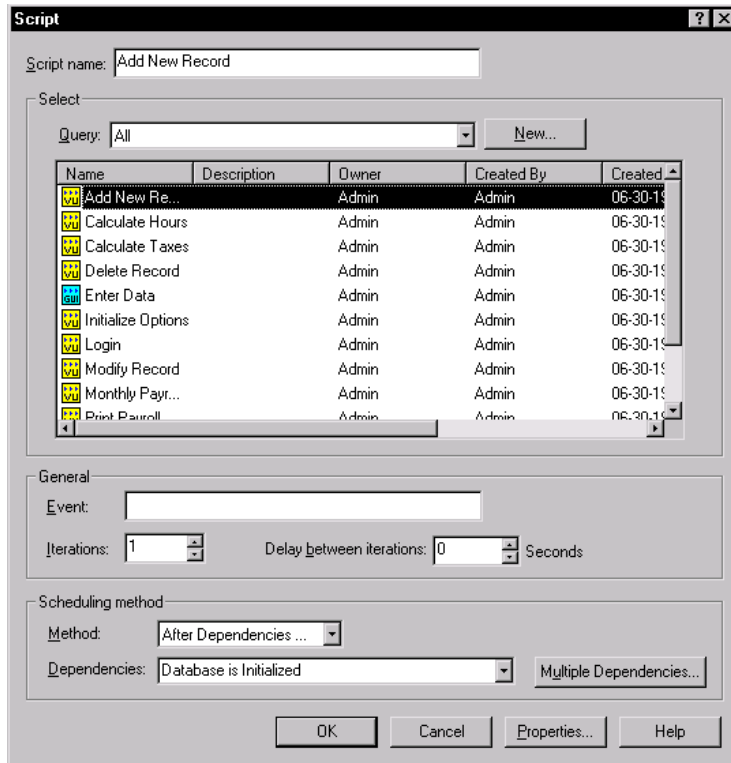


2. In the **Script name** box, enter the name of the script that will set an event.
3. In the **Event** box, type the name of the event.
4. In the **Iterations** box, select the number of iterations. If you set multiple iterations, the script sets the event after the last iteration completes.
5. Click **OK**. You have now made a script *set* an event.

Setting a Dependency on an Event

To add a script that depends on an event:

1. Click **Insert** → **Script**.



2. In the **Script name** box, enter the name of the script that will depend on an event.
3. In the **Scheduling method** box, select **After Dependencies**.
4. In the **Dependencies** box, select the name of the event. To make the script wait on more than one event, click **Multiple Dependencies**.
5. Click **OK**. You have now made a script *depend* on an event.

NOTE: The previous example shows how to add a script that sets an event and another script that depends upon an event. However, scenarios, executables, transactors, and delays can also set events, and executables can be dependent on an event.

Setting Shared Variables

If you are running virtual users, you can initialize shared variables in a schedule. A shared variable maintains its value across all scripts in a schedule. Each virtual user can access and change the shared variable.

Shared variables are useful in the following situations:

- ▶ For synchronizing users when you need more specific coordination than a synchronization point provides. For example, you may want to limit a transaction so that only five users perform it at once. In that case, you can use a shared variable with the VU language `wait` library routine.
- ▶ For blocking a user from executing until a global event occurs. It is generally easier to set an event and a dependency than to set a shared variable. However, if the event depends on some logic *within* a script, you must use a shared variable.
- ▶ For counting loops within a VU script. If you want to set a loop for an entire script, it is easier to set a selector or an iteration count within the schedule. However, if only a portion of the script loops, you can set a shared variable to control the number of iterations of that loop.
- ▶ For monitoring specific transaction counts and conditions. You can view shared variables when you monitor a schedule, and they provide detailed information about the progress or state of a schedule run.

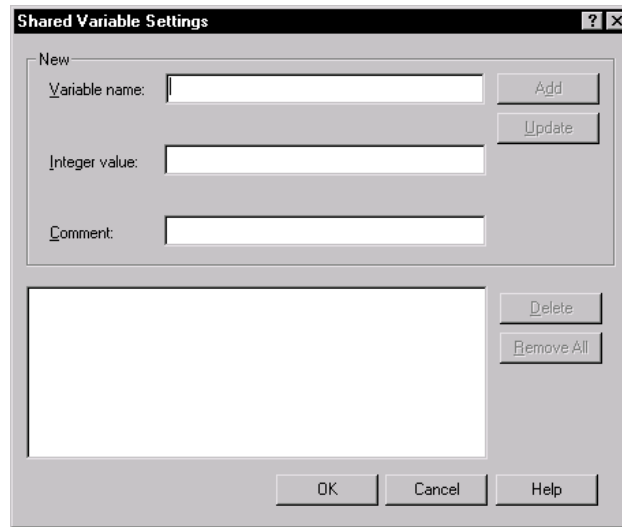
You *declare* a shared variable within a VU script with the keyword `shared`. For more information about declaring shared variables, see the *VU Language Reference*.

You *initialize* a shared variable within a schedule. This is optional—the default value is 0.

You *manipulate* the value of a shared variable through the logic in a VU script or when you monitor the schedule.

To initialize a shared variable:

1. Click the **Shared Variables** button to the right of the schedule.



2. In the **Variable name** box, type the name of the shared variable.
3. In the **Integer value** box, type the initial value for the shared variable. When you run the schedule, the shared variable is set to this value.
4. Optionally, type an explanation of the shared variable In the **Comment** box.
5. Click **Add**. The shared variable and its value appear in the bottom portion of the dialog box. If you add a comment, the comment appears as well.

To change a shared variable:

1. Click the **Shared Variables** button to the right of the schedule. The Shared Variable Settings dialog box appears, with a list of the shared variables that you have set.
2. Click the shared variable that you want to change. The variable name and value appear in the top portion of the dialog box.
 - To modify the shared variable, change its name or value and click **Update**.
 - To delete the value of a shared variable, click **Delete**.
3. To reset all of the shared variable values to 0, which is the default initialization, click **Remove All**.
4. Click **OK**.

Printing and Exporting a Schedule

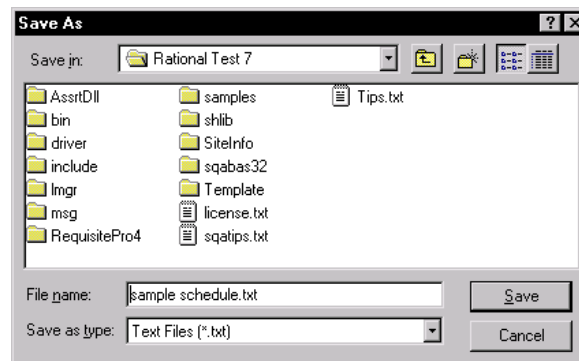
Designing a schedule can involve many iterations and changes. You may find it helpful to examine a printed view of a schedule. You can print a schedule or export it as a .txt file.

To print a schedule:

1. Click **File** → **Open** → **Schedule**, and open the schedule you want to print.
2. Click **File** → **Print**.
3. Adjust the printing settings, if necessary, and click **OK**.

To export a schedule as a .txt file:

1. Click **File** → **Open** → **Schedule**, and open the schedule you want to save.
2. Click **File** → **Export to File**.



3. Select a folder. The default folder is your current directory.
4. Enter a file name for the exported data. The default file name is the name of the schedule.
5. Click **Save**.

Saving a Schedule

After you have finished modifying a schedule, you should save your modifications. A schedule that is not saved has an asterisk in the title bar.

To save a schedule:

- ▶ Click **File** → **Save**.

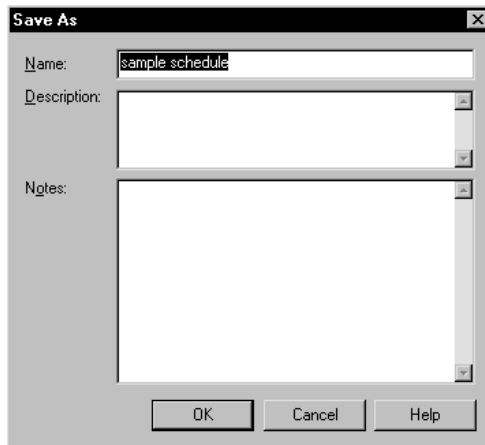
To save more than one schedule:

- ▶ Click **File** → **Save All**.

NOTE: If you click **Tools** → **Options**, click the **Create Schedule** tab, then select the **Check schedule when saving** box, one verification screen appears for each schedule being saved.

To save a schedule under a different name:

1. Click **File** → **Save As**.



2. Type a new name in the **Name** box.

If the schedule with that name already exists, a warning dialog box asks whether you want to overwrite the existing schedule.

3. Optionally, type a description. If you have many similar schedules, it is useful to add a description. This description appears when you click **File** → **Properties**.

4. Optionally, type a note. If you have many similar schedules, it is useful to add a note.

NOTE: **Description** and **Notes** are both fields that you can display when you create a query. For information about creating queries, see the *Using Rational Robot* manual.

5. Click **OK**.

Checking a Schedule

While you are working on a schedule, you might change it so that it will not run correctly. For example, you might insert a script into a schedule before it is recorded. Although LoadTest automatically checks a schedule before it runs, you can check a schedule at any time to correct problems without actually running it.

LoadTest checks a schedule for many kinds of errors, including the following:

- ▶ The schedule contains a script that exists in the repository but is empty. To correct this problem, record or create the script with Robot.
- ▶ The schedule does not contain any user groups. A schedule must have at least one user group to run.
- ▶ The schedule contains an empty user group. Either delete the user group or add scripts and other items to it.
- ▶ A user group contains an empty scenario. Either delete the scenario or add items to it.
- ▶ The schedule contains a selector that is empty.

To check a schedule:

- ▶ Click **Run** → **Check Schedule**.

LoadTest displays any problems with the schedule in a separate window.

NOTE: You can set options so the schedule is checked automatically whenever you save it. To check the schedule automatically, click **Tools** → **Options**, click the **Create Schedule** tab, then select the **Check schedule when saving** check box.

Checking Agent Computers

If you are running users on Agent computers, it is a good idea to check the Agents before you run the schedule. This way, you can determine whether any problems exist before you run the schedule.

When you check Agent computers, LoadTest ensures that:

- ▶ All of the Agent computers scheduled for the users actually exist. For example, if you incorrectly typed the name of an Agent computer, LoadTest notifies you.
- ▶ The Agent computers are available and running.
- ▶ The Agent software is running. The same release of PerformanceStudio software must be installed on both the Master and the Agent computers.

To check the Agent computers:

- ▶ Click **Run** → **Check Agents**.

LoadTest displays any problems with the Agent computers in a separate window.

Controlling Runtime Information of a Schedule

LoadTest lets you control the way a schedule runs. The following list describes the types of runtime settings that you can change:

- ▶ How users are started—either all at once or in groups.
- ▶ The criteria for whether a schedule passes or fails.
- ▶ The order in which the user groups run.
- ▶ Timing information such as how long the run should be, how long to take to initialize the system, whether to suppress timing delays, and whether to initialize timestamps for each script.
- ▶ The number to feed to the random number generator.

To set the runtime settings for a schedule:

1. Click the **Runtime** button to the right of the schedule.

2. Set the **Start group information**. This option specifies how many users to start at the same time. You can adjust this number to avoid overloading the server. The next group of users can start after the current group of users has finished initializing. You can select one of the following:
 - **Start all users at once** – Select this to start all users at the same time.
 - **Start users in groups** – Select this to set the number of users to start at a time in the **Number to start at a time** box.

NOTE: If you are starting users in groups, you should also set the number of scripts that each user needs to run before the next group starts. To do this, click the **User Settings** button to the right of the schedule, and modify the **Start Scripts** box.

3. Set the **Schedule pass criteria**. This option determines what the LogViewer lists as a “passing” schedule. If the schedule does not meet the criteria, the LogViewer lists the Schedule Start and Schedule End events as “failed.” You can select one or more of the following:
 - **Schedule ran to completion** – The schedule ran to completion without manual termination of the run.
 - **All users completed normally** – All users in the schedule were able to complete all of their assigned schedule items.
 - **All GUI scripts returned success** – All GUI scripts passed, which means that no verification points failed and no user action commands timed out.

4. Set the **Execution order**. This option defines the order in which users are started, and therefore determines which user groups are executed if you run fewer users than the maximum number defined. You can select one of the following:
 - **Sequential** – Runs each user as it is encountered in the schedule (from the top to the bottom).
 - **Balanced** – Balances the run among the user groups in proportion to the schedule.
 - **Custom** – Lets you select specific user groups or users to run. This option, which applies to fixed user groups only, can be useful for troubleshooting. For example, if a script does not work, and that script is used only by the Accounting group, you can run that group only. Click the **Define** button to run a particular user or user group, and then click **Add** to add that user or group to the execution list.

NOTE: You can also temporarily disable a fixed user group by setting the number of users to 0. Right-click on the user group, click **Properties**, and set **Number** to 0. LoadTest ignores the user group when you run the schedule.

Run fixed users first – Runs the fixed user groups before the scalable user groups, regardless of the execution order. If your user groups are all fixed, this option has no effect.

Assume that you have defined four user groups with a total of 101 users, and you have the following schedule:

Schedule	Event	Dependencies	User Groups
<ul style="list-style-type: none"> [-] User Groups <ul style="list-style-type: none"> [+] End of Month Accounting: 1 user(s) [+] Accounting: 20 user(s) [+] Data Entry: 30 user(s) [+] Sales: 50 user(s) [+] Scenarios 			<ul style="list-style-type: none"> Computers User Settings Runtime Termination Shared Variables

Although your schedule contains 101 users, you want to run it with only 10 users. The following table lists how the execution order affects which users are run:

If you select	You run these user groups
Sequential	1 End of Month Accounting 9 Accounting
Balanced	2 Accounting 3 Data Entry 5 Sales
Balanced and Run Fixed Users First	1 End of Month Accounting 2 Accounting 3 Data Entry 4 Sales

5. Set the **Time information**. This option specifies the maximum amount of time for the schedule to run, how long you want to allow for users to initialize, whether to include timing delays, and whether to keep the timestamps from script to script or to reset them for each script. You can select one or more of the following:
 - **Duration of run** – Specify the maximum number of seconds for the schedule to run. If you select 0, no time limit is imposed on the schedule run.
 - **Initialization time** – The maximum number of seconds for all users to confirm that they completed initializing. If you have changed the number of start scripts, make sure that you set this time high enough.
 - **Suppress timing delays** – Runs a schedule very quickly because all timing delays in virtual user scripts are suppressed. This choice is useful if you are testing a schedule to see whether it runs correctly and you are not interested in the timing delays. Note, however, that this creates a maximum workload on the server, Master, and Agent computers. Do not suppress timing delays if you are running a large number of virtual users.
 - **Initialize timestamps for each script** – Indicates whether timestamps are carried over from script to script or are reinitialized with each virtual user script.
6. In the **Seed** box, select a seed number. LoadTest uses this seed to generate the random numbers for selectors and shared access in datapools.

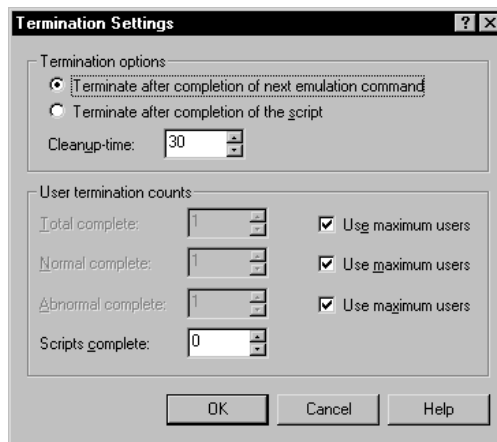
7. In the **Enable IP Aliasing** box, click to use IP aliasing. This makes each virtual user have a different source IP address. This option has meaning only if you are running HTTP scripts, and your system administrator has set up your computer to use IP aliasing. For information on setting up IP Aliasing, see Appendix B, *Configuring Master and Agent Computers*.
8. Click **OK**.

Controlling How a Schedule Terminates

LoadTest lets you set the conditions that force a schedule to terminate. For example, you may want to stop a schedule if you discover that a large number of users are completing abnormally, indicating that something is wrong with the run.

To control how a schedule terminates:

1. Click the **Termination** button to the right of the schedule.



2. Under **Termination Options**, select what happens if you manually stop the schedule when it is running. You can select the following options:
 - **Terminate after completion of next emulation command** – Each user exits after the next emulation command finishes. This option usually stops a run quickly.
 - **Terminate after completion of the script** – Each user exits after the current script finishes. If you select this option, enter a larger value for the **Clean-up time**.

- **Clean-up time** – Enter the number of seconds allowed from the time a termination is requested to the time LoadTest forces the termination of a user.
3. Set the **User termination counts**. When you run a schedule, as the virtual users log off, you may no longer be interested in measuring the workload. For example, if you run a schedule with 1000 users, after 700 users log off, you may no longer be interested in measuring the system response and you may want to terminate the schedule to save time. This option lets you set the number of users and the number of scripts that, when complete, trigger the termination of the schedule.
- **Total complete** – Enter a number of users. When that number completes, the schedule terminates. If you select **Use maximum users**, the schedule terminates when all users finish.
 - **Normal complete** – Enter a number of users. When that number completes successfully, the schedule terminates. If you select **Use maximum users**, the schedule terminates when all users finish.
 - **Abnormal complete** – Enter a number of users. When that number completes abnormally, the schedule terminates. If you select **Use maximum users**, the schedule terminates when all users finish.
 - **Scripts complete** – Enter a number of scripts. When that number of scripts finish running, the schedule terminates. A value of 0 means that no limit is placed on the number of scripts, and the schedule terminates when it has finished running all of the items in it.

NOTE: The VU library routines that generate random numbers are `negexp`, `rand`, and `uniform`. For more information about these routines, see the *VU Language Reference*.

4. Click **OK**.

Running a Schedule

When you run a schedule, each user executes its assigned schedule items. The results of running the schedule are stored in **logs**. After you run the schedule, you can run reports to analyze the data stored in the logs and display run results in the form of graphs and charts.

To run a schedule:

1. Click **Run** → **Schedule**.

The screenshot shows the 'Run Schedule' dialog box with the following details:

- Schedule information:** Name: sample schedule; Number of users: 30; Max Users: Scalable to 2000 License Limit.
- License Information:** Max. No. GUI Users: 1000; Max. No. Virtual Users: 1000.
- Log Information:** Build: Build 1; Log Folder: sample schedule; Log: Users 30 #01.
- Resource Monitoring:** Monitor resources; Update rate (sec): 5.

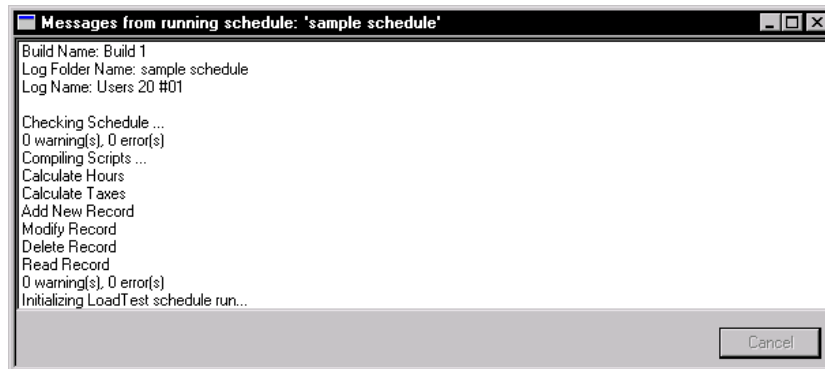
2. To change the schedule, click the **Change** button. The Open Schedule dialog box appears, from which you can select a different schedule to run.
3. Enter the number of users that you want to run in the **Number of users** box. The dialog box displays the number of virtual and GUI users that your license permits.

If a schedule includes both fixed and scalable user groups, the fixed user groups are assigned first. So, for example, if your schedule includes one user group fixed at 10 users, and you run 100 users, 10 users are assigned to the fixed user group, and the remaining 90 users are distributed among the scalable user groups.

4. The name of the **Log Folder** is based on the schedule, and the **Log Name** is based on the number of users and the number of times you have run the schedule. For example, if you run the sample schedule three times, with 10 users, 15 users, and 20 users, all three logs will be in the sample schedule folder. The log names will be U sers 10 # 01, U sers 15 # 02, and U sers 20 # 03. Therefore, the log name U sers 20 # 03 indicates that this is the third time you have run the schedule, and the schedule is being run with 20 users.

To change the log folder or log name, click the **Change** button.

5. If you plan to monitor resources, select the **Monitor resources** check box. This option lets you monitor your computer resource usage when you play back the schedule and then graph this usage data over the corresponding user response times when you analyze your results.
6. To change the rate at which the views get updated, type a new update interval in the **Update Rate** box. The lower the interval, the faster the update.
7. Click the **Options** button to change the options that appear when you monitor the schedule.
8. Click **OK**. LoadTest displays a Preparing to Run window similar to the following:



LoadTest checks the schedule, and compiles any uncompiled or out-of-date scripts.

To cancel the schedule run while LoadTest is checking the schedule, click **Cancel**.

After LoadTest checks the schedule and compiles the necessary scripts, it minimizes the Preparing to Run window. At this point, you cannot cancel or dismiss this window. To stop the run, click **Run** → **Stop Schedule**.

For information about monitoring a schedule as it runs, see Chapter 8, *Monitoring Schedules*.

▶ ▶ ▶ C H A P T E R 8

Monitoring Schedules

This chapter discusses how to monitor a schedule. It includes the following topics:

- ▶ About monitoring schedules
- ▶ Displaying the schedule views
- ▶ Displaying a histogram
- ▶ Displaying the user views
- ▶ Displaying the Shared Variables view
- ▶ Displaying the Script view
- ▶ Displaying the Sync Points view
- ▶ Displaying the Computer view
- ▶ Displaying the Transactor view
- ▶ Displaying the Group views
- ▶ Filtering and sorting views
- ▶ Changing shared variable information manually
- ▶ Debugging a script
- ▶ Changing monitor defaults
- ▶ Controlling the schedule during a run

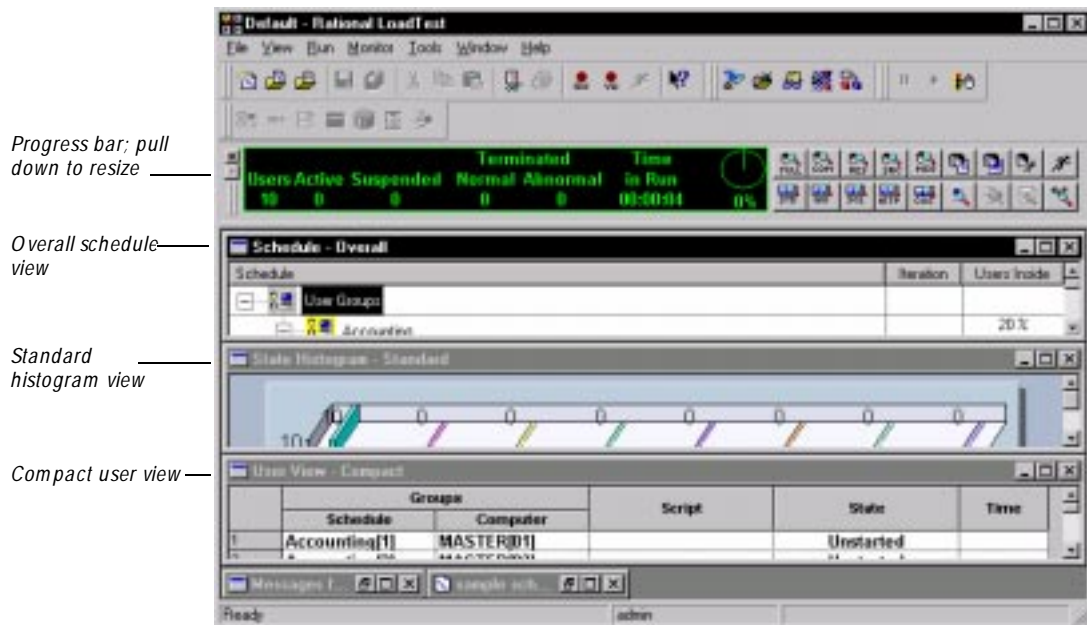
About Monitoring Schedules

While a schedule is running, you may want to monitor its progress. Monitoring a schedule lets you not only confirm that a schedule is progressing successfully, but also lets you discover potential problems early in the run so you can intervene if necessary. You can suspend and restart virtual users, change the values of shared variables, and release users waiting on synchronization points.

LoadTest's monitoring tools provide you with up-to-date information that is dynamically updated as the schedule runs. This information includes:

- ▶ The number of commands that have executed successfully and the number of commands that have failed.
- ▶ The general state of the users: whether they are initializing, connecting to a database, exiting the schedule, or performing other tasks.
- ▶ Whether any users have terminated abnormally.

When you run a schedule, LoadTest displays the monitoring information in a Progress bar and in views. The Progress bar gives you a quick summary of the state of the run and cannot be changed. You can change the views, however, to provide summary information or detailed information about each user. The following figure shows the Progress bar and the default views:



The Progress bar lets you quickly assess how successfully the schedule is running. The Progress bar provides the following information:

- ▶ **Users** – The total number of users in the run.
- ▶ **Active** – The number of user that are neither suspended or terminated.
- ▶ **Suspended** – The number of virtual users in a paused state.
- ▶ **Abn. Term** – The number of users that terminated without completing all of their assigned tasks.
- ▶ **Normal Term.** – The number of users that completed their tasks successfully.
- ▶ **Time in Run** – The time the schedule has been running, expressed in *hours:minutes.seconds*.
- ▶ **%Done** – The approximate percentage of the schedule that has completed.

LoadTest also displays three views of the running schedule:

- ▶ The Overall Schedule view, which displays general information about the status of users. For more information, see *Displaying the Schedule Views* on page 8-4.
- ▶ The Standard Histogram view, which is a bar chart that provides a general idea of what tasks the users are performing. For example, some users might be initializing, some users might be executing VU code, and some users might be connecting to the database. This chart shows the number of users in each state.

LoadTest displays the Standard Histogram view by default. However, if your users are running GUI scripts, or if you are testing SQL database or Web performance, you may want to display a bar chart specifically geared to those tests. For more information, see *Displaying the Histogram Views* on page 8-5.

- ▶ The Compact user view, which displays information about the current state of the users. In this view, you can click on a particular user to display additional information about that user or control its operation. For more information, see *Displaying the User Views* on page 8-12.

Displaying the Schedule Views

The schedule views are very similar to the actual schedule that you have designed. Columns show you which iteration is being executed and what percentage of the users in a group are currently in a script or a selector.

To display a schedule view:

1. Click **Monitor** → **Schedule**.
2. Select one of the schedule views:
 - **Overall** – Use this view to display general information about the status of the schedule. LoadTest displays this view by default when you run a schedule. The following figure shows an Overall schedule view:

Schedule	Iteration	Users Inside
<ul style="list-style-type: none"> [-] User Groups <ul style="list-style-type: none"> Accounting: 20.000% <ul style="list-style-type: none"> Calculate Hours: 1 time(s) Calculate Taxes: 1 time(s) Data Entry: 30.000% <ul style="list-style-type: none"> Set Up Database Application: 1 time(s) Random without replacement: 10 selection(s) <ul style="list-style-type: none"> Add New Record: 1 time(s), 2 wt. Modify Record: 1 time(s), 7 wt. Delete Record: 1 time(s), 1 wt. Sales: 50.000% <ul style="list-style-type: none"> Set Up Database Application: 1 time(s) Read Record: 1 time(s) Scenarios 		
		20 %
		0 %
		0 %
		30 %
		0 %
	3/10	100 %
		0 %
	1/1	100 %
		0 %
		50 %
		0 %
		0 %

The Overall schedule view is similar to the actual schedule that you have designed. However, it contains two additional columns:

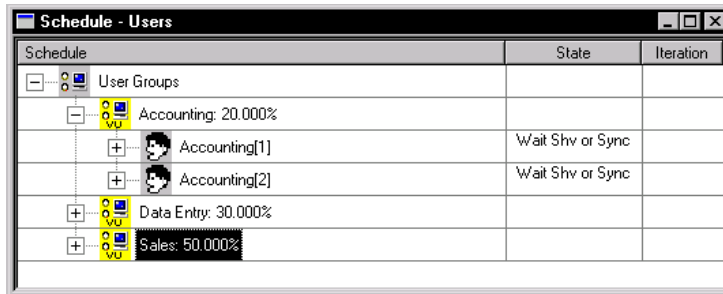
The **Iteration** column shows how many iterations are in the schedule item and the iteration in progress, averaged over all users currently executing that schedule item.

For example, 3/10 indicates that, for the users currently executing that schedule item, on the average, the users are executing the 3rd of 10 total iterations.

The **Users Inside** column shows the percentage of users that are currently executing each portion of the schedule. The percentage next to the user group shows the percentage of total users that have been assigned to the group and have not yet exited the schedule. The percentage next to the items within a user group, however, shows the percentage of users within that group that are executing that item.

For example, if the Sales user group contains 50 percent of the total users, the Users Inside column for that group is 50 percent. If all users in the Sales group are executing the Read Record script, the Users Inside column for that script is 100 percent.

- **User** – Use this view to display the exact schedule progress of a particular user. The following figure shows a Users schedule view:



Displaying the Histogram Views

The histogram views group the users into various states, such as exiting and initializing. Use a histogram view to display a bar graph of how many users are in each state.

To display a histogram view:

1. Click **Monitor** → **Histogram**.
2. Select one of the histogram views:
 - **Standard** – Data is grouped in a general way. Select this histogram if you want a general overview of the user states.
 - **GUI** – Data is grouped appropriately for tests that run GUI scripts.
 - **SQL** – Data is grouped appropriately for tests that access SQL databases.
 - **HTTP** – Data is grouped appropriately for tests that access Web servers.

- **IIO P** – Data is grouped appropriately for tests that access IIO P servers.
- **Custom** – Data is grouped according to your needs. For information about customizing a histogram, see *Configuring Custom Histograms* on page 8-36.

The following figure shows a Standard histogram:



In this histogram, seven users are in the Quiet state and three users have exited.

The following sections describe the different types of histograms that you can display.

Standard Histograms

In a Standard histogram, which is displayed by default, data is grouped in a general way. The following table describes the bars in a Standard histogram:

Bar Name	Description
Unstarted	The process associated with the user has not started. If you see this state for a while, you have probably started users and scripts in groups. Until a group completes initialization, subsequent groups are in this state.
Init	Users that are initializing.
Quiet	Users that are thinking, delaying, or suspended, or waiting on a shared variable or synchronization point.
Server	States related to communicating with the server.

(Continued)

Bar Name	Description
Code	Users that are executing VU or SQABasic code.
Overhead	States related to run overhead. These states are GetTask (getting the next schedule task), InitScript (initializing a script), Match (pattern matching), and Read_Shv (reading a shared variable over the network).
GUI	Users performing GUI-related operations.
Exit	Users that have finished the schedule, with either normal or abnormal termination.

GUI Histograms

A GUI histogram displays information pertinent to tests that run GUI scripts. The following table describes the bars in a GUI histogram:

Bar Name	Description
Init	Users that are initializing.
Input	Users that are typing input.
WaitApp	Users that are waiting on an application output.
Quiet	Users that are delayed.
Code	Users that are executing SQABasic code.
Overhead	States related to run overhead. These states are GetTask (getting the next schedule task) and InitScript (initializing a script).
Other	All other states.
Exit	Users that have finished the schedule, with either normal or abnormal termination.

SQL Histograms

A SQL histogram displays information pertinent to tests that access SQL databases. The following table describes the bars in a SQL histogram:

Bar Name	Description
Init	Users that are initializing.
Connect	Users that are waiting to connect to the database server.
Exec	Users that are executing VU <code>sqlprepare</code> or <code>sqlexec</code> statements.
Query	Users that are waiting on the results of a query.
Quiet	Users that are thinking, delaying, or suspended, or waiting on a shared variable or synchronization point.
GUI	Users that are performing GUI-related operations.
Other	All other states.
Exit	Users that have finished the schedule, with either normal or abnormal termination.

HTTP Histograms

An HTTP histogram displays information pertinent to tests that access Web servers. The following table describes the bars in an HTTP histogram:

Bar Name	Description
Init	Virtual users that are initializing.
Connect	Virtual users that are waiting to connect to the Web server.
Send	Virtual users that are sending data to the Web server.
Recv	Virtual users that are waiting for data from the Web server.
Linespeed	Virtual users that are being artificially delayed to achieve a specific network linespeed.

Bar Name	Description
Quiet	Virtual users that are thinking, delaying, or suspended, or waiting on a shared variable or synchronization point.
GUI	Virtual users that are performing GUI-related operations.
Other	All other states.
Exit	Virtual users that have finished the schedule, with either normal or abnormal termination.

IIO P Histograms

An IIO P histogram displays information pertinent to tests that access IIO P protocol. The following table describes the bars in an IIO P histogram:

Bar Name	Description
Init	Virtual users that are initializing.
Bind	Virtual users that are obtaining an object reference.
Connect	Virtual users that are waiting to connect to the server.
Invoke	Virtual users that are invoking a remote operation.
Quiet	Virtual users that are thinking, delaying, or suspended, or waiting on a shared variable or synchronization point.
GUI	Virtual users that are performing GUI-related operations.
Other	All other states.
Exit	Virtual users that have finished the schedule, with either normal or abnormal termination.

Zooming In on Histogram Bars

Each bar in a histogram shows a summary state that contains individual states. You can zoom in on a bar to see a breakdown of how many users are in each state.

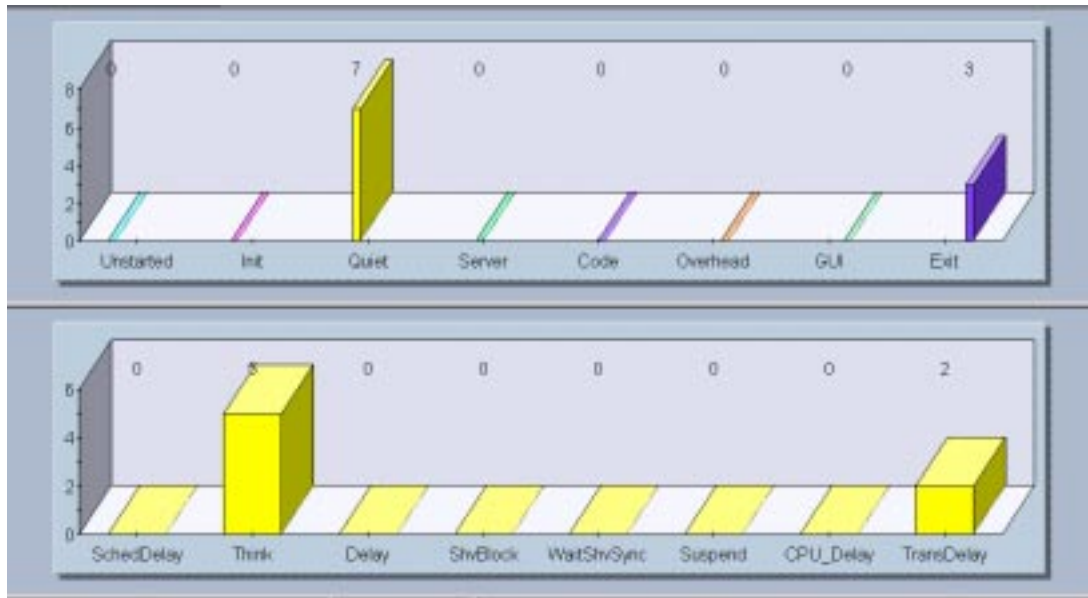
To zoom in on a histogram bar:

- ▶ Double-click on a bar that contains users. A window appears that displays the individual states.

NOTE: To restore the window to its original state, click **View** → **Reset**.

Monitoring Schedules

The following figure shows an expanded histogram, after you have clicked on the Quiet bar:



Seven users are classified as Quiet. The expanded histogram shows that five users are in the Think state, and two are in the TransDelay state.

The individual states in the histogram bars are as follows:

This bar	Indicates that
Bind	Virtual users that are obtaining an object reference.
Cleanup	A virtual user is cleaning up (process cleanup) before exiting.
CPU_Delay	A virtual user is emulating a CPU (client application) delay before submitting a command to the server.
Connect	A virtual user is executing a connect emulation function.
Disconn	A virtual user is executing a disconnect emulation function. However, the client has not yet disconnected.
Delay	A virtual user is executing the VU delay routine, or a GUI user is executing the SQABasic DelayFor() command.
Exit	A virtual user has exited.
ExitSQABasic	A GUI user has finished executing an SQABasic script.

(Continued)

This bar	Indicates that
ExternC	A virtual user is executing an external C routine called from within the script.
GetTask	A virtual or GUI user is waiting to be assigned its next schedule task.
Keys	A GUI user is keystroking.
Init	A virtual or GUI user is being initialized (process initialization).
InitScript	A virtual user is preparing to execute a script.
Invoke	Virtual users that are invoking a remote operation.
Match	A virtual user is executing <code>tux_prev</code> pattern-matching code while the user is waiting for completion of the server's response.
Read_Shv	A virtual user is running on an Agent computer and reading a shared variable from the Master computer.
RecvDelay	A virtual user encountered a linespeed delay when receiving.
SchedDelay	A virtual or GUI user is executing a delay that you set in the schedule.
SendDelay	A virtual user encountered a linespeed delay when sending.
Sending	A virtual user is sending data with an <code>http_request</code> or <code>sock_send</code> emulation command.
ShvBlock	A virtual user is temporarily blocked while trying to obtain exclusive access to change the value of a shared variable.
SQABasic	A GUI user is executing SQABasic code.
SQL_Exec	A virtual user is executing or preparing a SQL command (<code>sqlexec</code> or <code>sqlprepare</code>) and waiting for the server to complete the operation.
StartApp	A GUI user is starting an application within a GUI script.
Suspend	A virtual user has been suspended.
TestCase	A virtual user is executing a <code>testcase</code> or <code>emulate</code> command.
Think	A virtual user or a GUI user is thinking before submitting a command to the server.
TransDelay	A virtual user or a GUI user is delayed waiting for the next transaction.
Tuxedo	A virtual user is executing a TUXEDO emulation command and waiting for the server to complete the operation.

(Continued)

This bar	Indicates that
Unstarted	The process associated with the user has not started. If you see this state for a while, you have probably started users and scripts in groups. Until a group completes initialization, subsequent groups are in this state.
VU_Code	A virtual user is executing VU code unrelated to user emulation commands. For example, the user may be accessing a datapool or performing logic that you added to the script.
WaitResp	A virtual user is waiting for completion of the server's response (a receive emulation command).
WaitShvSync	A virtual user is waiting on an event in a schedule, waiting to be released from a synchronization point, or executing the <code>wait</code> routine, but the event has not yet occurred.
WaitObj	A GUI user is waiting for a window or control to appear.

Displaying the User Views

The user views display the status and details of GUI and virtual user operations. Display one of the user views to see the status of individual users.

To display a user view:

1. Click **Monitor** → **User**.
2. Select one of the user views:
 - **Full** – Contains complete information about all users.
 - **Compact** – Contains summary information about all users. This is the most efficient user view to use when you are running Agent computers.
 - **Results** – Contains information about the success and failure rate of each VU emulation command.
 - **Source** – Displays the line number and the name of the source file being executed.
 - **Message** – Similar to the Compact user view, but also displays the first 20 letters of text from the VU `display` library routine.

The following items apply to all user views:

- ▶ To make tracking certain users easier, you can change which users are displayed. For more information, see *Filtering and Sorting Views* on page 8-27.
- ▶ When you display a user view, you can also display the script that the user is currently running. Simply double-click on the number in the first column, next to the user. LoadTest displays the script. For more information, see *Displaying the Script View* on page 8-18.
- ▶ When a user terminates abnormally, LoadTest writes a message stating the reason for termination to the running Schedule window. You can view this message from any user view. Right-click on the terminated user, and then select the **View Termination Message** option.

A user that terminates abnormally can be easily identified in the user views because its **Exited** state is displayed in red.

The rest of this section describes and gives examples of each user view.

Compact User View

The Compact user view contains summary information about all users. The following figure shows an example of a Compact user view:

	Groups		Script	State	Time
	Schedule	Computer			
1	Accounting[1]	MASTER[01]	Calculate	Thinking	00:00:00
2	Accounting[2]	MASTER[02]	Calculate	Thinking	00:00:01
3	Data Entry[1]	MASTER[03]		Exited	
4	Data Entry[2]	MASTER[04]		Exited	
5	Data Entry[3]	MASTER[05]		Exited	
6	Sales[1]	MASTER[06]		Exited	
7	Sales[2]	MASTER[07]		Exited	
8	Sales[3]	MASTER[08]		Exited	
9	Sales[4]	MASTER[09]		Exited	
10	Sales[5]	MASTER[10]		Exited	

The Compact user view displays the following information:

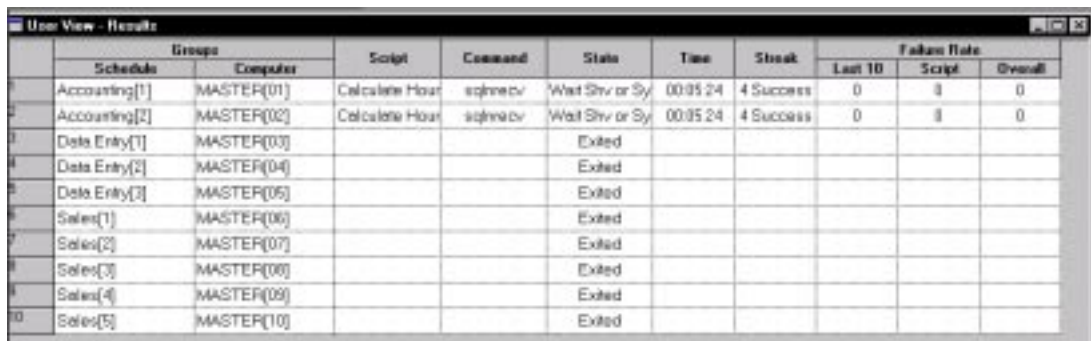
- ▶ **Groups** – Contains information about the user group. The **Schedule** column displays the user group to which the user belongs, as well as a number identifying the individual user within the group. This identification remains constant throughout the run. The **Computer** column displays the computer on which the user is running.
- ▶ **Script** – The script that each user is running. In this example, both Accounting users are running the Calculate Hours script.

Monitoring Schedules

- ▶ **State** – The state that the user is in. In this example, two users are thinking and eight users have exited the run. If a user has terminated abnormally, LoadTest displays the word **Exited** in red.
- ▶ **Time** – The time each user has been in that state. In this example, the Accounting[2] user has been thinking for 1 second.

Results User View

The Results user view contains information about the success and failure rate of each VU emulation command. The following figure shows an example of a Results user view:



	Groups		Script	Command	State	Time	Streak	Failure Rate		
	Schedule	Computer						Last 10	Script	Overall
1	Accounting[1]	MASTER(01)	Calculate Hour	sqlnrecv	Wait Shv or Sy	00:05:24	4 Success	0	0	0
2	Accounting[2]	MASTER(02)	Calculate Hour	sqlnrecv	Wait Shv or Sy	00:05:24	4 Success	0	0	0
3	Data Entry[1]	MASTER(03)			Exited					
4	Data Entry[2]	MASTER(04)			Exited					
5	Data Entry[3]	MASTER(05)			Exited					
6	Sales[1]	MASTER(06)			Exited					
7	Sales[2]	MASTER(07)			Exited					
8	Sales[3]	MASTER(08)			Exited					
9	Sales[4]	MASTER(09)			Exited					
10	Sales[5]	MASTER(10)			Exited					

In addition to the information displayed in the Compact user view, the Results user view contains the following information:

- ▶ **Command** – The VU emulation command that is executing. In this example, the two Accounting users are executing the VU emulation command `sqlnrecv`.
- ▶ **Streak** – Succession of successes or failures of VU language emulation commands. For example, 4 successes indicates that 4 emulation commands in a row have been successfully executed.
- ▶ **Failure Rate** – The number of failures in the last ten VU emulation commands, the number of failures in the current script, and the number of failures overall.

Source User View

The Source user view displays the line number and the name of the source file that is being executed. The following figure shows an example of a Source user view:

	Groups		Script	Command	State	Time	Source		Cmd Count
	Schedule	Computer					File	Line	
1	Accounting[1]	MASTER[01]	Calculate	sqlnrecv	Wait Shw	01:35:02	Calculate	37	4
2	Accounting[2]	MASTER[02]	Calculate	sqlnrecv	Wait Shw	01:35:03	Calculate	37	4
3	Data Entry[1]	MASTER[03]			Exited				
4	Data Entry[2]	MASTER[04]			Exited				
5	Data Entry[3]	MASTER[05]			Exited				
6	Sales[1]	MASTER[06]			Exited				
7	Sales[2]	MASTER[07]			Exited				
8	Sales[3]	MASTER[08]			Exited				
9	Sales[4]	MASTER[09]			Exited				
10	Sales[5]	MASTER[10]			Exited				

In addition to the information displayed in the Compact user view, the Source user view contains the following information:

- ▶ **Command** – The VU emulation command that is executing. In this example, the two Accounting users are executing the VU emulation command `sqlnrecv`.
- ▶ **Source** – Generally the same as the script. However, if a GUI script calls another script, or if a VU script contains an include file, the called script or the include file is displayed.
- ▶ **Cmd Count** – The number of VU emulation commands that have been executed in the current script. This number helps you distinguish between executions of the same command on different loop iterations. In this example, the first user is on line 37 and the command count is 4. If line 37 is part of a loop, the next time LoadTest executes that line, the line number is the same but the command count increases.

Message User View

The Message user view is similar to the Compact user view, but it also displays messages from the VU display library routine. If you have added this routine to a VU script, you may want to show this view.

The following figure shows an example of a Message user view:

	Groups		Script	State	Time	Message
	Schedule	Computer				
1	Accounting[1]	MASTER[01]	Calculate	Thinking	00:00:00	
2	Accounting[2]	MASTER[02]	Calculate	Thinking	00:00:01	
3	Data_Entry[1]	MASTER[03]		Exited		
4	Data_Entry[2]	MASTER[04]		Exited		
5	Data_Entry[3]	MASTER[05]		Exited		
6	Sales[1]	MASTER[06]		Exited		
7	Sales[2]	MASTER[07]		Exited		
8	Sales[3]	MASTER[08]		Exited		
9	Sales[4]	MASTER[09]		Exited		
10	Sales[5]	MASTER[10]		Exited		

In addition to the information displayed in the Compact user view, the Message user view contains the following information:

- ▶ **Message** – Text displayed from a running VU script. If the user executes a VU display library routine, the first 20 characters of its text appear here, and remain until they are overwritten by the next display routine.

Full User View

The Full user view contains complete information about all users. The following figure shows an example of a Full user view:

	Groups		Type	Script	Command	State	Time	Source		Cmd Count
	Schedule	Computer						File	Line	
2	Accounting[2]	MASTER[02]	VU	Calculate	sqlproc	Wait Shv	00:01:27	Calculate	37	4
3	Data_Entry[1]	MASTER[03]	VU	Delete Re	sqlprepar	Thinking	00:00:01	Delete Re	984	21
4	Data_Entry[2]	MASTER[04]	VU	Modify Re		Client Can	00:00:00	Modify Re	13	0
5	Data_Entry[3]	MASTER[05]	VU	Modify Re		Client Can	00:00:00	Modify Re	13	0
6	Sales[1]	MASTER[06]	VU	Read Rec		Client Can	00:00:00	Read Rec	13	0
7	Sales[2]	MASTER[07]	VU	Read Rec		Client Can	00:00:00	Read Rec	13	0
8	Sales[3]	MASTER[08]	VU	Read Rec	sqlproc	Waiting Is	00:00:00	Read Rec	71	13
9	Sales[4]	MASTER[09]	VU	Read Rec		Client Can	00:00:00	Read Rec	13	0
10	Sales[5]	MASTER[10]	VU	Read Rec	sqlprepar	Thinking	00:00:02	Read Rec	984	21

In addition to the information displayed in the Compact user view, the Full user view contains the following information:

- ▶ **Type** – Whether the user is a virtual user or a GUI user.
- ▶ **Command** – The VU emulation command that is executing. In this example, all users are executing the VU emulation command `sqlexec`.
- ▶ **Source** – Generally the same as the script. However, if a GUI script calls another script, or if a VU script contains an include file, the called script or the include file is displayed.
- ▶ **Cmd Count** – The number of VU emulation commands that have been executed in the current script. If this script is part of a loop, the next time LoadTest executes that line, the line number stays the same but the command count increases. Thus, the command count helps you distinguish between executions of the same command on different loop iterations.
- ▶ **Streak** – Succession of successes (S) or failures (F) in the entire schedule run. For example, S21 means that 21 emulation commands in a row have been successfully executed from the time the schedule began running.
- ▶ **Failure Rate** – The number of failures in the last ten emulation commands, the number of failures in the current script, and the number of failures overall.

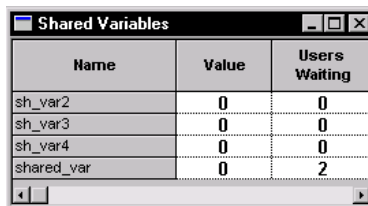
Displaying the Shared Variables View

The Shared Variables view lets you inspect the values of any shared variables that you have set in your schedule or script.

To display the Shared Variables view:

- ▶ Click **Monitor** → **Shared Variable**.

The following figure shows a Shared Variables view:



Name	Value	Users Waiting
sh_var2	0	0
sh_var3	0	0
sh_var4	0	0
shared_var	0	2

This view shows the name of each shared variable, the value of the variable, and the number of users waiting for the shared variable to reach a certain value.

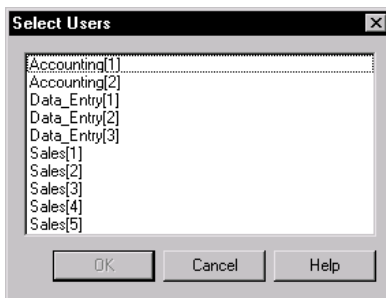
You can change the value of a shared variable from this view. For information about changing the value, see *Changing the Value of a Shared Variable* on page 8-31.

Displaying the Script View

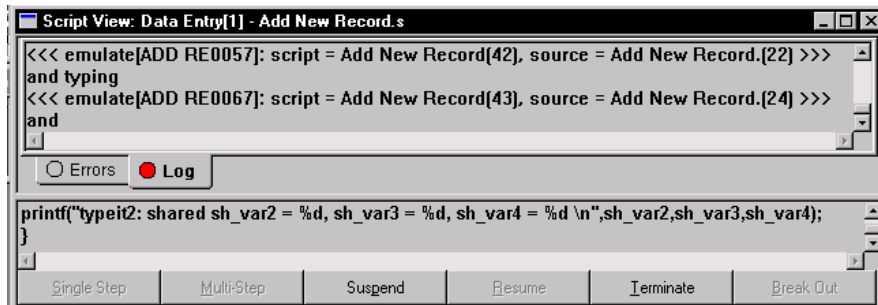
The Script view displays the line of code that a user is running. The Script view is useful if you want to watch the progress of a user through a script.

To display the Script view:

1. Click **Monitor** → **Script**. LoadTest displays a list of users that are running scripts:



2. Click the user whose progress you want to check. To select more than one user, hold down the CTRL key while clicking. The Script view appears:



The top half of the window shows two tabs: **Errors** and **Log**. These tabs display the last 100 lines of the User Log and User Error files. The top tab automatically toggles to the last message that was written.

The bottom half of the window shows the script. The script displays, line by line, what the user is doing.

NOTE: Even if you have set the log level to **All**, successful emulation commands are put into a buffer and are temporarily unavailable when you monitor a user in the Script view. This is to increase LoadTest's efficiency. Unsuccessful commands are displayed immediately; however, your log and error files will contain the complete results. For efficiency reasons, only the last 100 lines of the log and error files are displayed when monitoring the Script view.

Displaying the Sync Points View

The Sync Points view displays information about the synchronization points that you have set in the schedule or that you have included in a script. This view also lets you manually release users that are waiting on a synchronization point.

To display the Sync Points view:

- ▶ Click **M onitor** → **S ync P oints**. The Sync Points view appears:

Name	State	Time	Timeout	Users			Delay	
				Arrived	to Sync	Late	Min	Max
stress t	Empty	00:00:15	00:05:00	0	150	0	00:00:00	00:00:00

The Sync Points view displays the following information:

- ▶ **Name** – The name of the synchronization point.
- ▶ **State** – The state the synchronization point is in. The states can be:
 - **Empty** – No users have arrived at the synchronization point.
 - **Waiting** – At least one user has arrived at the synchronization point, but not all of the users have arrived.
 - **Released** – The users are released from the synchronization point. This column also indicates whether the users were released because they all reached the synchronization point (**Normal**), whether you have released the users manually (**M onitor**), or whether the synchronization points have timed out (**T imeout**).

- ▶ **Time** – The time the synchronization point has been in the current state.
- ▶ **Timeout** – The timeout period that you set in the schedule, or **Infinite**, if you did not set a timeout period.
- ▶ **Users** – The number of users that have reached the synchronization point:
 - **Arrived** – The number of users that have arrived at the synchronization point before it was released.
 - **To sync** – The number of users that must arrive to release the synchronization point.
 - **Late** – The number of users that arrived at the synchronization point after it was released.
- ▶ **Delay** – If the release time is together, the release delay that you have set in the schedule. If the release type is staggered, the minimum and maximum release times.

Displaying the Users Waiting on a Synchronization Point

To display the users waiting on a synchronization point:

1. Click **Monitor** → **Sync Points**. The Sync Points view appears.
2. Right-click the name of the synchronization point, and then click **See Users**.

Releasing a Synchronization Point

You might decide to release a synchronization point, even though the required number of users has not yet been reached. Subsequent users that arrive at the synchronization point are not held. However, if you have set a restart time and a maximum time in the schedule, the users will be delayed. So, for example, if you release a synchronization point but have set a restart time of 1 second and a maximum time of 4 seconds, each user who reaches that synchronization point is delayed from 1 to 4 seconds.

To release a synchronization point:

1. Click **Monitor** → **Sync Points**. The Sync Points view appears.
2. Right-click the name of the synchronization point, and click **Release**.
3. A confirmation message appears, asking you to confirm the release. Click **Yes** or **No**.

Displaying the Computer View

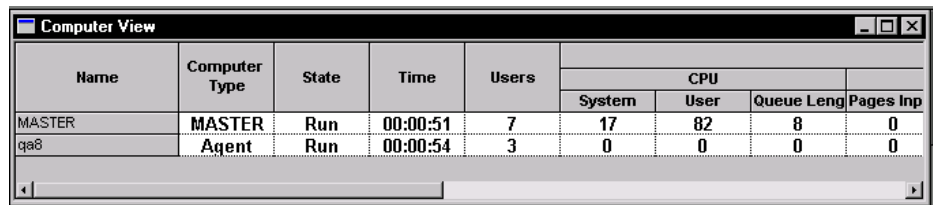
Use the Computer view to check the computer resources used during a schedule run, as well as the status of the Master and Agent computers at the beginning and end of a run.

Viewing Resource Usage During a Run

To check computer resources used during a schedule run:

- ▶ Click **Monitor** → **Computers**. The Computer view appears.

LoadTest displays the computer resources used for each Master and Agent computer in the run.



Name	Computer Type	State	Time	Users	CPU			
					System	User	Queue Leng	Pages Inp
MASTER	MASTER	Run	00:00:51	7	17	82	8	0
qas	Agent	Run	00:00:54	3	0	0	0	0

The Computer view displays the following information:

- ▶ **Name** – The name of the computer that you specified in the schedule. Master is the local computer.
- ▶ **Computer Type** – The type of computer: either Master, Agent, or Server.
- ▶ **State** – The state the computer is in. When you display the Computer view manually, it is generally in the run state.
- ▶ **Time** – The time the computer has been in the current state. The time is displayed in *hours:minutes:seconds*.
- ▶ **Users** – The total number of users assigned to run on the computer.
- ▶ **CPU System** – The percent of CPU cycles servicing the operating system.
- ▶ **CPU User** – The percent of CPU cycles servicing user processes.
- ▶ **CPU Queue Length** – The number of processes or threads that are ready to run but have to wait in a queue.

This number should be 0 or very small unless the **CPU System** and **CPU User** percentages are close to 100.

- ▶ **Memory Pages Input/Sec.** – The number of pages per second that are read into memory.
- ▶ **Memory Pages Output/Sec.** – The number of pages per second that are swapped onto disk. This number should be considerably smaller than the memory pages that are input.

Together, these numbers can indicate memory bottlenecks.

- ▶ **Memory % Used** – The percentage of memory used.
- ▶ **Disk Transfers/Sec.** – This shows the disk access speed (seek, rotation, and transfer time) for up to four disks. If one disk is much slower than the others, it might be fragmented. If the transfer rate peaks when the response time is down, this also could indicate a problem with your disk.

NOTE: If you are monitoring resources for a computer that runs HP-UX or AIX, the **Disk Transfers/Sec.** column will always show **n/a**. This is because HP-UX and AIX are unable to supply LoadTest with disk transfer information. However, all other columns will be correct.

- ▶ **% Disk Used** – The percent of used disk space on the monitored disk.
- ▶ **Delay** – This lets you gauge the general state of your network. At regular intervals, LoadTest sends a small ICMP packet to the remote computer and times that request, in ms. This time does not include any service time used by a user-level process on the remote computer. The time should stay relatively consistent and quite small. A large number or a number that varies widely might indicate network problems.
- ▶ **Service Time** – This lets you gauge the general state of your network. At regular intervals, LoadTest sends a small TCP packet to the specified computer and times that request, in ms. This time includes the service time for a user-level process on the remote computer to reply to the packet. The time should stay relatively consistent and quite small. A large number or a number that varies widely might indicate network problems.

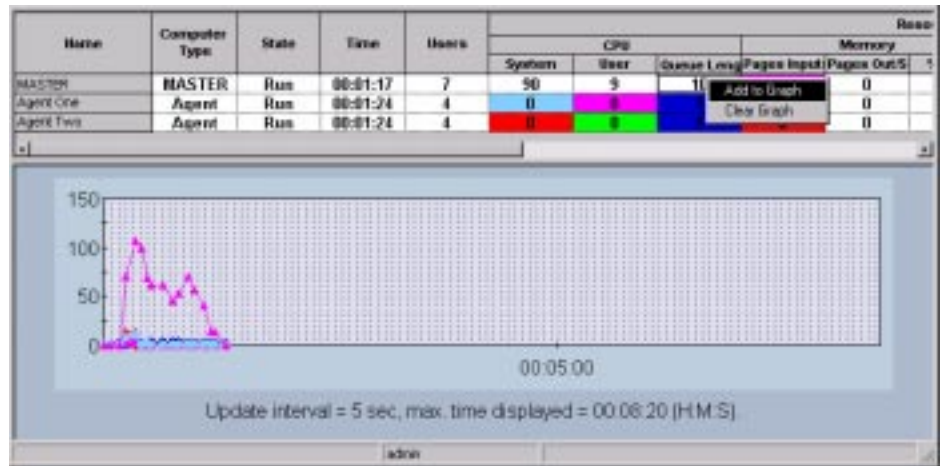
NOTE: In addition to monitoring your computer resources, you can report on them. The Response report lets you compare your response time with your computer resource usage.

Graphing Resource Usage During a Run

You can graph the resources that your computer uses during a schedule run. To graph computer resources:

1. Click **Monitor** → **Computers**. The Computer view appears.
2. Right-click a cell in the **Resources Used** or **Network** columns, and click **Add to Graph**.

The following figure shows **Memory Pages Input** being added to the graph:



To change the color of an item that is in the graph:

- ▶ Right-click on the corresponding cell and click **Toggle Text Color** or **Change Text Color**.

To remove an item that is in the graph:

- ▶ Right-click on the corresponding cell and click **Remove from Graph**.

To remove all items that are in the graph:

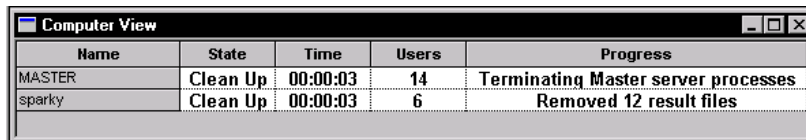
- ▶ Right-click on any cell in the graph and click **Clear Graph**.

Viewing Computers at the Start or End of a Run

The Computer view appears automatically when Agent computers start up. When all Agents are up and running, the Computer view closes. When Agents begin shutting down, the Computer view reappears automatically so you can watch the cleanup activities such as transferring files to the Master computer.

The Computer view includes **Progress** messages, which indicate when the computer is creating or initializing processes, transferring files, terminating users, and so on.

The following figure shows a Computer view at the end of a run:



Name	State	Time	Users	Progress
MASTER	Clean Up	00:00:03	14	Terminating Master server processes
sparky	Clean Up	00:00:03	6	Removed 12 result files

The Computer view displays the following information:

- ▶ **Name** – The name of the computer that you specified in the schedule. Master is the local computer.
- ▶ **State** – The state the computer is in. It can be one of the following:
 - **Not Connected** – The Master computer has not yet connected to the Agent computer.
 - **Initializing** – The computer is being initialized, or is transferring compiled scripts and datapools that are out of date.
 - **Run** – The computer is running users.
 - **Termination** – The computer is in termination mode, waiting for users to exit.
 - **Clean Up** – The computer is cleaning up before exiting. This includes transfer and removal of the result files.
 - **Exit** – The computer has exited.
- ▶ **Time** – The time the computer has been in the current state. The time is displayed in *hours:minutes:seconds*.
- ▶ **Users** – The total number of users assigned to run on the computer.

Displaying the Transactor View

The Transactor view shows the status of the transactors that you inserted into the schedule.

To display a Transactor view:

1. Click **Monitor** → **Transactors**.

The Transactor view contains the following information:

- ▶ **Name** – The name that you gave the transactor when you inserted it in the schedule.
- ▶ **Type** – Whether the transaction is Independent or Coordinated.
- ▶ **State** – The state that the transactor is in. It can be one of the following:
 - **Not Started** – An initial state, when the transactor has not run any users.
 - **Arriving** – This state pertains to Coordinated transactors only. At least one user has arrived at the sync point, but not all users have arrived.
 - **Active** – The transactor is running at least one user.
 - **Inactive** – The transactor is not running any users.
- ▶ **Users** – The number of users in the **Arriving** or **Active** states.
- ▶ **Start Time** – The time the transactor first entered the **Active** state.
- ▶ **Active Time** – The total amount of time the transactor has been in the **Active** state.
- ▶ **Transactions** – The number of transactions that are scheduled by the transactor, but not necessarily completed by the user.
- ▶ **Target Rate** – The rate that you set for the transactor when you inserted it in the schedule.
- ▶ **Actual Rate** – The rate that the transactions are actually running. LoadTest calculates this rate by dividing **Transactions** by the **Active Time**.

The **Target Rate** specifies the number of *started* transactors, but the **Actual Rate** calculates the number of *completed* transactions. Because the transactions take some time to complete, the **Actual Rate** will approach, but will not reach, the **Target Rate**. However, over time and enough transactions, the **Actual Rate** should become close to the **Target Rate**.

- ▶ **% Late** – The percent of transactors that were unable to begin running at the desired time.

For coordinated transactors, this usually means that not enough users are available to run the transactors. You may want to run the schedule again with more users.

For independent transactors, this usually means that the time it takes to run one transaction is longer than the time between two transactions.

If too many transactors are late, then the target transaction rate will not be maintained or the transactor will not accurately simulate peaks in the transaction rate.

Displaying the Group Views

The Group views show the status of all user groups that you defined in the schedule. Both Group views show the same information, but the Schedule view shows the information by user group, and the Computer view shows the information by computer.

To display a Group view:

1. Click **Monitor** → **Groups**.
2. Select one of the groups:
 - **Schedule** – A list of the user groups in a schedule. The following figure shows the Schedule view:

Group	User					
	Type	Total	Active	Suspended	Abnormal	Normal
Accounting	VU	2	2	0	0	0
Data_Entry	VU	3	0	0	3	0
Sales	VU	5	0	0	5	0

- **Computer** – A list of the user groups assigned to the same computer. The following figure shows the Computer view:

Group	User					
	Type	Total	Active	Suspended	Abnormal	Normal
MASTER	VU	10	10	0	0	0

The Group views contain the following information:

- ▶ **Type** – The type of users in the group, either VU, GUI, or Mixed. A Mixed user group appears in a Computer view when a computer runs both GUI and virtual user groups.
- ▶ **Total** – The total number of users in the group.
- ▶ **Active** – The number of users in the group currently running.
- ▶ **Suspended** – The number of virtual users in the group that are suspended.
- ▶ **Abnormal** – The number of users that terminated without completing all of their assigned tasks.
- ▶ **Normal** – The number of users that completed their tasks successfully.

Displaying the Users in a Group

To display the users in the groups:

1. Click the user in the left column.
2. Click the right mouse button to display the shortcut menu.
3. Click **See users**.

Filtering and Sorting Views

This section discusses how to customize a view. For example, you can sort users in various ways, or you can filter users and groups so that only certain information is displayed.

Sorting the Users Displayed in a User View

While displaying a user view, you may want to see the users in a particular order. For example, you can sort the users alphabetically, or you can sort them in the order in which they started.

To change the order in which the users are displayed:

1. Click **Monitor** → **User**.
2. Select one of the user views:
 - **Full** – Contains complete information about all users.
 - **Compact** – Contains summary information about all users.
 - **Results** – Contains information about the success and failure rate of each VU emulation command.

Monitoring Schedules

- **Source** – Displays the line number and the name of the source file being executed.
 - **Message** – Similar to the Compact user view, but also displays the first 20 letters of text from the VU display library routine.
3. Select a column under the **Schedule** or **Computer** heading:

	Groups		Script	State	Time
	Schedule	Computer			
1	Accounting[1]	MASTER[01]	Calculate	Get Task	02:55:53
2	Accounting[2]	MASTER[02]	Calculate	Get Task	02:55:53
3	Data Entry[1]	MASTER[03]		Exited	
4	Data Entry[2]	MASTER[04]		Exited	
5	Data Entry[3]	MASTER[05]		Exited	
6	Sales[1]	MASTER[06]		Exited	
7	Sales[2]	MASTER[07]		Exited	
8	Sales[3]	MASTER[08]		Exited	
9	Sales[4]	MASTER[09]		Exited	
10	Sales[5]	MASTER[10]		Exited	

4. Click the right mouse button. The shortcut menu appears:

	Groups		Script	State	Time
	Schedule	Computer			
1	Accounting[1]	MASTER[01]	Calculate	Get Task	03:32:51
2	Accounting[2]	MASTER[02]	Calculate	Get Task	03:32:51
3	Data Entry[1]	MASTER[03]		Exited	
4	Data Entry[2]	MASTER[04]		Exited	
5	Data Entry[3]	MASTER[05]		Exited	
6	Sales[1]	MASTER[06]		Exited	
7	Sales[2]	MASTER[07]		Exited	
8	Sales[3]	MASTER[08]		Exited	
9	Sales[4]	MASTER[09]		Exited	
10	Sales[5]	MASTER[10]		Exited	

5. Select the order to sort by:
- **Schedule Order** – The order in which the user group appears in the schedule.
 - **Execution Order** – The order in which the users are started.
 - **Schedule Groups** – Alphabetical listing of schedule groups.
 - **Computer Groups** – Alphabetical listing of computer groups.

NOTE: The current sort order is unavailable. In this example, Execution Order is unavailable because it is the sort order being used.

Filtering a User View

When you display a user view, you can filter users so that only certain users appear. This is useful if your schedule contains many users and you want to focus on the progress of a few of these users.

Including and Excluding Selected Users

For example, you can select certain users and include or exclude them from a view. To change which users are displayed in a user view:

1. Click **Monitor** → **User**.
2. Select the user view you want to filter.
3. Select a single row or consecutive rows by clicking on the number in the first row.
4. Click the right mouse button to display the shortcut menu.
5. Click **Filter users**.



- If you click **Include**, only the users that you selected are displayed.
- If you click **Exclude**, all users except those that you selected are displayed.

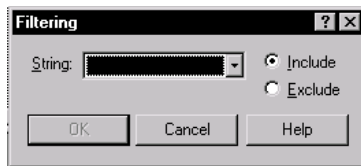
Filtering a User by Value

You can filter a user on any value that will stay constant during the run, such as the name of its group, the type of script it is running, or the name of the computer it is running on.

For example, you might be running a test with 200 virtual users in the Accounting user group, 300 virtual users in the Data Entry user group, and 500 virtual users in the Sales user group. You want to see only the Data Entry users. You can filter the group so that LoadTest displays only the group with the “Data Entry” value.

To filter a user by value:

1. Click **Monitor** → **User**.
2. Select the user view that you want to filter.
3. Click the **Schedule**, **Group** or **Type** heading, and click the right mouse button in any cell in the view. Note that even if you click a cell in another column, the column whose heading you clicked remains highlighted. You can filter only on the values on the highlighted column.
4. Click **Filter users** → **By Value**.



5. Select a string in the **String** box, and click either **Include** or **Exclude**.
6. Click **OK**.

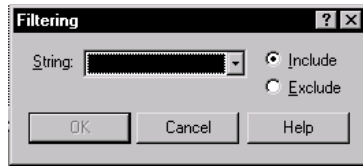
Filtering a Group View

If the Group view displays many columns, you can filter out some columns to provide more room to view the columns that you want to see. You can filter a group on any value that will stay constant during the run, such as the name of the user group or the type of script.

To filter a Group view:

1. Click **Monitor** → **Groups**.
2. Select the group view that you want to filter.
3. Click the **Group** or **Type** heading, and click the right mouse button to display the shortcut menu.

4. Select **Filter by Value** from the shortcut menu.



5. Select a string in the **String** box, and click either **Include** or **Exclude**.
6. Click **OK**.

Restoring the Default Views

If you have zoomed in on a histogram bar, filtered a view, or changed the widths of a column in a view, you may want to restore the bar or views to their original settings.

To restore a view to its original setting:

1. Display the view that you want to restore.
2. Click **View** → **Reset**.

Changing the Value of a Shared Variable

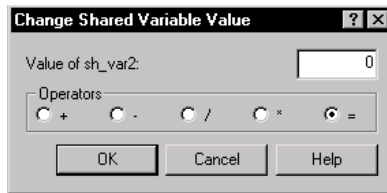
You can change the value of a shared variable when you are monitoring a schedule.

To change the value of a shared variable:

1. Click **Monitor** → **Shared Variable**.

Name	Value	Users Waiting
sh_var2	0	0
sh_var3	0	0
sh_var4	0	0
shared_var	0	2

2. Double-click the variable name, or click the right mouse button and select **Change Value** from the shortcut menu.



3. If the shared variable is read only, type a new value in the **Value of** box.

If the shared variable is being dynamically updated, however, you cannot simply type in a new value. By the time you read the value, determine the new value, and change it, a virtual user may have modified the value. If this occurs, your change is lost. Instead:

- Type an operand in the **Value of** box.
- Under **Operators**, choose an operator. If you choose the subtract (-) or divisor (/) operators, the order for operations is:

existing value - new value

existing value / new value

For example, assume the shared variable has a current value of 6. If you type 4 in the **Value of** box and click the - operator, the new value of the shared variable is 2, because $6 - 4 = 2$.

4. Click **OK**.

Displaying the Virtual Users Waiting on a Shared Variable

If your scripts contain shared variables, you can see the virtual users waiting on each shared variable.

To display the virtual users waiting on a shared variable:

1. Double-click the variable name, or click the right mouse button.
2. Click **See Users** from the shortcut menu.

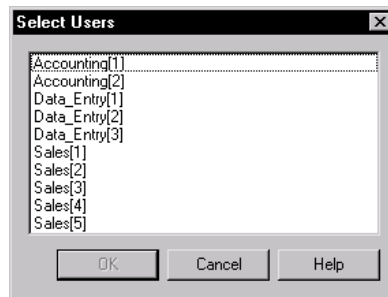
The virtual users are displayed in a Compact user view.

Debugging a VU Script

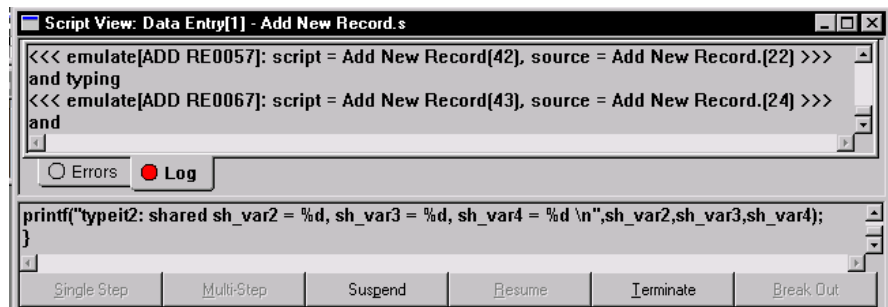
You may encounter problems when you are monitoring a schedule. LoadTest provides you with tools that enable you to debug a VU script. When you debug a script, it is a good idea to run the schedule with just one user, correct the script, and then run the schedule as usual.

To debug a script:

1. Click **M onitor** → **Script**. LoadTest displays a list of users that are running scripts.



2. Click the virtual user that is running the script you want to display, and then click **OK**.
3. The Script view appears:



Select the choice that you want:

- **Single Step** – Steps through a VU script one emulation command at a time, allowing you to see what happens at each command. To use this choice, first suspend a user. This is useful for pinpointing problems.

Monitoring Schedules

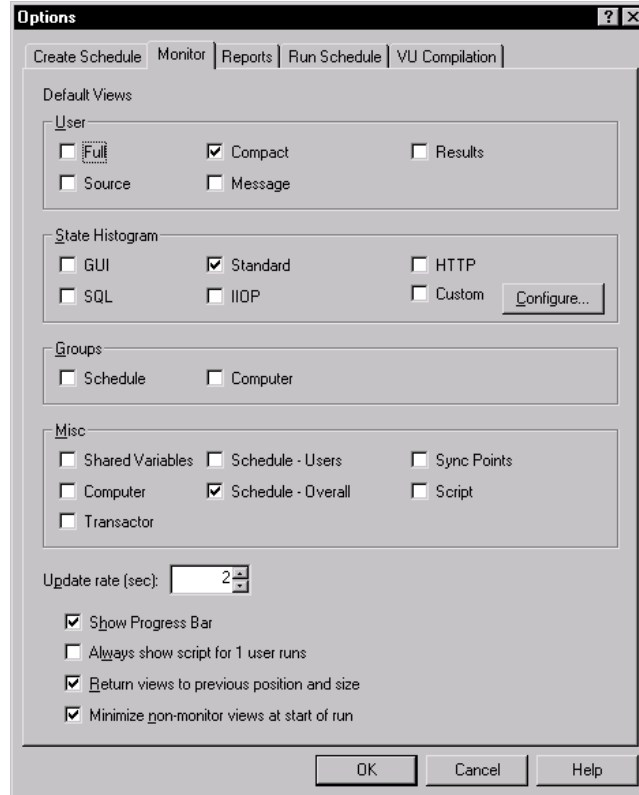
- **Multi-step** – Steps through a VU script multiple emulation commands at a time. To use this choice, first suspend the user. Then you can select a number of commands to execute at a time.
- **Suspend** – Suspends a virtual user at the beginning of the next emulation command.
- **Resume** – Allows a suspended virtual user to resume its progress through a script.
- **Terminate** – Ends the virtual user's execution of a script.
- **Break Out** – Moves a virtual user out of the following three states:
 - Waiting on a shared variable
 - Waiting on a response
 - VU delay function

Changing Monitor Defaults

When you monitor a schedule, you can set which views are displayed automatically, how often the views are refreshed, and whether toolbars are displayed automatically when you run a schedule. You can even configure the Custom histogram, and change its colors, as described in the next section.

To change monitoring defaults:

1. Click **Tools** → **Options**.
2. Click the **Monitor** tab.



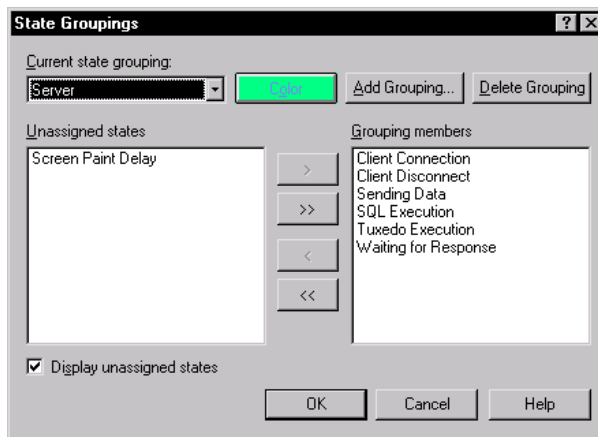
3. Under **Default views**, select the check boxes for the views and toolbars that you want displayed when you run a schedule.
4. Select **Always show script for 1 user runs** if you always want to see the Script view for a one-user run. This is typically used to test a schedule and debug it.
5. Click **OK**.

Configuring Custom Histograms

By default, the Custom histogram is identical to the Standard histogram. However, unlike the other histograms, you can configure the Custom histogram. You can configure the groups, create new groups, and change the colors that designate a group.

To configure the groups:

1. Click **Tools** → **Options**. The Options dialog box appears.
2. Under **State Histogram**, select **Custom**, and then click the **Configure** button. The State Groupings dialog box appears:



From this box, you can assign states to and remove states from a group, add an entire group and assign states to it, or delete an entire group. The following sections discuss how to do this.

3. Select the **Display unassigned states** check box to display unassigned states in an Other bar of the Custom histogram. If you clear this box, unassigned states are ignored, and the Other bar is not displayed.

Assigning States to a Group

To assign a state to a Custom Histogram group:

1. In the State Groupings dialog box, select a group from the **Current State Grouping** box.
2. In the **Unassigned States** box, click the state you want to add to the group. You can assign a state to only one group.

3. Click the right arrow.
4. Click **OK**.

The state is assigned to the group.

Removing States from a Group

To remove a state from a Custom Histogram group:

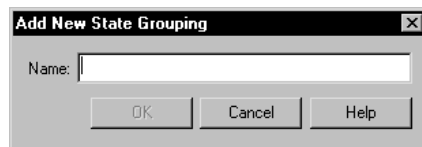
1. In the State Groupings dialog box, select a group from the **Current State Grouping** box.
2. In the **Grouping members** box, click the state you want to remove from the group.
3. Click the left arrow.
4. Click **OK**.

The state is removed from the group.

Adding a Group

To add an entire group to the Custom histogram:

1. In the State Groupings dialog box, click the **Add Grouping** button.



2. Type the name of the new group.
3. Click **OK**.
4. Click the states you want to add to the group. To select more than one state, hold down the CTRL key while you click.
5. Click the single right-arrow key. The states are added to the **Grouping members** box.
6. To assign a color to the state group, click the **Color** button, click a color, and then click **OK**.
7. Click **OK**.

The new group is added to the Custom histogram.

Deleting a Group

To delete a group from the Custom histogram:

1. In the State Groupings dialog box, select the group you want to delete from the **Current State Grouping** box.
2. Click the **Delete Grouping** button.
3. Click **OK**.

The group is deleted, and all states that were in the group are now unassigned.

Controlling the Schedule During a Run

LoadTest provides a variety of ways to help you control a schedule while it is running. For example, you can suspend a schedule to change settings or examine its progress.

Suspending and Resuming Virtual Users in a Schedule

While a schedule is running you can suspend and resume all virtual users in the schedule, or you can suspend and resume individual virtual users. This is useful if a problem occurs during the run and you want to investigate it.

To suspend and resume all virtual users in the schedule:

- ▶ Click **Run** → **Suspend** or **Run** → **Resume**.

To suspend or resume individual virtual users:

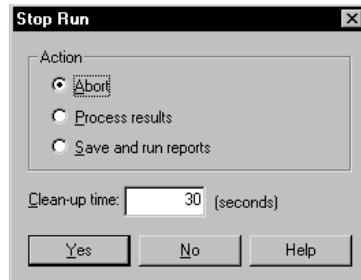
1. Click **Monitor** → **Users**.
2. Select the user row that you want to suspend.
3. Click the right mouse button.
4. Select **Suspend** or **Resume**.

Stopping a Schedule

You can stop the execution of a schedule. This is useful if there is a serious problem and you do not want to wait for the test to finish.

To stop a run:

1. Click **Run** → **Stop**.



2. Under **Action**, click the option that you want:
 - **Abort** – Stops the run and does not save any results. Click this option if you do not plan to run any reports or look at any User Error, User Output, or User Log files through the LogViewer.
 - **Process Results** – Stops the run but saves the results so that you can run reports, and look at any User Error, User Output, or User Log files through the LogViewer.
 - **Save and Run Reports** – Stops the run, saves the results, and produces reports, just as if your run completed normally.
 - **Clean-up time** – The amount of time allowed from the time you request termination until LoadTest forces the termination of the run.

NOTE: When you abort a large multiuser schedule that includes multiprocessor Master or Agent computers, choose a Clean-up time of 60 seconds or more to allow users (sqa7svui processes) time to exit on their own. The default **Clean-up time** of 1 second often causes the Master to terminate many user processes at once, and can result in leftover sqa7svui processes. Although not harmful, they clutter the process table. They can be killed individually using Task Manager, or all at once by logging off.

Monitoring Schedules

▶▶▶ C H A P T E R 9

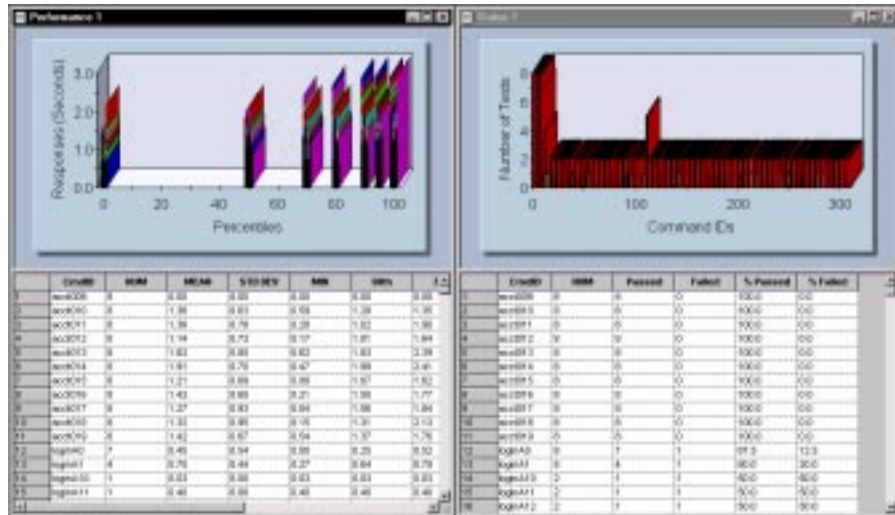
Analyzing Results

This chapter explains how to use LoadTest reports to analyze performance data. It includes the following topics:

- ▶ About LoadTest reports
- ▶ Running a report and viewing log files
- ▶ Printing a report
- ▶ Printing report output
- ▶ Copying a report or its output within LoadTest
- ▶ Renaming a report or report output
- ▶ Deleting a report or report output
- ▶ Exporting report output
- ▶ Comparing the output of Performance reports
- ▶ Customizing reports
- ▶ Changing report defaults
- ▶ Types of reports

About LoadTest Reports

If your schedule has completed successfully, LoadTest automatically runs Status and Performance reports against the data in the log and displays the **report output**. The following figure shows output from a Status and a Performance report:



After you have examined the output from these reports, you can save it or delete it.

If you save the output, LoadTest gives it the default names of Status 1 and Performance 1, and saves it under the logs in the repository. To view the output again, click **File** → **Open** → **Report Output**, and select Performance 1 and Status 1.

If you delete the output, you can re-create it by running the Status and Performance reports against the same log. Simply click **Run** → **Report** → **Performance** (or **Status**), select the build and log folder that contain the log, and then select the log.

In addition to the Status and Performance reports, LoadTest provides various types of reports designed to analyze the results of a schedule run. For example, you can determine how long it took for a virtual user to execute a command, and how response times varied with different schedule runs. You can also define new reports.

The following table summarizes the types of LoadTest reports:

Use this report	To do this	For information, see
Performance	<p>Display the response times, and calculate the mean, standard deviation, and percentiles for each response time in the schedule run.</p> <p>The output groups responses by command ID and shows only responses that passed. In contrast, Response report output shows each command ID individually and shows passed and failed responses.</p>	<i>Performance</i> on page 9-44
Compare	<p>Compare the response times measured by Performance reports. After you have generated output from several Performance reports, use the Compare report to compare a specific field.</p>	<i>Compare</i> on page 9-40
Response	<p>Display individual response times and whether a response has passed or failed. This report is useful for looking at data points for individual responses as well as trends in the data.</p> <p>The output shows each command ID individually and the status of the response. In contrast, Performance report output groups responses by command ID and shows only passed responses.</p> <p>You can right-click on the report output, select a computer that was in the run, and graph the resource monitoring statistics for that computer. These are the same statistics that you display when you monitor a schedule.</p>	<i>Response</i> on page 9-46
Status	<p>Obtain a quick summary of which commands passed or failed. The output displays the status of all VU emulation commands and SQABasic timer commands. If you have failures, you may want to run the Analog report.</p>	<i>Status</i> on page 9-48
Analog	<p>Examine errors in your run. The output shows the “conversation” between the virtual user and the server. If you access a database, the output shows the database errors, as well as the number of rows that you expect to receive versus the number returned from the server. If you need further information, run a Trace report.</p>	<i>Analog</i> on page 9-38

(Continued)

Use this report	To do this	For information, see
Trace	Examine any failures in detail. The output formats raw data from the logs, without performing statistical analysis. It provides information including the timestamps from each VU emulation command and SQABasic timer command, and the counts of data sent and received.	<i>Trace</i> on page 9-49
Usage	View cumulative response time and summary statistics, as well as throughput information for VU emulation commands only.	<i>Usage</i> on page 9-52

Running a Report and Viewing Log Files

LoadTest automatically runs the Performance and Status reports at the end of a successful schedule run. However, you may also want to view the log files, which are the “raw” result files before you run reports on them. Or you may want to run other reports. This section describes how to view log files and how to run different reports.

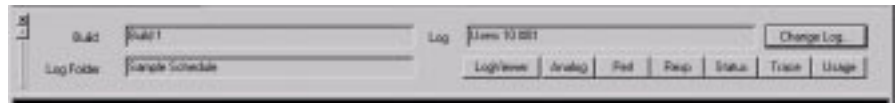
Viewing the Log Files

The Rational LogViewer shows log files for each virtual user, as well as a log file for the entire schedule run. The log files are:

- ▶ A Schedule Log file, which contains the compilation and run messages that you saw when you ran the schedule
- ▶ A User Error file, which lists warnings and errors that occurred during the schedule run
- ▶ A User Output file, which lists messages that the VU script wrote to standard output
- ▶ A User Log file, which lists the data for each VU emulation command executed, as well as any messages coded in the script with the VU `log_msg` library routine

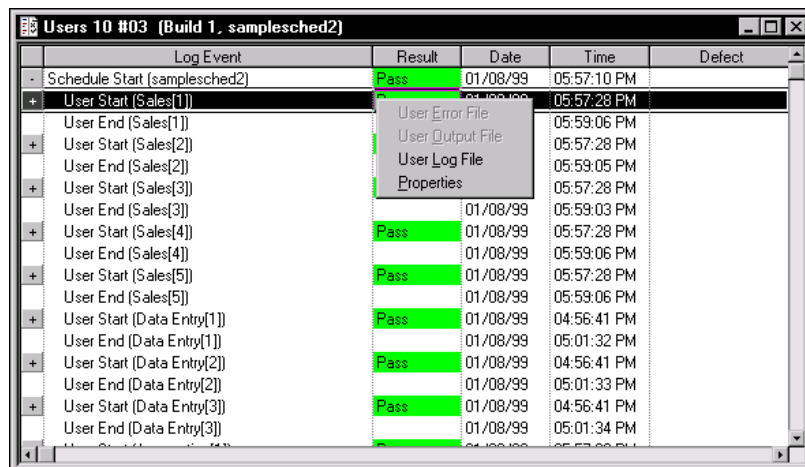
To display these log files:

1. Click **View** → **Report Bar**. The report bar appears:



2. LoadTest displays the log of the last schedule you ran. If necessary, click **Change Log** to view the log files from a previous schedule run.
3. Click the **LogViewer** button. The LogViewer appears.
4. To display the Schedule Log file, right-click **Schedule Start** (near the top of the **Log Event** column), and click **Schedule Log File**.

To display a User Log file, right-click a user group, and select a file:



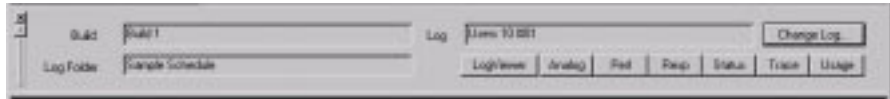
For more information about the LogViewer and how it relates to GUI users, see the *Using Rational Robot* manual.

Running a Report from the Report Bar

The quickest way to run a report is to click its name on the report bar.

To run a report from the report bar:

1. Click **View** → **Report Bar**. The report bar appears:



2. LoadTest displays the log of the last schedule you ran. If necessary, click **Change Log** to report on a log from a previous schedule run.
3. Click one of the report buttons, such as **Perf** or **Status**, to run the report.

LoadTest displays the report output. After you have examined the output, you can save it or delete it.

To run a different report on the log, click another report button.

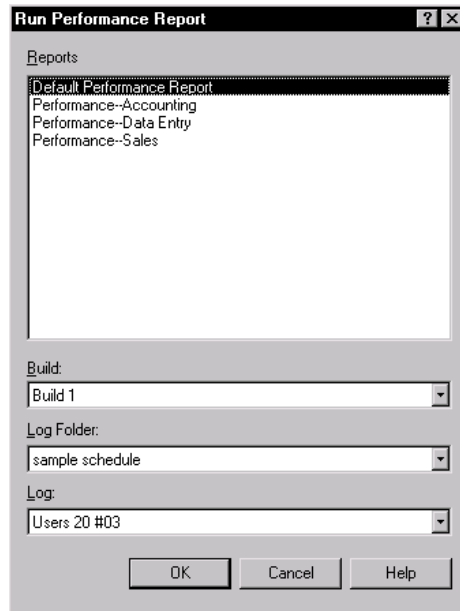
NOTE: You can customize the report bar by populating it with your own reports. Click **Tools** → **Options**, and then click the **Reports** tab. For more information, see *Changing the Reports that Run from the Report Bar* on page 9-37.

Running a Report from the Menu Bar

Although LoadTest lets you run reports quickly from the report bar, you can run only one report of each type against a log. You may want to run a number of reports. For example, assume you have defined some new Performance reports, and you want to run each report. You run these reports from the menu bar.

To run a report from the menu bar:

1. Click **Run** → **Report**, and select the type of report to run. A dialog box similar to the following appears:



2. From the **Reports** list, select the Performance report that you want to run.
3. LoadTest displays the build, log folder, and log of your most recent schedule run. You can change this information, if necessary.
4. Click **OK**.

LoadTest runs the report against the log and displays the report output. After you have examined the report output, you can save it or delete it.

Printing a Report

To print a report:

1. At the menu bar, click **File** → **Open**, and then select a report.
2. Click **File** → **Print**.
3. If necessary, modify the Properties dialog box for the printer that you have chosen.

Printing Report Output

To print report output:

1. Click **File** → **Open** → **Report Output**, and double-click a report output.
2. LoadTest displays the report output. Optionally, to add a header, click **View** → **Settings**, and then click the **Edit Graph Labels** tab. For more information, see *Changing a Graph's Labels* on page 9-33.
3. Click **File** → **Print**.
4. If necessary, modify the Properties dialog box for the printer that you have chosen.

Copying Report Output to the Clipboard

LoadTest displays Compare, Performance, Response, and Status report output in graphs and tables. You can copy the table portion onto the clipboard, and then paste it into another application such as Excel, Word, or Paint.

To copy a table into another application:

1. Click **File** → **Open** → **Report Output**, and select the rows that you want to copy.
2. Click **Edit** → **Copy**. LoadTest copies the table to the clipboard.
3. Open the application that you want to paste the table into.
4. Click the application's **Paste** command. LoadTest pastes the table into the application.

Copying a Report or Its Output within LoadTest

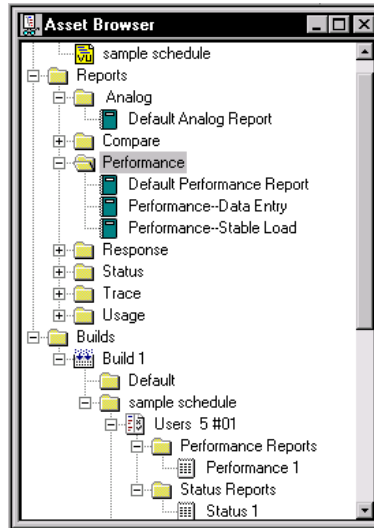
LoadTest lets you copy reports and report output.

Copying a report is useful if, for example, you have defined a rather complex report and you want to modify one option. Although you can define another report from scratch, it is much easier—and safer—to copy the report and then change that option. With this method, you can be sure that you have modified only that option.

Copying graphical report output is useful if you want to change the report output settings. For example, you may want to change the format of the output graph from bar to stack. Or you may want to filter the output. By copying the output and then changing the settings of the copy, you can preserve both the original and the changed output.

To copy a report or report output:

1. Click **View** → **Asset Browser**.

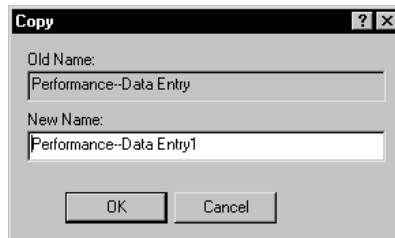


2. Select the item that you want to copy. The reports are located in the Reports folder. The report output is in the Builds folder.

In this example, the Asset Browser displays:

- Four reports—the Default Analog Report, the Default Performance report, Performance Data Entry, and Performance--Stable Load.
- Two report outputs—Performance 1 and Status 1.

3. Click **Edit** → **Copy**.



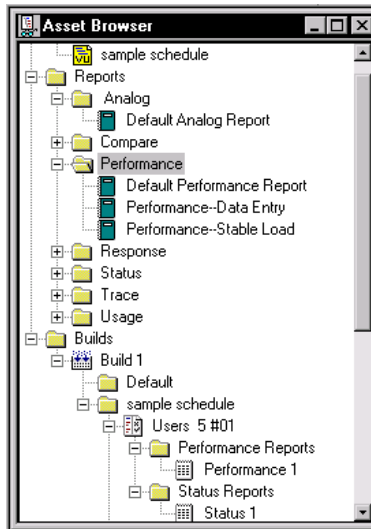
4. Type a new name for the copy. (Otherwise, LoadTest adds a 1 to the original name, as shown in the dialog box.) Then, click **OK**.

Renaming a Report or Report Output

After you have defined a number of reports, you might want to rename them. For example, you might adopt a new naming convention for reports, or you might want to standardize your naming conventions.

To rename a report or report output:

1. Click **View** → **Asset Browser**.



2. Select the item that you want to rename. The reports are located in the Reports folder. The report output is in the Builds folder.

In this example, the Asset Browser displays:

- Four reports—the Default Analog Report, the Default Performance report, Performance Data Entry, and Performance--Stable Load.
- Two report outputs—Performance 1 and Status 1.

3. Click **Edit** → **Rename**.

Deleting a Report or Report Output

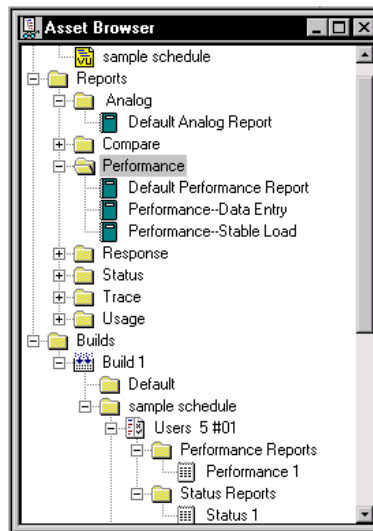
After you have defined a number of reports, you may find that some of them are no longer useful. You can delete both reports that you have defined and the default reports, which come with LoadTest.

Be careful about deleting default reports, however. The default reports affect all projects in the repository. Deleting a default report deletes it from other projects, as well as from your own project. Therefore, before you delete a default report, make sure that people in other projects, as well as people in your project, do not use the report.

LoadTest also lets you delete report output. This is useful if you run reports frequently and accumulate report output that you no longer need.

To delete a report or its output:

1. Click **View** → **Asset Browser**.



2. Select the item that you want to delete. The reports are located in the Reports folder. The report output is in the Builds folder.

In this example, the Asset Browser displays:

- Four reports—the Default Analog Report, the Default Performance report, Performance Data Entry, and Performance--Stable Load.
- Two report outputs—Performance 1 and Status 1.

3. Click **Edit** → **Delete**.
4. LoadTest asks you to confirm your deletion. Click **Yes** or **No**.

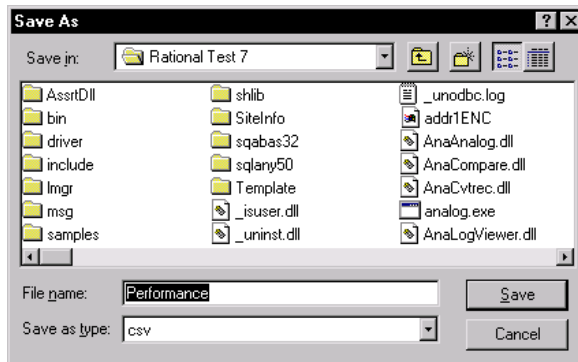
NOTE: To restore a default report that you have deleted, click **Tools** → **Options**, click the **Reports** tab, and then click the **Restore Defaults** button.

Exporting Report Output

The Performance, Status, Compare, and Response reports display data graphically. You can export this graphic data to a .csv file for further processing.

To export report output:

1. Click **File** → **Open** → **Report Output**, and then double-click a Performance, Status, Compare, or Response report output.
2. Click **File** → **Export to File**.



3. Select a folder. The default folder is the Rational Test 7 folder.
4. Enter a file name for the exported data. The default file name is the type of report.
5. Click **Save**.

Comparing the Output of Performance Reports

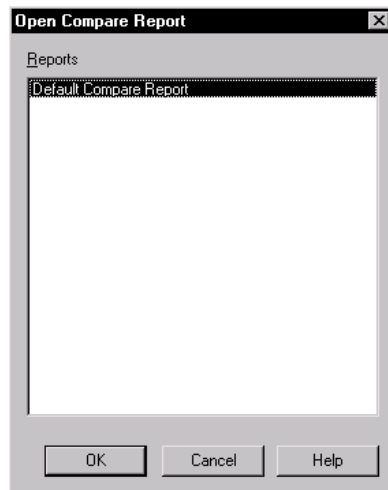
When you run schedules with a different number of users or with other different options, you often want to compare the output. You can run two Performance reports and visually scan the two Performance report outputs. However, a much more precise method is to run a Compare report on the two Performance report outputs.

Running a Compare report lets you focus on one column of a Performance report and compare it across different schedule runs.

A Compare report can compare the output of up to seven Performance reports.

To run a Compare report:

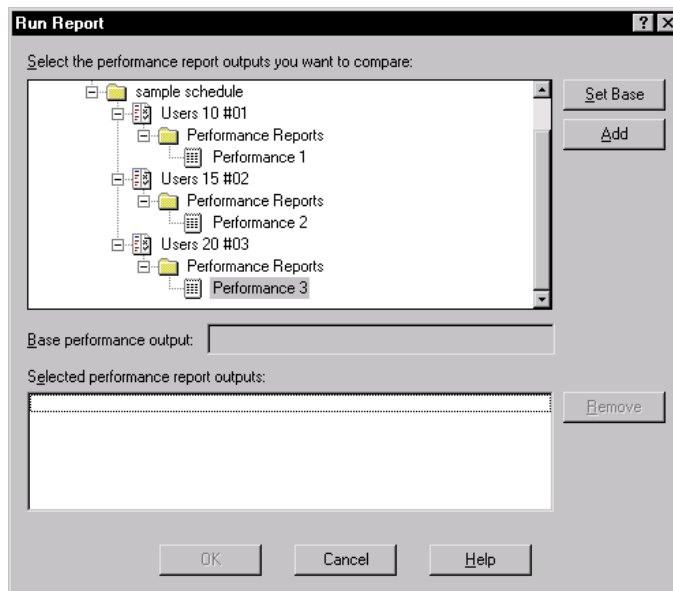
1. Click **Run** → **Report** → **Compare**.



This example shows only the default Compare report, but if you have defined other Compare reports, LoadTest displays them as well.

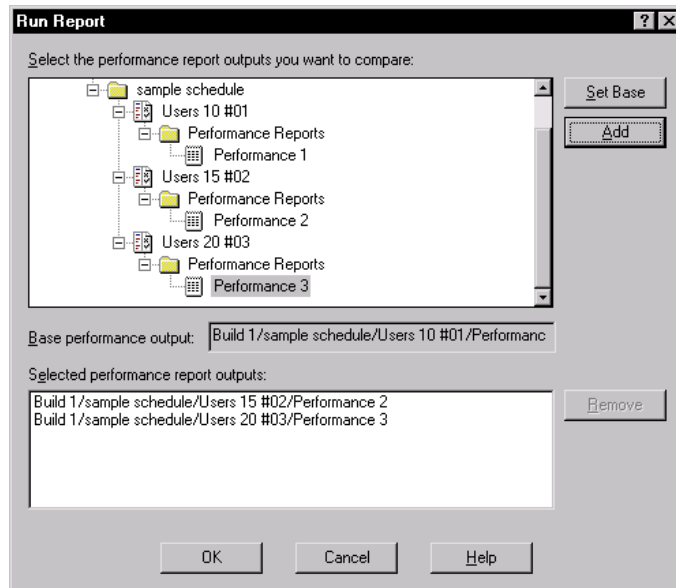
2. Select the Compare report that you want to run, and then click **OK**.

The following dialog box appears, which lets you select the output that you want to compare:



3. Select the **Base performance output** from the list. Note that the report output is stored under the logs, so you can click the plus signs on the left of the hierarchy to expand the list.
4. Click the **Set Base** button. LoadTest lists this output in the **Base performance output** box.
5. Click the Performance report output that you want, and then click the **Add** button, or simply double-click the Performance output. This adds the report output to the compare list.

The following dialog box shows a report that uses the output from a ten-user run as the base, and compares a fifteen-user run and a twenty-user run to that base:



6. Click **OK** to run the Compare report.
7. Click **File** → **Save** (or **File** → **Save As**) to save the report output.

Customizing Reports

LoadTest lets you customize reports for your particular testing requirements.

You can customize a report by:

- ▶ Filtering the data. For example, you can filter the report so that it contains only one user group, only certain scripts, and only certain command IDs.
- ▶ Change a report's advanced options. For example, you can modify a Response report so that extremely long responses are not included in the output.
- ▶ Change a graph's type and appearance. For example, you can display a graph as a line graph or a bar graph.

After you have customized a report and saved it, you can use it repeatedly to quickly analyze the data that you need.

Filtering Report Data

LoadTest provides a set of reports with default settings and options. You can, however, filter the reports so that only certain data appears in the output.

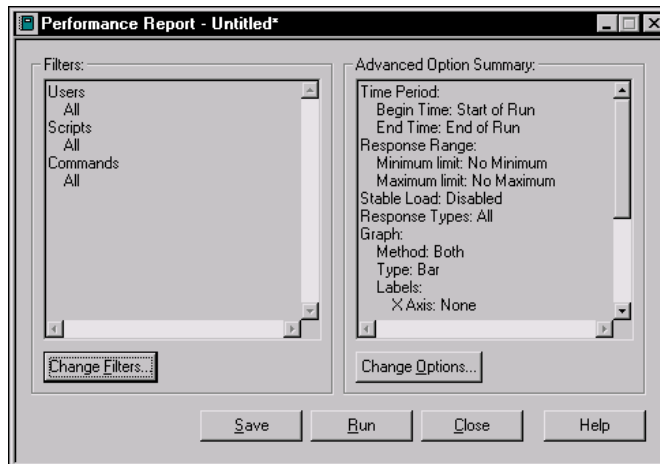
For example, the Performance report output on page 9-2 contains information from many command IDs, and the graph is rather complex. To see fewer command IDs, you can zoom in on the graph, as explained on page 9-32. Alternately, you can right-click the report output, click **Settings**, and then click **Select Command IDs**.

However, instead of filtering the report output, it is much easier to simply filter the report beforehand so that the output contains only the information you are interested in.

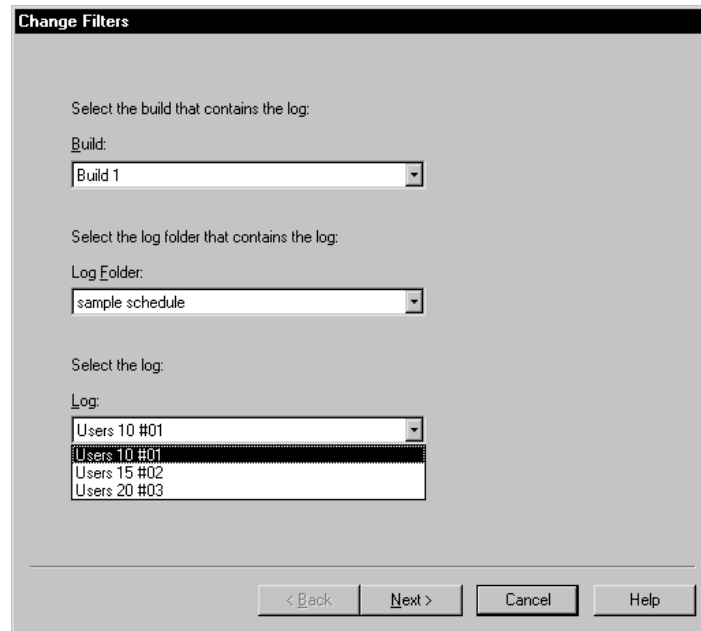
You can filter a report so it includes only certain users, only certain scripts, or only certain commands.

The following example explains how to filter a Performance report so that its output shows only information about the Data Entry user group.

1. Click **File** → **New**, and select a report. In this case, select the Performance report.



2. Since you want to filter the users, click **Change Filters**.



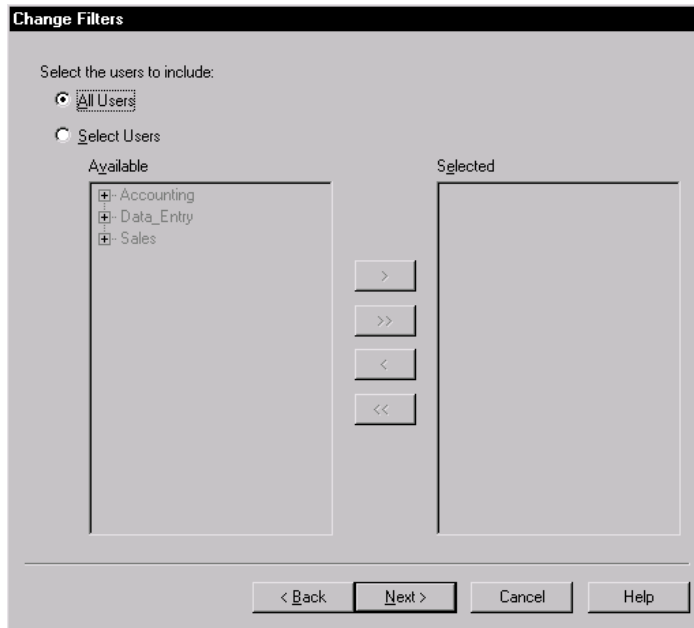
The image shows a dialog box titled "Change Filters" with a grey background and a black border. It contains three sections for selection:

- Select the build that contains the log:** A dropdown menu labeled "Build:" with "Build 1" selected.
- Select the log folder that contains the log:** A dropdown menu labeled "Log Folder:" with "sample schedule" selected.
- Select the log:** A dropdown menu labeled "Log:" with a list of options: "Users 10 #01", "Users 15 #02", and "Users 20 #03". The "Users 10 #01" option is currently selected and highlighted.

At the bottom of the dialog box, there are four buttons: "< Back", "Next >", "Cancel", and "Help".

3. Select the build, schedule, and log that you want to run the report on, and then click **Next**.

NOTE: If you are filtering users, you should generally select the log with the largest number of users. This ensures that your report will filter all of the users. For example, if you select **Users 10 # 01**, only 10 users are filtered—even if you run the report against 15 or 20 users. However, if you select the log with the largest number of users (**Users 20 # 03**) you can run the report against any of the logs shown, and all of the users will be filtered.



4. Because you are filtering a user group, click **Select Users**. Then select the Data Entry user group, and click > to move the user group to the **Selected** column.

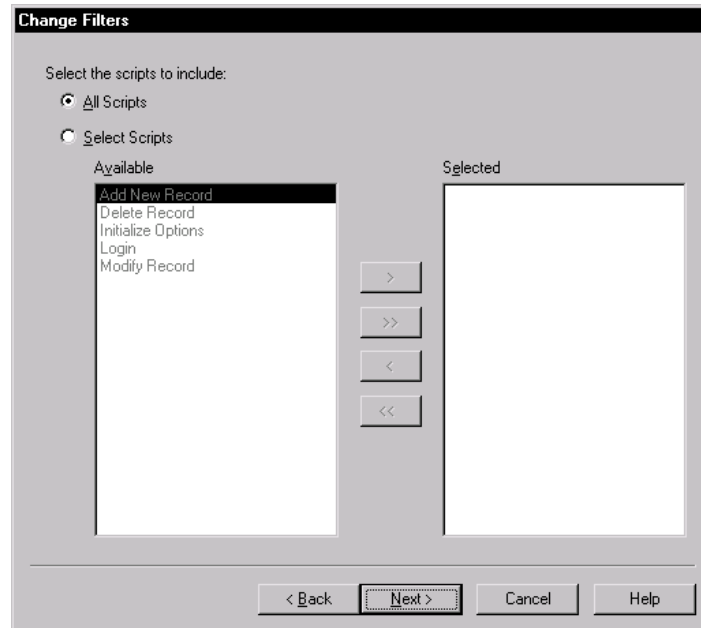
To select an individual user, click the plus sign next to the user group, which expands it to show the individual users. Then, select one or more users in the **Available** list and click > , or click > > to add all of the users. You can also double-click a user in the **Available** list to add it to the **Selected** list.

To remove users from the selected list, make your selection and click < or < < .

When you finish selecting the users, click **Next**.

NOTE: Clicking **All Users** is not the same as clicking **Select Users** and then moving all of the users to the **Selected** column. Clicking **All Users** produces a report that selects all users—no matter which log you run against the report. Moving all of the users into the **Selected** column filters only the users in that log. For example, assume that your log contains 20 users, and you move them all into the **Selected** column. If you run that report against a 200-user log, the report will include only the 20 users that you filtered. However, if you click **All Users**, the report will include all users.

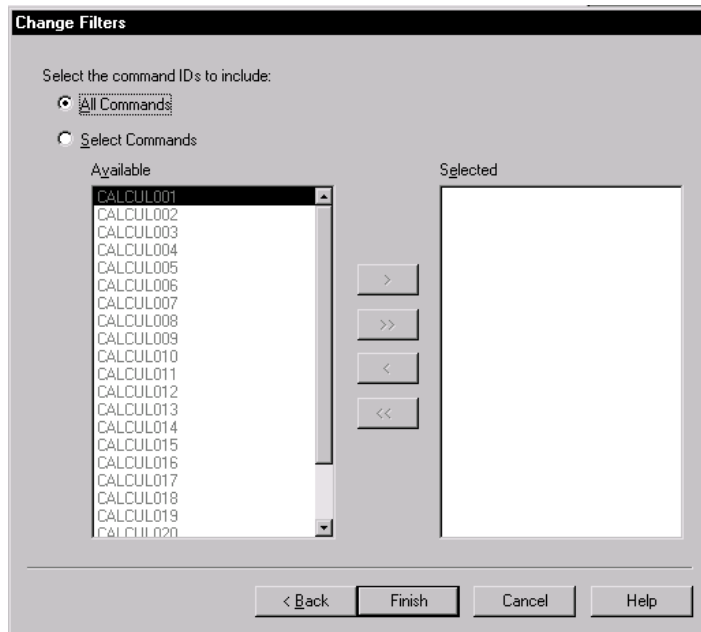
5. The following dialog box appears, which lets you select the scripts that you want to include. Because you have filtered the users, this list includes only the scripts that the Accounting users run:



- To select all scripts, without filtering, click **All Scripts**.
- To select a script, click **Select Scripts**. Then select a script, and click > to move it to the **Selected** column.

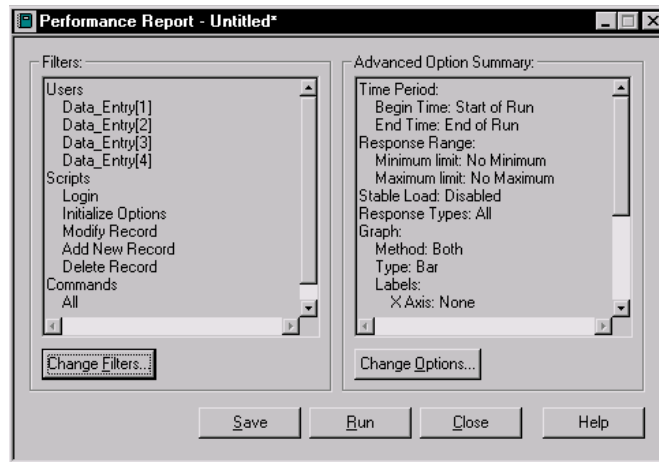
When you finish selecting the scripts, click **Next**.

- The following dialog box appears, which lets you select the command IDs that you want to include:

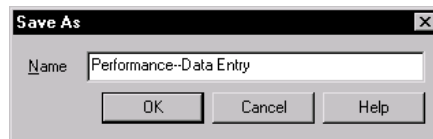


- To select all command IDs, without filtering, click **All Commands**.
 - To select a particular command ID, click **Select Commands**. Then select a command, and click > to move it to the **Selected** column.
- When you finish selecting the commands, click **Finish**.

The following window appears. From this window, you can change the filters again, change the advanced options, run the report immediately, or save it.



8. To save the report, click **Save**.
9. Type a new name for the report, and then click **OK**.



NOTE: It is a good idea to select a name that reflects the items being filtered. This lets you quickly identify a report.

The next time you open a Performance report, LoadTest displays the report that you have defined.

NOTE: The previous example shows how to define a report that filters data. However, you can also filter the output *after* you run a report. When the output is displayed, click **View** → **Settings**, and then click the **Select Command IDs** or **Response Range** tabs. For more information, see *Filtering Command IDs that Appear in a Graph* on page 9-33.

Setting Advanced Options

All LoadTest reports have advanced options, which determine how the report output is calculated and displayed. The specific advanced options are different for each report. To “fine tune” a report, you can change the advanced options.

To see the advanced options for a report:

1. Click **File** → **New**, and then click a report.
2. Under **Advanced Option Summary**, click the **Change Options** button.
3. The Advanced Options dialog box appears.

NOTE: For more information about advanced options, see *advanced report options, setting* in the LoadTest Help index.

The following table summarizes each advanced option, and lists the reports that use the option:

This advanced option	Is in these reports
Graph – Display the report output as a graph, a table, or both, change the type of graph displayed, change the labels for the graph axes, and add headers and footers.	Status, Performance, Response, Compare
Response Range – Include only responses that fall between a maximum and minimum time. The default includes all response times. However, you may want to set a maximum response time to eliminate outliers. If you change this option for one report, change the other reports too, so that the output reflects the same information. For more information, see <i>Eliminating Outliers</i> on page 9-24.	Status, Performance, Response, Compare
Response Types – Include only HTTP responses or responses with timers. The default includes all responses. The Status and Response reports also let you filter responses that contain verification points.	Status, Performance, Response
Sort Method – Sort command IDs numerically or in the order in which they were run. The default is to sort command IDs alphabetically.	Status, Performance, Response

(Continued)

This advanced option	Is in these reports
<p>Stable Load – Specify a number of users that must be logged on before results are reported. The default is to report results when any number of users are logged on. You may want to change this option so that a certain number of users, or all users, must be logged on. For more information, see <i>Reporting on a Stable Load</i> on page 9-25.</p> <p>If you change this option for one report, change the other reports, too, so that the output reflects the same information.</p>	Status, Performance, Response
<p>Time Period – Report on a specific portion of the schedule run. The default is to report on the entire run.</p>	Status, Performance, Response
<p>Calculation – Change how response times are calculated. Generally, the default is adequate. The default measures the time from the end of the last send command until the last byte of the response is received. If you change this option for one report, change the other reports, too, so that the output reflects the same information.</p>	Performance, Response
<p>Response Status – Include only passed responses, or only failed responses. The default is to include all responses. Generally, the default is adequate.</p>	Trace, Response
<p>Summary – Summarize data by user, script, command ID (Status), or run (Usage). The default for the Status report is detailed by command ID; the default for the Usage report is by run.</p>	Status, Usage
<p>Options – Report only failed commands and display only certain nonprintable characters in your report. The default is to include only failed commands and no unprintable characters. You may want to include both passed and failed commands in your reports so that you can see the failed commands in context. For more information, see <i>Including Passed and Failed Commands</i> on page 9-25.</p>	Analog
<p>Percentiles – Change how the response times are grouped. Generally, the defaults of 50, 70, 80, 90, and 95 are adequate.</p>	Performance

(Continued)

This advanced option	Is in these reports
Timestamps – Omit timestamps from the report. When you omit timestamps, reports with the same emulation activity but different repetitions are identical.	Trace
Command Types – Include only SQL, HTTP, TUXEDO, IIOP, or socket commands; only <code>testCase</code> commands; or commands that include timers. The default is to include all VU emulation commands.	Trace
Script Filters – Include only certain line numbers or VU command counts. The default is to include all emulation commands.	Trace
Environment Variables – Include only certain types of VU environment variables. The default is to include all environment variables.	Trace

Eliminating Outliers

Report output often contains a few values, called **outliers**, that are completely out of the normal range. For example, suppose you run a Performance report on 1000 users. Your response time ranges from 2 to 7 seconds. However, the response for one command ID is 30 seconds—far more than normal. Since this occurs only once, and it is a nonrepresentative time, you want to eliminate it from the report output because it skews the data.

The following steps show how to eliminate outliers:

1. Click **File** → **New** → **Performance Report**.
2. Under **Advanced Option Summary**, click the **Change Options** button.
3. The Advanced Options dialog box appears. Click the **Response Range** tab.
4. Under **Maximum Limit**, click the **Specified maximum** box, and enter a limit; for example, 29 seconds. This eliminates any response that is more than 29 seconds.
5. Click **Save**, and save the report under a descriptive name such as **Performance-29 Sec Outliers**.

Now that you have defined the Performance--29 Sec Outliers report, you can run it immediately or save it. The next time you open a Performance report, LoadTest displays this report along with the default Performance report.

NOTE: The previous example shows how to eliminate outliers in a Performance report. You can also eliminate outliers in Status, Response, and Compare reports.

Reporting on a Stable Load

It is useful to limit your report so that it includes only times when you have a stable load. For example, you are probably not interested in response times when only a few users have logged on to the system, or when most of the users have logged off.

To define a Performance report that includes information only when the load is stable:

1. Click **File** → **New** → **Performance Report**.
2. Under **Advanced Option Summary**, click the **Change Options** button.
3. In the Advanced Options dialog box, click the **Stable Load** tab.
4. Select **All Users**, or select **Custom number of users**, and enter a number.
5. Click **OK**. The Performance Report window appears.
6. Click **Save** and save the report under a descriptive name, such as **Performance--Stable Load**.

Now that you have defined the Performance--Stable Load report, you can run it immediately or save it. The next time you open a Performance report, LoadTest displays this report along with the default Performance report.

NOTE: The previous example shows how to define a stable load in a Performance report. You can also define a stable load in Status and Response reports.

Including Passed and Failed Commands

It is often useful to include passed commands as well as failed commands in an Analog report.

For example, assume you are running a schedule with 20 virtual users. Of these 20 users, 17 users pass and three users fail. The three users fail when they modify a certain record.

When you examine the failed commands only, it is tempting to conclude that something is wrong with that particular record—perhaps the record does not exist in the database. However, to get a clear overall picture, you need to look at the passed commands as well as the failed commands. It could be that the 17 users who passed also tried to modify the record. In that case, the flaw does not lie with the record itself but with the logic of your script.

To include both passed and failed commands in an Analog report:

1. Click **File** → **New** → **Analog Report**.
2. Under **Advanced Option Summary**, click the **Change Options** button.
3. In the Advanced Options dialog box, select the **Include success and failure messages** check box.
4. Click **OK**. The Analog Report dialog box appears.
5. Click **Save** and save the report under a descriptive name, such as **Analog--Passed and Failed Commands**.

Now that you have defined the Analog--Passed and Failed Commands report, you can run it immediately or save it. The next time you open an Analog report, LoadTest displays this report along with the default Analog report.

Reporting on a Particular Command ID

The default Response report can look confusing because it contains information about every command ID. This information is useful for assessing trends in the data. However, you may also want to report on a particular command ID or a small group of command IDs, and display the report in a line histogram, which is easy to read.

The following steps show how to filter on a command ID, and how to display this command ID as a line histogram:

1. Click **File** → **New** → **Response Report**.
2. Under **Filters**, click the **Change Filters** button.
3. Select the log that you want to run the report on, and click **Next**.
4. Because you are filtering on a command ID, check the **Selected command IDs** box and then click **Next**.
5. A list of command IDs appears. Click the arrow buttons to move the appropriate command ID to the **Selected** column, and then click **Finish**.
6. Under **Advanced Option Summary**, click the **Change Options** button.
7. The Advanced Options dialog box appears. Click the **Graph** tab.
8. Under **Graph type**, click **Line Histogram**, and then click **OK**.

9. The Response Report dialog box appears. Click **Save** and save the report under a descriptive name, such as **Response--Accounting Command IDs**.

Now that you have defined the Response--Accounting Command IDs report, you can run it immediately or save it. The next time you open a Response report, LoadTest displays this report along with the default Response report.

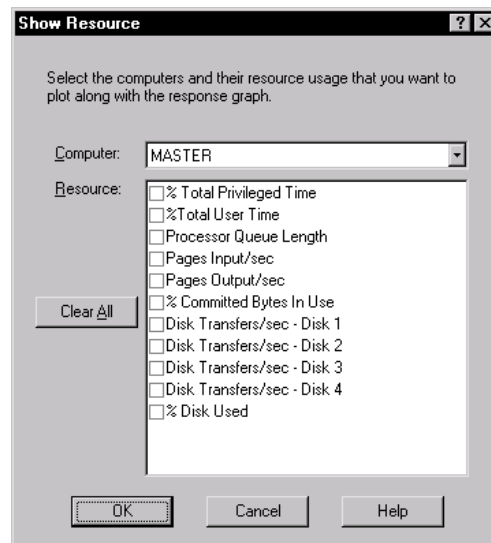
Mapping Computer Resource Usage onto Response Time

Monitoring computer resources is essential in performance testing. If you have a performance problem, you need to determine whether it is caused by a large number of users or by a hardware bottleneck. The Response report lets you overlay computer resource statistics over response time. If your response time increases, you can determine whether this was caused by a computer resource problem.

LoadTest needs to be set up to *collect* the information on computer resources before you can *report* on computer resources. Therefore, when you run a schedule, be sure to select the **Monitor resources** check box.

The following procedure shows how to create a report that maps computer resources onto response time.

1. Click **File** → **New** → **Response Report**.
2. Right-click on the graph, and click **Show Resources**.



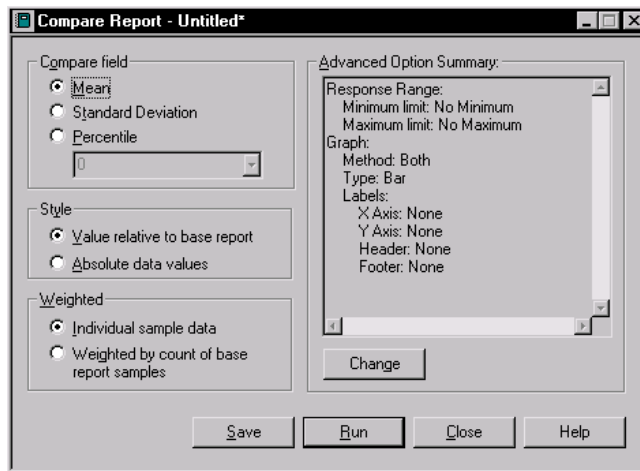
3. Select the computer you want to report on.

4. Select the check boxes you want to report on. Note that these boxes correspond to the resource monitoring categories that you can see when you monitor a schedule.
5. Click **OK**. A graph of computer recourses is superimposed on the Response report.

Defining a Compare Report

Defining a Compare report is similar to defining other reports. One useful Compare report defines the 90th percentile data and uses absolute, rather than relative, data values. To define this report:

1. Click **File** → **New** → **Compare Report**.



2. Use the options under **Compare field** to select the field of the Performance report outputs that you want to compare.
 - **Mean** – Compares the mean value of the response times.
 - **Standard Deviation** – Compares the standard deviation for the response times.
 - **Percentile** – Compares the response times based on the percentile that you select. The percentile must be in the Performance report. For example, if the Performance reports calculate the 50, 70, 80, 90, and 95 percentiles, the Compare report must use one of these percentiles.

This example shows how to define a Compare report for the 90th percentile, so click the **Percentile** box and enter **90**.

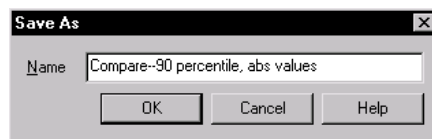
3. Use the options under **Style** to compare response times relative to the base performance output, or compare response times by their absolute values.
 - **Value relative to base report** – Compares the response times relative to the base performance output. With this option, the first column in the report (the base performance output) is always 1, and the other columns are relative to that number. So, for example, if the base lists a response time as 2.5, and another output lists the response time as 5, the Compare report lists them as 1 and 2.
 - **Absolute data values** – The indicated response times appear in the report. So, for example, if the base lists a response time as 2.5, and another output lists the response time as 5, the Compare report lists them as 2.5 and 5.

This example shows how to define a Compare report for the absolute values, so click **Absolute data values**.

4. The Weighted section lets you decide whether to weigh the response times that occur most frequently.
 - **Individual sample data** – The response times are not weighed. Therefore, a command ID that occurs ten times and a command ID that occurs 100 times will have an equal influence on the response times statistics.
 - **Weighted by count of base report samples** – The response times are weighted to reflect the frequency of occurrence of the command ID to which they correspond. Therefore, command IDs that occur more frequently will have more influence on the response time statistics, and command IDs that occur less frequently will have less influence on the statistics.
5. To change the Advanced options, click the **Change** button.

NOTE: For more information about advanced options, see *Setting Advanced Options* on page 9-22 and *advanced report options, setting* in the LoadTest Help index.

6. When you have completed your modifications, click **Save** to save your new report.
7. In the Save As dialog box, type a descriptive name:



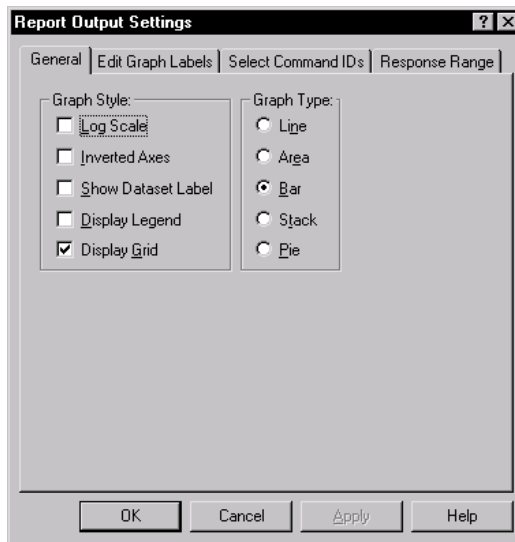
You have now defined a new Compare report. When you run this report, you select the Performance report outputs that you want to compare.

Changing a Graph's Appearance or Type

LoadTest displays the Compare, Performance, Response, and Status reports as graphs as well as reports. The Settings dialog box lets you change the type of graph that appears and enhance its display.

To change the type or appearance of a graph:

1. Click **File** → **Open** → **Report Output**, and double-click a report output.
2. Click **View** → **Settings**.

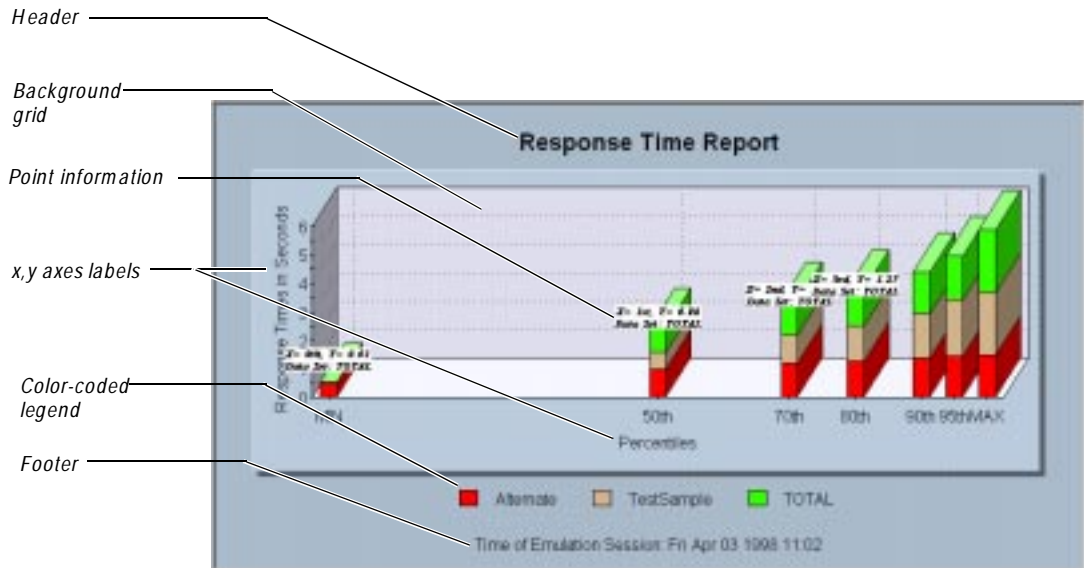


NOTE: The previous figure shows the Settings dialog box for Performance reports. The dialog box for other reports is slightly different.

From this dialog box, you can change the appearance of a graph, change the labels of a graph, and filter information such as the command IDs. In a Performance report, you can also change the response range that appears in the graph. The following sections describe how to do this.

Changing a Graph's Appearance

LoadTest lets you control a graph's format and appearance. You can display or clear information about selected points and datasets without affecting the graph's data. The following figure shows a stack graph with a header, background grid, and various other options:



To change a graph's appearance:

1. Click the graph that you want to change.
2. Click **View** → **Settings**. The Settings dialog box appears.
3. Select any of the following check boxes:
 - **Log Scale** – Scales any graphical display type to its logarithmic equivalent.
 - **Inverted Axes** – Switches the relative positions of the graph's axes.
 - **Show Dataset Label** – Applies the data set labels to the graph.
 - **Display Legend** – Displays a color-coded legend for all displayed graphical components (not available on the Response report).
 - **Display Grid** – Displays a grid that is useful for visual comparisons (not available on the Pie graph).

4. When you finish making changes, click **Apply**.
5. Click **OK** to close the dialog box.

Displaying and Clearing Data Point Information

When working with graphs, you may want to display the value of a specific point in a graph. To display information about a data point:

- ▶ Move the mouse pointer to the desired area of the graph, and click **CTRL-SHIFT-BUTTON 1**.

To clear data point information:

- ▶ Right-click the graph, and then click **Clear Point Information**.

Changing a Graph's Type

When working with graphs, you can change the type of graph that LoadTest displays.

To change a graph's type:

1. Click the graph that you want to change.
2. Click **View** → **Settings**. The Report Output Settings dialog box appears.
3. Select the type of graph that you want and click **Apply**.
4. Click **OK** to close the dialog box.

Enlarging and Rotating a Graph

By clicking combinations of **Shift/Control** keys and mouse buttons, you can further manipulate a graph's appearance:

To do this	Click this button	And then
Enlarge a graph's size.	CTRL-BUTTON 1 BUTTON 2	Drag the mouse toward the bottom of the graph.
Change a graph's position.	SHIFT-BUTTON 1 BUTTON 2	Move the mouse to reposition the graph.
Zoom in on a graph's axes.	SHIFT-BUTTON 1	Draw a box around the area to zoom, then release BUTTON 1 .
Zoom in on a graph's data.	CTRL-BUTTON 1	Draw a box around the area to zoom, then release BUTTON 1 .

(Continued)

To do this	Click this button	And then
Rotate the view of a graph (stack and pie graphs only).	BUTTON 1 BUTTON 2	Move the mouse up and down to change the inclination angle. Move the mouse left and right to rotate the graph (stack only).
Reset a graph to its original size.	the lowercase letter “r”	not applicable

Changing a Graph’s Labels

When working with a Status, Performance, or Compare graph, you can change the labels of the graph.

To change a graph’s labels:

1. Click the graph that you want to change.
2. Click **View** → **Settings**. The Settings dialog box appears.
3. Click the **Edit Graph Labels** tab. This tab lets you assign or modify labels for a graph’s *x* and *y* axes, as well as for its header and footer.
4. Type the labels that you want.
5. Optionally, to change the font, style, or size of the axis labels, header, or footer, click the **Select** button. Choose the appropriate settings, and click **OK** to close the Font dialog box.
6. Click **Apply**.

Filtering Command IDs that Appear in a Graph

LoadTest lets you to filter report data *before* you create the report output. However, you can also filter command IDs that appear in a graph *after* you have created the report output. This feature is useful if you create a graph that is rather complex, and you want to examine portions of it in more detail.

To filter the command IDs in a graph:

1. Click the graph that you want to change.
2. Click **View** → **Settings**. The Settings dialog box appears.

3. Select the command IDs that you want to appear in the output, and click < to move them to the **Available** column.
4. Click **Apply**.

For information about defining a report that filters command IDs, see *Filtering Report Data* on page 9-16.

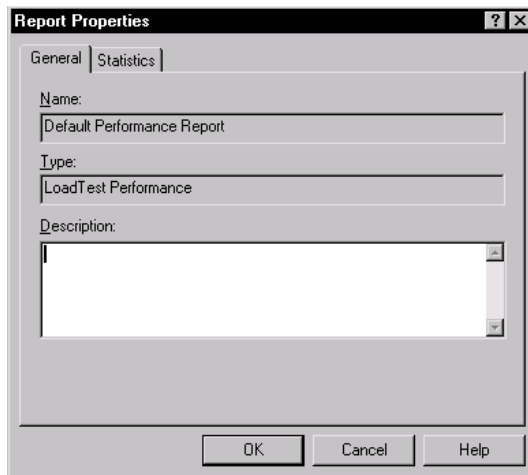
Editing the Properties of a Report or Report Output

A report's properties are stored in the repository. These properties include the name of the report, a description of the report, who created the report, and when the report was created.

Most of these properties are added automatically. However, you may want to add a description of the report or the report output. To do this, you edit the report properties.

To edit the properties of a report or report output:

1. Click the report or report output whose properties you want to edit.
2. Click **File** → **Properties**. The Report Properties (or Report Output Properties) dialog box appears:



3. Click the tab that you want to edit.
4. Add a description of the report or the report output, and click **OK**.

Changing Report Defaults

LoadTest automatically generates Performance and Status reports at the end of a schedule run. In addition, you can click a report on the report bar, and LoadTest will run the report that you click.

You can change the reports that LoadTest generates at the end of a run. For example, you can have LoadTest automatically display a Usage report in addition to the Performance and Status reports. Or you can have LoadTest generate a Performance report that you have defined instead of the default Performance report.

You can also change the reports that LoadTest runs when you click the report bar button. For example, instead of having LoadTest run the default Performance report, you can have it run a Performance report that you have defined.

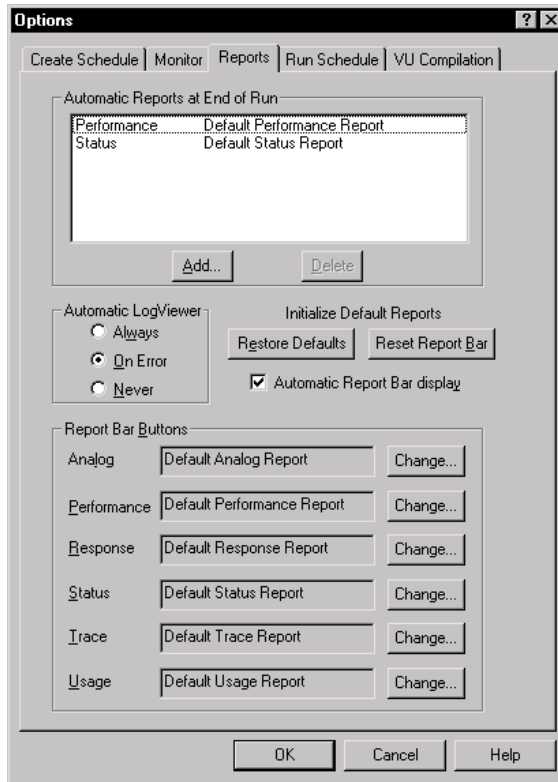
The following sections explain how to change the reports generated at the end of a run and how to change the reports that are generated when you click a report on the report bar.

Changing the Reports that Run Automatically

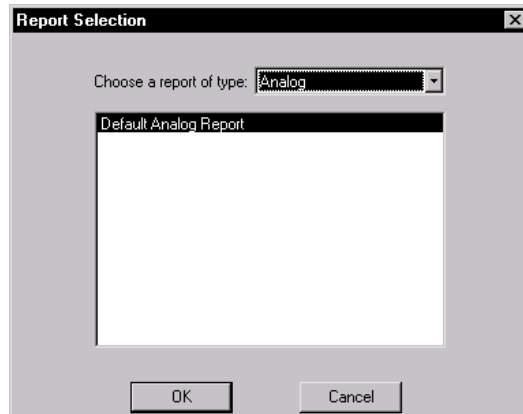
LoadTest automatically displays Performance and Status reports at the end of the schedule run. However, you can change the reports that LoadTest displays.

To change the reports that LoadTest displays at the end of a schedule run:

1. Click **Tools** → **Options**, and then click the **Reports** tab.



2. To delete a report, click its name in the **Automatic Reports at End of Run** list, and then click the **Delete** button.
3. To add a report, click the **Add** button.



Select the type of report that you want to add.

LoadTest lists the default report as well as any other reports that you have defined. Click the report that you want LoadTest to display.

4. After you have finished adding and deleting reports, click **OK**.

At the end of a schedule run, LoadTest displays the reports that you have selected.

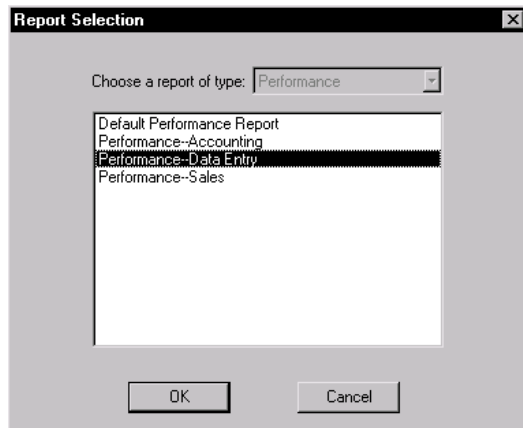
Changing the Reports that Run from the Report Bar

The report bar lets you run reports by simply pressing a button. This bar automatically runs the default reports. However, you may have defined a new report that you want to run instead of the default reports.

The following example shows how to customize the report bar so that when you click the **Perf** button, the Performance--Data Entry report runs instead of the default Performance report:

1. Click **Tools** → **Options**, and then click the **Reports** tab. The Options dialog box appears. The Report Bar Buttons area lists the reports that appear automatically at the end of a schedule run.
2. Click **Change** next to the report that you want to change. In this example, you are changing the Performance report.

3. The dialog box displays both the default Performance report and any Performance reports that you have defined:



4. Click **Performance--Data Entry**, and click **OK**.

The next time you click the **Perf** button, LoadTest runs the Performance--Data Entry report.

NOTE: To reset the report bar so that it generates the default reports, click **Tools** → **Options**, click the **Reports** tab, and then click the **Reset Report Bar** button.

Types of Reports

The following sections describe each LoadTest report in detail, and give an example of each type of report.

Analog

Use Analog reports to identify the differences between expected and actual responses in a virtual user script.

The report output shows the commands sent, and the rows and errors received, during the run. The output displays all client/server traffic related to each VU emulation command.

The following figure shows an example of Analog report output:

```

==> Start of data for input log file Accounting[1] <==
<<< sqlxrec[CALCUL001]: script = Calculate Hours(1), source = Calculate Hours(20
) >>>
alter session set nls_language= 'AMERICAN' nls_territory= 'AMERICA' nls_currency
= '$' nls_iso_currency= 'AMERICA' nls_numeric_characters= ',' nls_date_format=
'DD-MON-YY' nls_date_language= 'AMERICAN' nls_sort= 'BINARY'
0 rows processed
<<< sqlprepare[CALCUL002]: script = Calculate Hours(2), source = Calculate Hours
(27) >>>
1= SELECT USER FROM DUAL
<<< sqlxrec[CALCUL003]: script = Calculate Hours(3), source = Calculate Hours(31
) >>>
(1) SELECT USER FROM DUAL
0 rows processed
<< * sqlnrecv[CALCUL004]: script = Calculate Hours(4), source = Calculate Hours(3
3) * >>>
1 row received from 1 table
ERROR 0: No error

```

What's in Analog Report Output?

Analog report output contains the following information:

Header and trailer lines, which show you which virtual user generated the data. In the following example, the header and trailer lines show data from the first virtual user in the Accounting user group, Accounting[1]:

```

==> Start of data for input log file Accounting[1] <==
==> End of data for input log file Accounting[1] <==

```

Inside the header and trailer lines, the Analog report shows data on each emulation command that a virtual user executed. For example:

```

*** cmd[cmdID]: script = scrname(cmdcnt), source = scrname(srcline) ***

```

Syntax	Meaning
<<<	<<< Indicates that the command recorded and played back successfully.
<< *	<< * Indicates that an error occurred during recording and the same error occurred during playback.
***	*** Indicates that no error occurred during recording but an error occurred during playback.
cmd	The name of the VU emulation command.
cmdID	The command ID.

(Continued)

Syntax	Meaning
<i>scrname</i>	The name of the script.
<i>cmdcnt</i>	The number of VU emulation commands that have been executed in that script.
<i>source</i>	In most cases, this is the same as the script. However, if a script calls another script, or if a VU script contains an include file, the calling script or the file that contains the include directive is displayed.
<i>srcline</i>	The source line.

Compare

The Compare report compares the response times measured by Performance reports. After you have generated output from several Performance reports, you can use a Compare report to compare the values of a specific field.

You can compare output that shows different numbers of users. You can also compare output from runs on different system configurations.

A Compare report can compare the output from one base report and up to six other Performance reports.

What's in Compare Report Output?

A Compare report can compare report output in a number of ways. It can compare the output absolutely, or it can compare the output relative to the base report that you select. In addition, the response times can be weighted, so that command IDs that occur frequently have more influence, or they can be unweighted, so that each command ID has equal influence.

There are four versions of the Compare report, and the following sections explain each one. In each report, the same Performance reports are used as input; the only difference is in the type of Compare report.

Absolute Compare Reports

Absolute Compare reports display the actual values of the response times, in seconds. The final line of the output gives the arithmetic sum of the response times.

The following figure shows the last few lines of an absolute Compare report output:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	0.01	0.04	0.01	0.01
150	READ R018	0.01	0.02	0.01	0.01
151	READ R019	0.00	0.01	0.00	0.00
152	READ R020	0.00	0.01	0.00	0.01
153	READ R021	0.00	0.01	0.00	0.00
154	READ R022	0.00	0.00	0.00	0.00
155	SUM	2.73	4.79	5.11	12.05

To define an absolute Compare report:

1. Click **File** → **New** → **Compare Report**.
2. Under **Style**, click **Absolute data values**.
3. Under **Weighted**, click **Individual sample data**.
4. Click **Save**, and save the report under a descriptive name.

Weighted Absolute Compare Reports

Weighted absolute Compare reports weigh response times by their frequency of occurrence and are useful for comparing “total” response times.

The weight applied is equal to the number of valid responses for that command ID in each output. If the command IDs in the output have a different number of responses, LoadTest uses the smallest non-zero number as the weight.

The weighted absolute value is the product of this weight and the absolute value for the response time. The final line of the weighted absolute Compare report gives the arithmetic sum of the weighted response times for each output.

The following figure shows the last few lines of a weighted absolute Compare report output:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	0.03	0.12	0.03	0.03
150	READ R018	0.03	0.06	0.03	0.03
151	READ R019	0.00	0.03	0.00	0.00
152	READ R020	0.00	0.03	0.00	0.03
153	READ R021	0.00	0.03	0.00	0.00
154	READ R022	0.00	0.00	0.00	0.00
155	WEIGHTED SUM	9.11	14.61	17.45	41.53

To define a weighted absolute Compare report:

1. Click **File** → **New** → **Compare Report**.
2. Under **Style**, click **Absolute data values**.
3. Under **Weighted**, click **Weighted by count of base report samples**.
4. Click **Save**, and save the report under a descriptive name.

Relative Compare Reports

Relative Compare reports list the “base” response time as 1.00 and the other response times relative to that base.

The final line of the output gives the geometric mean of the relative response times for each output. To determine the geometric mean, LoadTest multiplies the response times, and then takes a root of the product that is equal to the number of response times. For example, if there are five response times, LoadTest multiplies them together and takes the fifth root of the product.

Mathematically, the geometric mean of a set of values x_1, x_2, \dots, x_k is expressed as:

$$(x_1 x_2 \dots x_k)^{1/k}$$

The following figure shows the last few lines of a relative Compare report output:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	1.00	4.00	1.00	1.00
150	READ R018	1.00	2.00	1.00	1.00
151	READ R019	1.00	10.00	1.00	1.00
152	READ R020	1.00	10.00	1.00	10.00
153	READ R021	1.00	10.00	1.00	1.00
154	READ R022	1.00	1.00	1.00	1.00
155	GEO MEAN	1.00	1.51	1.07	1.44

To define a relative Compare report:

1. Click **File** → **New** → **Compare Report**.
2. Under **Style**, click **Value relative to base report**.
3. Under **Weighted**, click **Individual sample data**.
4. Click **Save**, and save the report under a descriptive name.

Weighted Relative Compare Reports

This report is the same as the relative Compare report, except that it also lists the weighted geometric mean.

The weighted geometric mean differs from the geometric mean in that it takes into account the frequency with which the different command IDs occur. Thus, frequently used command IDs have a greater influence on the weighted geometric mean than infrequently used ones, in contrast to the geometric mean, where all command IDs have equal influence.

The weight applied when calculating the weighted geometric mean for each command ID equals the number of valid responses for that ID in each file being compared. If the number of valid responses for a command ID differs among the files, the smallest non-zero count is used as its weight.

Mathematically, the weighted geometric mean of a set of values x_1, x_2, \dots, x_k having frequencies (weights) f_1, f_2, \dots, f_k , where $f_1 + f_2 + \dots + f_k = N$, is expressed as:

$$(x_1^{f_1} x_2^{f_2} \dots x_k^{f_k})^{1/N}$$

The following figure shows the last few lines of a weighted relative Compare report output:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	1.00	4.00	1.00	1.00
150	READ R018	1.00	2.00	1.00	1.00
151	READ R019	1.00	10.00	1.00	1.00
152	READ R020	1.00	10.00	1.00	10.00
153	READ R021	1.00	10.00	1.00	1.00
154	READ R022	1.00	1.00	1.00	1.00
155	GEO MEAN	1.00	1.51	1.07	1.44
156	WGHT GMEA	1.00	1.30	1.08	1.47

To define a weighted relative Compare report:

1. Click **File** → **New** → **Compare Report**.
2. Under **Style**, click **Value relative to base report**.
3. Under **Weighted**, click **Weighted by count of base report samples**.
4. Click **Save**, and save the report under a descriptive name.

N/A and Undefined Responses

Occasionally, you might see the strings `n/a` and `Undefn` in the output of a Compare report. The following table describes when LoadTest displays these strings:

If this occurs	The Compare report output will
A command ID is in the base report but does not exist in the other report output.	List <code>n/a</code> for that command ID.
A command ID is in the report output but does not occur in the base report.	Ignore that command ID.
You are producing a relative report, and some command IDs have a response time of 0.	List the response time as <code>0</code> in the base report, and list the other results corresponding to that command ID as <code>Undefn</code> .
All the response times for a report output are listed as <code>n/a</code> or <code>Undefn</code> .	List the geometric mean or sum as <code>n/a</code> .

Performance

Use Performance reports to display the response times observed during the schedule run. Performance report output displays the response times for the selected commands. It also provides the mean, standard deviation, and percentiles for response times.

Performance reports use the same input data as Response reports, and provide similar data sorting and filtering. However, Performance reports group responses with the same command ID, while Response reports shows each command ID individually.

The following figure shows an example of Performance report output. The graph plots the seconds of response time against preset percentiles. Thus, this graph shows 15 bars for each percentile category, because a total of 15 commands are graphed. The MIN category shows the minimum response time for each command ID. The 50th category shows the 50th percentile of time for each command ID. (Half of the command IDs had a shorter response time and half had a longer response time.) The MAX category shows the maximum response time for each command ID.



What's in Performance Report Output?

Performance report output contains the following information:

- ▶ **Cmd ID** – The command ID associated with the response.
- ▶ **NUM** – The number of responses for each command ID.
- ▶ **MEAN** – The arithmetic mean of the response times of all responses for each command ID.
- ▶ **STD DEV** – The standard deviation of the response times of all responses for each command ID.
- ▶ **MIN** – The minimum response time of all responses for each command ID.
- ▶ **50th, 70th, 80th, 90th, 95th** – The percentiles of the response times of all responses for each command ID. So, for example, if the 95th percentile of Add Ne002 is 0.53, 95% of the responses are less that 0.53 seconds.
- ▶ **MAX** – The maximum response time of all responses for each command ID.

To display the total response time in the graph and in the last line of the report output follow these steps:

1. Click **File** → **Open** → **Report Output**, and select the output from a Performance report.

2. Right-click on the graph and select **Settings**.
3. Click the **Select Commands IDs** tab, and select **TOTAL**.

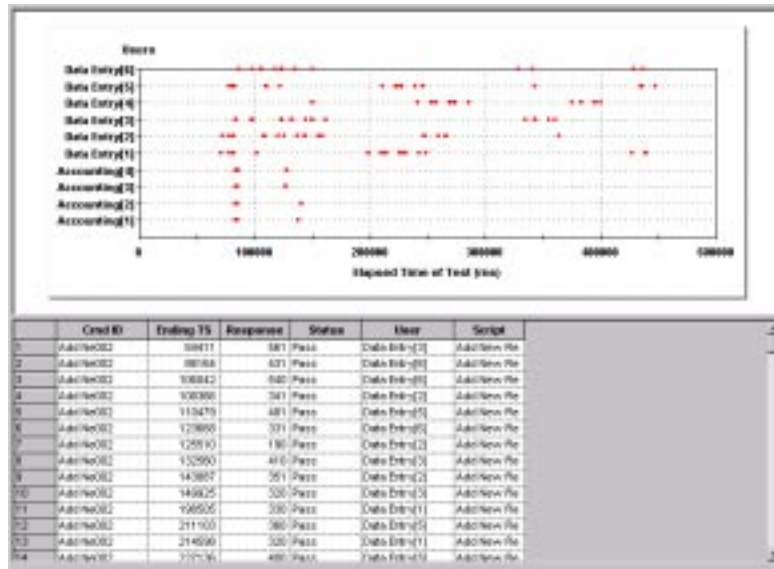
Response

Response reports display individual response times. Response reports use the same input data as Performance reports, and provide similar data sorting and filtering. However, Response report output shows each command ID individually, while Performance report output groups responses with the same command ID.

Response reports are useful for the following:

- ▶ Checking the trend in the response time. The Response report shows the response time versus the elapsed time of the schedule run. The response time should be clustered around one point rather than getting progressively longer or shorter. If the trend changes, check that you have excluded login and setup time in your results. The worst case is that you might need to change your test design.
- ▶ Checking any spikes in the response time. If the response time is fairly flat except for one or two spikes, you may want to investigate the cause of the spikes.
- ▶ Filtering the data so that it contains only one command ID, and then graphing that command ID as a histogram.
- ▶ Optionally, checking the resources used by a computer in the run. To see the resources used, right-click on the Response report output and select a computer.

The following figure shows an example of Response report output. The graph plots each virtual user versus the response time, in milliseconds. Thus, this graph shows that the first user in the accounting group (Accounting 1) executed two commands. This graph contains many short lines, which resemble dots, and indicate that the response times for all the users are quite short. The longer a line is on the X axis, the longer the response time, because the X axis graphs the response time.



What's in Response Report Output?

Typically, Response report output contains two sections, one for expected responses and one for unexpected responses. The responses within each section are sorted by command ID, and within each command ID, responses are sorted by the ending timestamp.

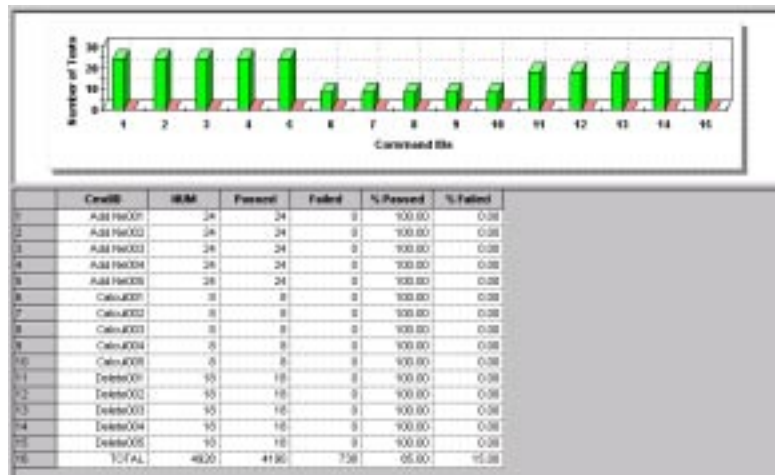
Response report output contains the following information:

- ▶ **Cmd ID** – The command ID associated with the response.
- ▶ **Ending TS** – The ending timestamp of the response. This timestamp corresponds to the value of VU language read-only variable `_lr_ts` for the response. The timestamp is the interval ending timestamp as defined by the Time Period report option.
- ▶ **Response** – The response time in milliseconds.
- ▶ **Status** – Displays **P** or **F** to indicate whether the response passed or failed.
- ▶ **User** – The virtual user corresponding to the response.
- ▶ **Script** – The name of the script corresponding to the response.

Status

Status reports show how well the responses you actually received during playback correspond with the responses that you expected to receive. If the response you received is the same or is expected, LoadTest considers that it has passed; otherwise, LoadTest considers it failed.

The following figure shows an example of Status report output. The graph plots the command number against the number of times the script ran, and displays commands that passed in green, and commands that failed in red. Thus, this graph shows that command 1 (command ID Add Ne01) ran 24 times and did not fail, and command 6 (command ID Calcul001) ran 8 times and did not fail.



What's in Status Report Output?

Status report output contains the following information:

- ▶ **Cmd ID** – The command ID associated with the response.
- ▶ **NUM** – The number of responses corresponding to each command ID. This number is the sum of the numbers in the **Passed** and **Failed** columns.
- ▶ **Passed** – The number of passed responses for each command ID (that is, those that did not time out).
- ▶ **Failed** – The number of failed responses for each command ID that timed out (that is, the expected response was not received).
- ▶ **% Passed** – The percentage of responses that passed for that command ID.
- ▶ **% Failed** – The percentage of responses that failed for that command ID.

The last line of the report output lists the totals for each column.

Trace

Trace reports list the activity of a run and examine in detail unusual or unexpected events. This report formats raw data from the schedule run without performing statistical analysis.

Trace report output displays the actual timestamps from the emulation, the counts of data sent and received, and the environment variable settings (if recorded). To display the actual data sent or received, run an Analog report.

The following figure shows an example of Trace report output:

```

Information for User Data_Entry[1]:
User's Environment: Screen = D, Server_connection = 1, Emulation = NONE,
                   Screen_match = CURSOR_DATA
User's Environment: Screen_mask = ()
User: Data_Entry[1] Release 7.0 Script: Add
Beginning timestamp of script Add:      7190
User's Environment: Server_connection = 1 ("calvin_1521")
Login Timestamp:      37182

```

Src Line	Cmd Count	Clnt	Command	Command ID	Count	First Timestamp	Last Timestamp	Stat
20	1	1	sqlexec	ADD NR001	0	37322	38103	pass
27	2	1	sqlprepare	ADD NR002	1	39927	39946	pass

What's in Trace Report Output?

Trace report output contains the following information:

- ▶ **Total Number of Users Emulated** – The number of virtual and GUI users in the schedule run.
- ▶ **Number of Users in Report** – The number of users that appear in the output. If you have filtered users, this number is different than the total number of users emulated.
- ▶ **LoadTest Release** – The release of the LoadTest system that you used when recording.
- ▶ **Time of Schedule Run** – The time that the schedule started to run.
- ▶ **Time of Day Reference for Timestamps** – The time that the schedule started to run, given as the number of seconds since 00:00:00 GMT, January 1, 1970. If you have a UNIX computer available, the UNIX `ctime(3)` library routine converts this and other timestamps into time-of-day format.

For each user, the Trace report lists the following information:

- ▶ A line indicating the user that is being reported. In this example, the user is `Data_Entry[1]`, or the first user in the Data Entry user group.
- ▶ Default environment variable values and environment variable changes taking place on each virtual user during playback. The changes correspond to environment control commands executed during the schedule run.

NOTE: Unless you set the record level to ALL during recording, most environment variable values are not recorded, and thus do not appear in the Trace report. The default record level is ESSENTIAL. For more information, see *Changing the Record Level Value* on page 7-46.

For each script, the Trace report lists the following information:

- ▶ A script header, which lists the user, the release, and the name of the script.
- ▶ **Beginning timestamp** – The time, in milliseconds (since the beginning of the schedule run), at the start of execution of each script.
- ▶ **Login Timestamp** – Recorded for each user. This corresponds to the time when the user's login status changed to ON.
- ▶ Emulation commands executed during the schedule run. A three-line heading appears if any emulation commands are included. The Trace report gives the following information for each emulation command:
 - **Src Line** – The line number of the command in the source file.
 - **Cmd Count** – A running tally of the number of emulation commands executed in a script. This column shows 0 for SQABasic timers.
 - **Clnt** – The value of the `Server_connection` environment variable associated with each SQL, HTTP, or socket emulation command.
 - **Attached to Server** – For TUXEDO sessions that start with a `tux_tpinit` emulation command, this line appears.
 - **Command** – The emulation command that was executed. If an HTTP response was read from cache and not from the network, the report output shows `http_nrecv(c)` (rather than `http_nrecv`).
 - **Command ID** – The command ID associated with the command

- **Count** – These values depend on the type of emulation command. For SQL emulation commands, this is generally the number of rows processed. For HTTP and socket commands, this is generally the number of bytes processed. For TUXEDO emulation commands, this is a code which can be 0 (buffer neither sent nor received), 1 (buffer sent but not received), 2 (buffer not sent but received), or 3 (buffer sent and received).
- **First Timestamp** – The time, in milliseconds, that the command started executing, relative to the beginning of the schedule run. In the Trace example, `sqlexec` started executing 37.322 seconds after the schedule began running.
- **Last Timestamp** – The time, in milliseconds, that the command stopped executing, relative to the beginning of the schedule run. In the Trace example, `sqlexec` stopped executing 38.103 seconds after the schedule began running.
- **Stat** – Whether the emulation command passed or failed.
- **Server n ("server_name") disconnected** – Indicates that the server connection has been closed.
- **Ending Timestamp** – The time, in milliseconds, that the script stopped executing, relative to the beginning of the schedule run. This value is reported for each script. The duration of the script is also reported.
- **Logout Timestamp** – The time, in milliseconds, that the user's login status changed from ON to OFF.

Usage

Usage reports display data on all VU emulation commands and responses. The report output describes throughput and user characteristics during the schedule run.

The following figure shows an example of Usage report output:



What's in Usage Report Output?

Usage report output contains a section on cumulative statistics and a section on summary statistics.

Cumulative Statistics

- ▶ **Active Time** – The sum of the active time of all users. The active time of a virtual user is the time the user spent thinking (including delays after the user's first recorded command), executing commands, and waiting for responses.
- ▶ **Inactive Time** – The sum of the inactive time of all users and scripts. The inactive time of a virtual user is the time before the user's first emulation command (including overhead time needed to set up and initialize the run), and, possibly, inter-script delay (the time between the last emulation command of the previous script and the beginning of the current script).
- ▶ **Passed Commands** – Total number of passed `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, `http_request`, `sock_send`, and `emulate` commands executed.
- ▶ **Failed Commands** – Total number of failed `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, `http_request`, `sock_send`, and `emulate` commands executed.

- ▶ **Passed Responses** – Total number of responses to input commands that were matched by passing receive commands (`sqlnrecv`, `sqllongrecv`, `http_nrecv`, `http_recv`, `sock_nrecv`, and `sock_recv`). This is not the same as the total number of expected receive commands, since a response may be matched by an arbitrary number of receive commands. A response is considered expected if all receive commands used to match it have an expected status.
- ▶ **Failed Responses** – Total number of responses that were matched by failing VU receive emulation commands. This is not the same as the total number of unexpected receive commands, since a response may be received by an arbitrary number of receive commands. A response is considered unexpected if any receive commands used to match it have an unexpected status.
- ▶ **Average Throughput** – Four measurements of average throughput are provided: passed command throughput, failed command throughput, passed response throughput, and failed response throughput. This represents the throughput of an average user.
- ▶ **Time Spent Waiting** – Total time spent waiting for responses, given both in seconds and as a percentage of active time. Time spent waiting is the elapsed time from when the input command is submitted to the server until the server receives the complete response. The time an `http_request` spends waiting for a connection to be established is counted as time spent waiting.
- ▶ **Time Executing Commands** – Total time spent in executing `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, and `emulate` commands. This measurement is provided both in seconds and as a percentage of active time. The time spent executing SQL commands is defined as the elapsed time from when the SQL statements are submitted to the server until these statements have completed. The time spent executing TUXEDO commands is defined as the time to execute the specific ATMI primitive until it succeeds or fails.
- ▶ **Time Spent in Input** – Total time spent sending user input to the server. This measurement is provided both in seconds and as a percentage of active time. The time spent by `http_request` and `sock_send` commands in sending input to the server is reported as time spent in input.
- ▶ **Time Spent Thinking** – Total time spent thinking, both in seconds and as a percentage of active time. The time spent thinking for a given command is the elapsed time from the end of the preceding emulation command until the current emulation command is submitted to the server. This definition of think time corresponds with that used during the run only if the environment variable `Think_def` in the script has the default LR (last received), which assumes that think time starts at the last received data timestamp of the previous response.

If any SQL emulation commands were executed, the Usage report output includes:

- ▶ **Rows Received** – Number of rows received by all reported `sqlnrecv` commands.
- ▶ **Received Rows/Sec** – Average number of rows received per second. Derived by dividing the number of rows received by the active time
- ▶ **Average Rows/Response** – Average number of rows in the passed and failed responses. Derived by dividing the number of rows received by the number of passed and failed responses.
- ▶ **Average Think Time** – Average think time in seconds for `sqlexec` and `sqlprepare` statements only.
- ▶ **SQL Execution Commands** – Number of `sqlexec` commands reported.
- ▶ **Preparation Commands** – Number of `sqlprepare` commands reported.
- ▶ **Rows Processed** – Number of rows processed by all reported `sqlexec` commands.
- ▶ **Processed Rows/Sec** – Average number of rows processed per second. Derived by dividing the number of rows processed by the active time.
- ▶ **Avg Rows/Execute Cmd** – Average number of rows processed by each `sqlexec` command. Derived by dividing the number of rows processed by the number of `sqlexec` commands reported.
- ▶ **Avg Row Process Time** – Average time in milliseconds for processing a row by an `sqlexec` command. Derived by dividing the time spent on `sqlexec` commands by the number of rows processed.
- ▶ **Avg Execution Time** – Average time in milliseconds to execute an `sqlexec` command. Derived by dividing the time spent on `sqlexec` commands by the number of `sqlexec` commands.
- ▶ **Avg Preparation Time** – Average time in milliseconds to execute an `sqlprepare` command. Derived by dividing the time spent on `sqlprepare` commands by the number of `sqlprepare` commands.

If any HTTP emulation commands were executed, the Usage report output includes:

- ▶ **Passed HTTP Connections** – Number of successful HTTP connections established by all reported `http_request` commands.
- ▶ **Failed HTTP Connections** – Number of HTTP connection attempts that failed to establish a connection for all reported `http_request` commands.

- ▶ **HTTP Sent Kbytes** – Kilobytes of data sent by reported `http_request` and `commands`.
- ▶ **HTTP Received Kbytes** – Kilobytes of data received by reported `http_nrecv` and `http_recv` commands.
- ▶ **Sent Kbytes/Connection** – Kilobytes of data sent by reported `http_request` and `commands` per connection. Derived by dividing the kilobytes of data sent by the number of successfully established HTTP connections.
- ▶ **Passed Connections/Min** – Number of successful HTTP connections established per minute. Derived by dividing the number of successful HTTP connections by the active time.
- ▶ **Avg Connect Setup Time** – Average time, in milliseconds, required to establish a successful HTTP connection. Derived by dividing the total connection time for all recorded `http_request` commands by the number of successful connections.
- ▶ **HTTP Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `http_request` and `commands` by the active time.
- ▶ **HTTP Recv Kbytes/Sec** – Kilobytes of data received per second. Derived by dividing the kilobytes of data received by all recorded `http_nrecv` and `http_recv` commands by the active time.
- ▶ **Recv Kbytes/Connection** – Kilobytes of data received by reported `http_nrecv` and `http_recv` commands per connection. Derived by dividing the kilobytes of data received by the number of successfully established HTTP connections.

If any socket emulation commands were executed, the U sage report output includes:

- ▶ **Passed socket Connections** – Number of successful socket connections established by all reported `sock_connect` functions.
- ▶ **Socket Sent Kbytes** – Kilobytes of data sent by reported `sock_send` commands.
- ▶ **Socket Received Kbytes** – Kilobytes of data received by reported `sock_nrecv` and `sock_recv` commands.
- ▶ **Passed Connections/Min** – Number of successful socket connections established per minute. Derived by dividing the number of successful socket connections by the active time.
- ▶ **Socket Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `sock_send` commands by the active time.

Analyzing Results

- ▶ **Socket Recv Kbytes/Sec** – Kilobytes of data received per second. Derived by dividing the kilobytes of data received by all recorded `sock_nrecv` and `sock_recv` commands by the active time.

If any `TUXEDO` emulation commands were executed, the Usage report output includes:

- ▶ **Tuxedo Execution Commands** – Number of `TUXEDO` commands reported.
- ▶ **Avg Execution Time** – Average time in milliseconds to execute a `TUXEDO` command. Derived by dividing the time spent on `TUXEDO` commands by the number of `TUXEDO` commands.

If any `start_time` emulation commands were executed, the Usage report output includes:

- ▶ **stop_time Commands** – Number of `stop_time` commands reported.
- ▶ **stop_time Cmds/Min** – Number of `stop_time` commands per minute. Derived by dividing the number of `stop_time` commands by the active time.
- ▶ **start_time Commands** – Number of `start_time` commands reported.
- ▶ **Avg Block Time** – Average response time in seconds for reported `stop_time` commands. Derived by dividing the sum of the response times for all `stop_time` commands by the number of `stop_time` commands. The response time of a `stop_time` command is the elapsed time between it and its associated `start_time` command.

If any `emulate` emulation commands were executed, the Usage report output includes:

- ▶ **Passed emulate Commands** – Number of `emulate` commands that report a passed status.
- ▶ **Passed emulate Time Spent** – Total time spent, from when the passed `emulate` commands start to where they end.
- ▶ **Failed emulate Commands** – Number of `emulate` commands that report a failed status.
- ▶ **Failed emulate Time Spent** – Total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Usage report output includes:

- ▶ **Passed testcase Commands** – Number of `testcase` commands that report a passed status.

- ▶ **Failed testcase Commands** – Number of `testcase` commands that report a failed status.

Summary Statistics

- ▶ **Duration of Run** – Elapsed time from the beginning to the end of the run. The beginning of the run is the time of the first emulation activity among all users and scripts, not just the ones you have filtered for this report. Similarly, the end of the run is the time of the last emulation activity among all users and scripts.
- ▶ **Passed Commands, Failed Commands, Passed Responses, Failed Responses** – Identical to their counterparts in *Cumulative Statistics* on page 9-52.
- ▶ **Total Throughput** – Four measurements of total throughput are provided: passed command throughput, failed command throughput, passed response throughput, and failed response throughput. The total throughput of passed commands is obtained by dividing the number of passed commands by the run's duration, with the appropriate conversion of seconds into minutes. Thus, it represents the total passed command throughput by all selected users at the applied workload, as opposed to the throughput of the average user. The total failed command, and the total passed and failed response throughputs are calculated analogously.

These throughput measurements, as well as the script throughput, depend upon the user and script selections. For example, if only three users from a ten-user run are selected, the throughput would not represent the server throughput at a ten-user workload, but rather the throughput of three selected users as part of a ten-user workload. As a guideline, the summary throughput measurements are most meaningful when all users and scripts are selected.

- ▶ **Number of Users** – Number of virtual users in the schedule run.
- ▶ **Number of stop_time Cmds** – Number of `stop_time` commands in the schedule run.
- ▶ **Number of Completed Scripts** – Scripts are considered complete if all activities associated with the script are completed before the run ends.
- ▶ **Number of Uncompleted Scripts** – Number of scripts that have not finished executing when a run is halted. Scripts can be incomplete if you halt the run or set the schedule to terminate after a certain number of users or scripts.
- ▶ **Average Number of Scripts Completed per User** – Calculated by dividing the number of completed scripts by the number of users.
- ▶ **Average Script Duration for Completed Scripts** – Average elapsed time of a completed script. Calculated by dividing the cumulative active time of all users and scripts by the number of completed scripts.

- ▶ **Script Throughput for Completed Scripts** – Number of scripts-per-hour completed by the server during the run. Calculated by dividing the number of completed scripts by the duration of the run, with the conversion of seconds into hours. This value changes if you have filtered users and scripts.

If any `stop_time` emulation commands were executed, the Usage report output includes:

- ▶ **Avg Number of stop_time Commands** – Calculated by dividing the number of `stop_time` commands by the number of users.
- ▶ **Average start_time/stop_time Duration** – Average response time in seconds for reported `stop_time` commands. Derived by dividing the sum of the response times for all `stop_time` commands by the number of `stop_time` commands. The response time of a `stop_time` command is the elapsed time between it and its associated `start_time` command.
- ▶ **stop_time Command Throughput for all Users** – Number of `stop_time` commands executed per minute during the schedule run. Derived by dividing the number of `stop_time` commands by the duration of the run.

If any `emulate` emulation commands were executed, the Usage report output includes:

- ▶ **Passed emulate Commands** – Number of `emulate` commands that report a passed status.
- ▶ **Passed emulate Time Spent** – Total time spent, from when the passed `emulate` commands start to where they end.
- ▶ **Failed emulate Commands** – Number of `emulate` commands that report a failed status.
- ▶ **Failed emulate Time Spent** – Total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Usage report output includes:

- ▶ **Passed testcase Commands** – Number of `testcase` commands that report a passed status.
- ▶ **Failed testcase Commands** – Number of `testcase` commands that report a failed status.

Working With Toolbars

Each component of Rational Robot has two default toolbars:

- ▶ **Standard** – Contains buttons for choosing the most frequently used commands for that component.
- ▶ **Tools** – Contains buttons for choosing other components.

LoadTest has two additional toolbars: Schedule and Run.

All toolbar buttons correspond to menu commands. Click a toolbar button to immediately access the menu command. The toolbar buttons are dynamic. This means that some toolbar buttons are enabled only when you select related menu or toolbar commands.

Viewing Information About Toolbar Buttons

There are several ways to view information about a toolbar button and its corresponding menu command:

- ▶ To see the name of the button in a yellow ToolTip, point to the button and pause.
- ▶ To see a brief description in the status bar, point to the button or menu command.
- ▶ To see more detailed information about the button or menu command, do one of the following:
 - Point to the button or menu command and press **F1**.
 - Click the **Help Pointer** icon on the right side of the Standard toolbar, and then point to the button or menu command and click the mouse button.



Displaying Toolbars

To display or hide a toolbar:

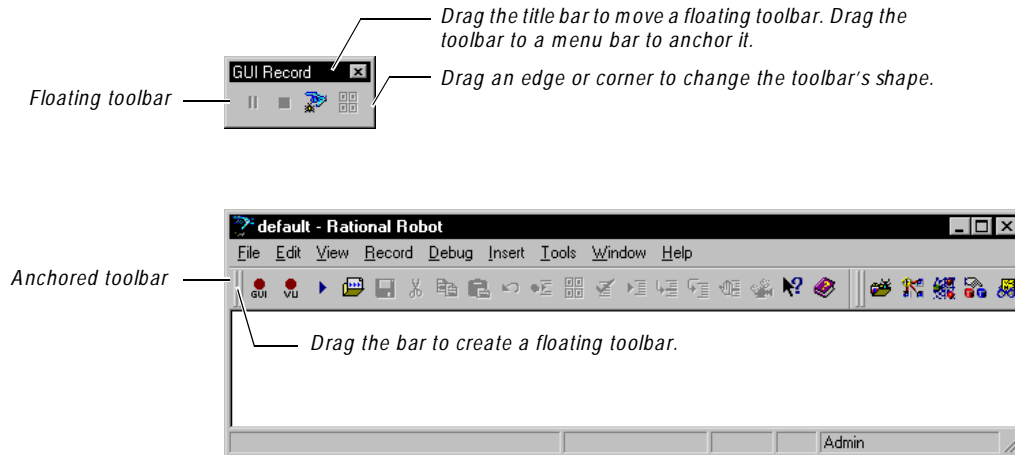
- ▶ Click **View** → **Toolbars**, and then click the name of the toolbar you want to display or hide.

A check mark appears in front of the name of each displayed toolbar.

Anchoring and Floating Toolbars

The Standard and Tools toolbars are anchored (docked) within each component's main window below the menu bar. However, you can drag an anchored toolbar from within a window and make it a floating (undocked) toolbar, which you can position and resize independently of the main window. When you do this, the toolbar remains visible even when you minimize a component or when the component is hidden behind another application. You can also drag a floating toolbar and anchor it inside of the window.

Floating toolbars are always on top of all other windows. This ensures that they are never hidden. The following figure shows a floating toolbar and an anchored toolbar.



Setting Toolbar Options

To set the toolbar options:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. In the **Toolbars** tab, select or clear the appropriate check boxes:
 - Show ToolTips** – Displays a ToolTip when you point to a button and pause.
 - Cool Look** – Changes the appearance of the toolbar buttons so that they have no borders. It does not change the behavior of the buttons.
 - Large Buttons** – Changes the size of the toolbar buttons.
3. Click **OK**.

Adding, Deleting, and Moving Toolbar Buttons

To add, delete or move a toolbar button:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. Click the **Commands** tab.
3. To add a button, click a menu name from the **Categories** list. Each name in the list represents a menu in the menu bar. Click a button to see its description. Drag the button to the toolbar. Make sure you release the mouse button within the toolbar.
4. To delete a button, drag it anywhere outside the toolbar.
5. To move a button, drag it to a new location.
6. Click **OK**.

Creating Your Own Toolbar

To create a custom toolbar that contains just the buttons you want:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. Click the **Toolbars** tab.
3. Click **New**.
4. Type the name for the new toolbar and click **OK**.
5. Click the **Commands** tab.
6. Click a menu name from **Categories**.
7. Click a button to see its description. Drag the button to the new toolbar. Make sure you release the mouse button within the new toolbar.
8. Repeat steps 6 and 7 until you have finished adding buttons.
9. Click **OK**.

Resetting and Deleting Toolbars

To restore a default toolbar to its original configuration or to delete a custom toolbar:

1. Click **View** → **Toolbars** → **Customize**, or right-click a toolbar and click **Customize**.
2. In the **Toolbars** tab, do one of the following:
 - To reset a default toolbar to its original configuration, highlight the toolbar in the list and click **Reset**.
 - To delete a custom toolbar, highlight the toolbar in the list and click **Delete**.

The name of this button changes depending on the type of toolbar that is selected.

3. Click **OK**.

▶▶▶ A P P E N D I X B

Configuring Master and Agent Computers

If your schedule runs a large number of users, you must set certain system environment variables for the run to complete successfully.

Running More Than 245 Users

If your schedule runs more than 245 users total, you must change two settings in the NuTCRACKER operating environment on the Master computer. To run more than 245 users on an NT Agent computer, you must make the same changes on that Agent.

To change these settings:

1. Click **Start** → **Settings** → **Control Panel** → **Nutcracker**.
2. Click the **NuTC 4 Options** tab.
3. Select **Semaphore Settings** from the Category list.
4. Change **Max Number of Semaphores** to $N + S + 10$, where N is the number of users you want to run and S is the number of shared variables used by VU scripts in the schedule.
5. Repeat for **Max Number of Semaphores Per ID**.
6. Click **OK**.
7. Click **Restart Later**.
8. Restart NT.

Running More Than 1000 Users

If your schedule runs more than 1000 users total, you must create an environment variable that sets the minimum shared memory size on the Master computer. To run more than 1000 users on an NT Agent computer, you must make the same changes on that Agent.

To create and set this environment variable:

1. Click **Start** → **Settings** → **Control Panel** → **System**.
2. Click the **Environment** tab.
3. Create an environment variable named `TEST7_LTMASTER_SHM_MINSZ`, and set its value to $700 * N$, where N is the number of users you want to run.

On the Master computer, N is the total number of users for the entire run. On the Agent computer, N is the number of users that run on that Agent.

4. Click **Set**, then click **OK**.
5. Restart NT.

Running More Than 1000 Users on One NT Computer

If your schedule runs more than 1000 users on an NT computer, you must create and set a system environment variable on each NT computer running more than 1000 users.

To create and set this environment variable:

1. Click **Start** → **Settings** → **Control Panel** → **System**.
2. Click the **Environment** tab.
3. Create an environment variable named `TEST7_LTMASTER_NTUSERLIMIT`, and set its value to the number of users you want to run.
4. Click **Set**, then click **OK**.
5. Restart LoadTest (on the Master computer) or the PerformanceStudio Agent (on the Agent computer) for the new setting to take effect on that computer.

Running More Than 24 Users on a UNIX Agent

If your schedule runs more than 24 users on a UNIX Agent computer, you must set the following system environment variables:

System Environment Variable	Value
Total LoadTest processes (NPROC, MAXUP)	The number of virtual users on the Agent+ 5.
Total open files (NFILE, NINODE)	$(6 * N) + (open_files * N) + (connections * N)$ <i>N</i> is the number of virtual users on the Agent. <i>open_files</i> is the number of files explicitly opened within scripts. <i>connections</i> is the number of connections open concurrently.
Total shared memory (SHMALL)	$724 + 5609N + 16S + 13G + group_names$ bytes <i>N</i> is the number of virtual users on the Agent. <i>S</i> is the total number of shared variables in all the scripts in the schedule. <i>G</i> is the total number of user groups in the schedule. <i>group_names</i> is the total length of all user group names in the schedule.
Semaphore set IDs (SEMMNI, SEMMAP)	1
Total semaphores (SEMNS)	The number of virtual users on the Agent
Semaphores per set (SEMMSL)	The number of virtual users on the Agent

Controlling TCP Port Numbers

The psmstr_v and psmstr_s network services have been added to control the ports on the Master computer to which the Agent communication software connect. These network services allow tests to be run with Master and Agent computers on different networks separated by a firewall by controlling the ports at which the listening Master server processes bind to.

In a PerformanceStudio run involving Agents there are multiple socket connections between the Master and each Agent.

Connections made from the Master to the Agent are always made to a single well-known port, on which the PerformanceStudio Agent is listening. This port defaults to 8800.

There are two connections made from each Agent to the Master, one to a Master server process named `sqa7vsrv` and another to a Master server process named `sqa7ssrv`. These two server processes each listen on a separate port. They do not bind to a specific port, but instead let the Master's operating system choose a port dynamically. The Master then communicates these port values to the Agent during run initialization. (Note that all Agents connect to the same two ports on the Master.) It is these two dynamically chosen ports on the Master which cause firewall administration problems because which two ports will be used cannot be determined in advance.

Control is achieved using the optional presence of network services (the traditional TCP/UDP network services defined in an `/etc/services` file, not to be confused with an NT service). On NT, the services file is found in `Drive:\WINNT\system32\drivers\etc\services`. There is one entry per line, listing the service name, the port number, and the protocol (TCP or UDP).

Specifically, control over the ports is provided as follows:

`sqa7vsrv` binds to the port (in priority order):

1. The value of the TCP service named `psmstr_v`, if defined.

Otherwise,

2. A port dynamically chosen by the system.

`sqa7ssrv` binds to the port (in priority order):

1. The value of the TCP service named `psmstr_s`, if defined.

Otherwise,

2. A port dynamically chosen by the system.

Note that the ports defined by these two services are independent. That is, they do not need to be adjacent, nor related to the well-known PerformanceStudio Agent port of 8800. They do need to be unique. We suggest using the ports 8801 and 8802 if they are not used for some other service on the Master.

For example, if you want the ports on the Master to be 8801 and 8802, add two lines to the services file:

```
psmstr_s8801/tcp# PerformanceStudio Master S server
psmstr_v8802/tcp# PerformanceStudio Master V server
```


In addition, the `psagent` network service has been added to control the port at which the PerformanceStudio Agent listens. If the well-known Agent port of 8800 is already in use by another application on one or more Agent computers, an alternate port needs to be specified using the `psagent` service.

The `psagent` service is put in the services file in the same way that the network services `psmstr_v` and `psmstr_s` are put in the file. The difference is that the `psagent` service must be defined on the Master and all Agents used in the testing run, and must be identical for all systems. The Agents must be rebooted after altering the service file.

For example, if you want the Agent to list on port 8888, add a line to the services file on both the Master and the Agent:

```
psagent8888/tcp# PerformanceStudio Agent
```

Setting Up IP Aliasing

LoadTest provides IP aliasing, which allows many IP addresses to be assigned to the same physical system. Every virtual user can be assigned a different IP address to realistically emulate your user community. The requests generated by these virtual users receive responses back from the Web server with timing characteristics and validation recorded intact.

To use IP aliasing on any particular computer, the system administrator must set up the IP addresses on that system.

For Windows NT, this can be done with the **Settings** → **Control Panel** → **Network** → **Protocols** → **TCP/IP Protocol** → **Properties** → **Advanced** → **IP Addresses** → **Add** button.

For UNIX, this can be done with the `ifconfig(1)` command line utility. See the `ifconfig` manual pages for specific details appropriate to that operating system. To set up large numbers of IP addresses it is convenient to use a Perl or UNIX shell script. An example Korn shell script for this purpose named `ipalias_setup` can be found in the `bin` directory of UNIX Agent installs. (You must have root privileges to set up IP aliases with `ifconfig`.)

Be careful when assigning IP addresses to a computer for obvious reasons, such as conflicting with existing IP addresses as well as routing considerations. We recommend that IP addresses be assigned by a qualified network administrator.

After IP Aliasing is set up, simply open a schedule, click the **Runtime** button, and select the **Enable IP Aliasing** check box.

If IP Aliasing is selected in the schedule, at the beginning of a run LoadTest software on each computer (Master or Agent) queries the system for all available IP addresses. Each VU user scheduled to run on that computer is assigned an IP address from that list, in round-robin fashion. In other words, if there are more VU users on a computer than IP addresses, an IP address is assigned to multiple VU users. If there are fewer VU users than IP addresses, some IP addresses are not used. This approach optimizes the distribution of IP addresses regardless of the number of users scheduled on any particular computer, and frees you from involvement or worry about matching IP addresses to specific users.

▶ ▶ ▶ A P P E N D I X C

Standard Datapool Data Types

This appendix contains:

- ▶ A table of standard data types
- ▶ A table of minimum and maximum ranges for the standard data types

Standard Data Type Table

Data types supply datapool columns with their values. You assign data types to datapool columns when you define the columns in the Datapool Specification dialog box.

The standard data types listed in the following table are included with your Rational Test software. Use these data types to help populate the datapools you create.

The standard data types (plus any user-defined data types you create) are listed in the Datapool Specification dialog box under the heading **Type**. **Type** and the other datapool column definitions (such as **Length** and **Interval**) referenced in the following table are some of the definitions you set in this dialog box.

Note that related data types (such as cities and states) are designed to supply appropriate pairings of values in a given datapool row. For example, if the Cities - U.S. data type supplies the value Boston to a row, the State Abbrev. - U.S. data type supplies the value MA to the row.

Standard Datapool Data Types

Standard data type name	Description	Examples
Address - Street	Street numbers and names. No period after abbreviations.	20 Maguire Road 860 S Los Angeles St 8th Fl 75 Wall St 22nd Fl
Cities - U.S.	Names of U.S. cities.	Lexington Cupertino Raleigh
Company Name	Company names (including designations such as Co and Inc where appropriate).	Rational Software Corp TSC Div Harper Lloyd Inc Sofinnova Inc
Date - Aug 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma (,) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>Oct 8, 1997 Jun 17, 1964 Nov 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"Oct 8, 1997" "Jun 17, 1964" "Nov 10, 1978"</p>
Date - August 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma (,) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>October 8, 1997 June 17, 1964 November 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"October 8, 1997" "June 17, 1964" "November 10, 1978"</p>

(Continued)

Standard data type name	Description	Examples
Date - MM/DD/YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>10/08/97 06/17/64 11/10/78</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows:</p> <p>"10/08/97" "06/17/64" "11/10/78"</p>
Date - MM/DD/YYYY	<p>Dates in the format shown.</p> <p>To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.</p>	<p>10/08/1997 06/17/1964 11/10/1978</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows:</p> <p>"10/08/1997" "06/17/1964" "11/10/1978"</p>
Date - MMDDYY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>100897 061764 111078</p>
Date - MM-DD-YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in Maximum must be greater than the year in Minimum).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 010100 and Maximum to 123199.</p>	<p>10-08-97 06-17-64 11-10-78</p>

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Date - MMDDYYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.	10081997 06171964 11101978
Date - MM-DD-YYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 01011900 and Maximum to 12312050.	10-08-1997 06-17-1964 11-10-1978
Date - YYYY/MM/DD	Dates in the format shown. To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 19000101 and Maximum to 20501231.	1997/10/08 1964/06/17 1978/11/10 If the slash is the delimiter, the values are stored in the datapool as follows: "1997/10/08" "1964/06/17" "1978/11/10"
Date - YYYYMMDD	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 19000101 and Maximum to 20501231.	19971008 19640617 19781110
Date, Julian - DDDYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 00100 and Maximum to 36599.	28197 16964 31478
Date, Julian - DDDYYYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 0011900 and Maximum to 3652050.	2811997 1691964 3141978

(Continued)

Standard data type name	Description	Examples
Date, Julian - YYDDD	<p>Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032.</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set Minimum to 00001 and Maximum to 99365.</p>	<p>97281 64169 78314</p>
Date, Julian - YYYYDDD	<p>Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set Minimum to 1900001 and Maximum to 2050365.</p>	<p>1997281 1964169 1978314</p>
Float - X.XXX	<p>Positive and negative decimal numbers in the format shown.</p> <p>Set Length to the number of decimal places to allow (up to 6).</p> <p>Set Minimum and Maximum to the range of numbers to generate.</p> <p>To generate numbers with more than 9 digits (the maximum allowed with the Integers - Signed data type), use the Float - X.XXX data type and set Decimals to 0.</p>	<p>243.63918 -95.99 155075028157503</p>
Float - X.XXXE+ NN	<p>Positive and negative decimal numbers in the exponential notation format shown.</p> <p>Set Length to the number of decimal places to allow (up to 6).</p> <p>Set Minimum and Maximum to the range of numbers to generate.</p>	<p>4.0285177E+ 068 -3.2381443E+ 024 8.8373255E+ 119</p>
Gender	<p>Either M or F, with no following period.</p>	<p>M F</p>
Hexadecimal	<p>Hexadecimal numbers.</p>	<p>1d6b77 ff 3824e7d</p>

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Integers - Signed	<p>Positive and negative whole numbers. This is the default data type.</p> <p>To include negative numbers in the list of generated values, set Minimum to the lowest negative number you want to allow.</p> <p>Maximum range:</p> <ul style="list-style-type: none"> ▶ Minimum = -999999999 (-999,999,999) ▶ Maximum = 999999999 (999,999,999) <p>For larger numbers, use a float data type.</p> <p>If you do not specify a range, the default range is 0 through 999,999,999.</p> <p>Use this data type to generate unique data in a datapool column (for example, when you need a “key” field of unique data). You can also use Read From File and user-defined data types to generate unique data.</p>	<p>1349 -392993 441393316</p>
Name - Middle	<p>Masculine and feminine middle names.</p> <p>If the middle name is preceded by a field with masculine or feminine value (such as a masculine or feminine first name), the middle name is in the same gender category as the earlier field.</p>	<p>Richard Theresa Julius</p>
Name - Prefix (e.g., Mr)	<p>Mr or Ms, with no following period.</p> <p>If the name prefix is preceded by a field with masculine or feminine value (such as a masculine or feminine gender designation), the name prefix is in the same gender category as the earlier field.</p>	<p>Mr Ms</p>
Names - First	<p>Masculine and feminine first names.</p> <p>If the first name is preceded by a field with masculine or feminine value (such as a masculine or feminine name prefix), the first name is in the same gender category as the earlier field.</p>	<p>Richard Theresa Julius</p>
Names - Last	<p>Surnames.</p>	<p>Swidler Larned Buckingham</p>

(Continued)

Standard data type name	Description	Examples
Names - Middle Initial	Middle initials only, with no following period.	B M L
Packed Decimal	A number where each digit is represented by four bits. Digits are non-printable. Note that commas and other characters that may be used to represent a packed decimal number may cause unpredictable results when the datapool file is read.	Non-printable digits.
Phone - 10 Digit	Telephone area codes, appropriate exchanges, and numbers.	7816762400 4123818993 5052658498
Phone - Area Code	Telephone area codes. To generate correct area code lengths, set Length to 3.	781 412 505
Phone - Exchange	Telephone exchanges. To generate correct exchange lengths, set Length to 3.	676 381 265
Phone - Suffix	Four-digit telephone numbers (telephone numbers without area code or exchange). To generate correct telephone number suffix lengths, set Length to 4.	2400 8993 8498
Random Alphabetic String	Strings of random upper case and lower case letters. Length determines the number of characters generated.	AQSEFuOZUIUpAGsEM DESieAiRFiEqiEIDiicEw edEIDiIciseWsDIEdGP
Random Alphanumeric String	Strings of random upper case and lower case letters and digits. Length determines the number of characters generated.	AYcHI8WmeMeM0AK4 HSk9vGAQU79esDE 7Eeis93k4ELXie7S32siDI4E

Standard Datapool Data Types

(Continued)

Standard data type name	Description	Examples
Read From File	<p>Assigns values from an ASCII text file to the datapool column. For example, you could export a database column to a text file, and then use this data type to assign the values in the file to a datapool column.</p> <p>You can use this data type to generate unique data. You can also use the Integers - Signed and user-defined data types to generate unique data.</p> <p>For information about using this data type, see <i>Creating a Column of Values Outside Rational Test</i> on page 6-48.</p>	Any values in an ASCII text file.
Space Character	An empty string.	""
State Abbrev. - U.S.	Two-character state abbreviations.	MA CA NC
String Constant	A constant with the value of Seed . The datapool column is filled with this one alphanumeric value.	1234 AAA 1b1b
Time - HH.MM.SS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	00.00.00 (midnight) 11.14.38 21.44.19
Time - HH:MM:SS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To include the colons (:) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	00:00:00 (midnight) 11:14:38 21:44:19 If the colon is the delimiter, the values are stored in the datapool as follows: "00:00:00" (midnight) "11:14:38" "21:44:19"
Time - HHMMSS	<p>Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).</p> <p>To set a range of times from midnight to 2 pm, set Minimum to 0 and Maximum to 140000.</p>	000000 (midnight) 111438 214419

(Continued)

Standard data type name	Description	Examples
Zip Code - 5 Digit	Five-digit U.S. postal zip codes. To generate the correct zip code lengths, set Length to 5.	02173 95401 84104
Zip Code - 9 Digit	Nine-digit U.S. postal zip codes.	021733104 954012694 841040190
Zip Code - 9 Digit with Dash	Nine-digit U.S. postal zip codes with a dash between the fifth and sixth digits.	02173-3104 95401-2694 84104-0190
Zoned Decimal	Zoned decimal numbers.	3086036 450 499658196

Data Type Ranges

These are the minimum and maximum ranges for the standard data types:

Type of range	Limitation
Maximum hours	23
Maximum minutes	59
Maximum seconds	59
Maximum two-digit year	99
Maximum four-digit year	9999
Maximum months	12
Minimum six-digit date	010100 (January 1, 00)
Maximum six-digit date	123199 (December 31, 9999)
Minimum eight-digit date	01010000 (January 1, 0000)
Maximum eight-digit date	12319999 (December 31, 9999)
Minimum negative integer (Integers - Signed)	-99999999 (-999,999,999)

Standard Datapool Data Types

(Continued)

Type of range	Limitation
Maximum positive integer (Integers - Signed)	999999999 (999,999,999)
Maximum decimal places (Float data types)	6
Male/Female title	Mr, Ms
Gender designation	M, F

Glossary

action object – In TestFactory, an object in the application map that represents an action to which a control in the application responds. Typical actions are mouse left-click, mouse right-click, and mouse left-double-click; the corresponding action objects in the application map are LeftClick, RightClick, and LeftDoubleClick.

ActiveX control – A reusable software control that takes advantage of Object Linking and Embedding (OLE) and Component Object Modeling (COM) technologies. Developers can use ActiveX controls to add specialized functions to applications, software development tools, and Web pages. Robot can test ActiveX controls in applications.

actual results – In a functional test, the outcome of testing an object through a verification point in a GUI script. Actual results that vary from the recorded baseline results are defects or intentional changes in the application. See also *baseline results*.

Administrator – See *Rational Administrator*.

Agent computer – In LoadTest, a computer that has the Rational Agent software installed and that plays back a virtual user or GUI script. In a LoadTest schedule, you can identify the Agent computer on which to run a script. See also *Rational Agent*.

API recording – In Robot, a virtual user recording method that captures API calls between a specific client application and a server. These calls are captured on the client computer.

application map – In TestFactory, a hierarchical list of controls and actions in the application-under-test, as well as the states of the application-under-test and the transitions between those states. An application map can include UI objects and action objects, as well as TestFactory objects such as Pilots, Test Suites, and scripts.

application-under-test – The software being tested. See also *system-under-test*.

Asset Browser – A window that displays testing resources such as builds, queries, scripts, schedules, reports, report output, and logs. The Asset Browser is available in TestManager and LoadTest.

AUT – See *application-under-test*.

automated testing – A testing technique in which you use software tools to replace repetitive and error-prone manual work. Automated testing saves time and enables a reliable, predictable, and accurate testing process.

AutoPilot – In TestFactory, a tool for running scripts, Test Suites, and Pilots. The scripts and Test Suites can run on your local computer or on computers in the Test Lab. The Pilots run on your local computer, and the scripts they generate can run on your local computer or on computers in the Test Lab.

base state – In TestFactory, the known, stable state in which you expect the application-under-test to be at the start of each script segment. See also *script segment*.

baseline results – In a functional test, the outcome of testing an object through a verification point in a GUI script. The baseline results become the expected state of the object during playback of the script. Actual test results that vary from the baseline results are defects or intentional changes in the application. See also *actual results*.

best script – In TestFactory, an optimized script generated by a Pilot. A best script contains the fewest number of script segments that provide the most coverage of the source code or user interface in the application-under-test.

breakpoint – A feature of the Robot debugger. When you assign a breakpoint to a line of code, and then run the script in the debugger environment, the script stops executing at that line of code. Control returns to you, and the breakpoint line is displayed. From here you can view variables, perform other debugging activities, and continue executing the script.

build – A version of the application-under-test. Typically, developers add new features or enhancements to each incremental build. As team members test a build, they enter defects against those features that do not behave as expected. You use TestManager to define and manage builds.

built-in data test – A data test that comes with Robot and is used with the Object Data verification point. A data test uses a specific property of the object, in conjunction with other parameters, to determine the data to capture. Although built-in data tests cannot be edited, renamed, or deleted, they can be copied and then edited, and they can be viewed. See also *custom data test*.

ClearQuest – See *Rational ClearQuest*.

client/server – An architecture for cooperative processing in which the software tasks are split between server tasks and client tasks. The client computer sends requests to the server, and the server responds.

code coverage – In TestFactory, the percentage of code that is tested by a script. This percentage is based on the portion of the code that a script touches, relative to all code in the application-under-test. A Pilot can use code coverage to determine the best script for a run. See also *UI coverage*.

command ID – In LoadTest’s VU language, an identifier for a command. Robot automatically assigns a unique command ID, composed of an alphanumeric prefix and a three-digit number, to each emulation command. Because command IDs appear in both the virtual user script and the LoadTest report output, they enable you to determine the relationship between an emulation command and its response times.

command ID prefix – In LoadTest, a prefix for a unique emulation command ID. The prefix defaults to the script name (up to the first seven characters). However, you can define the prefix in the Generator tab of the Virtual User Record Options dialog box.

custom data test – A customer-defined data test used with the Object Data verification point. A data test uses a specific property of the object, in conjunction with other parameters, to determine the data to capture. Custom data tests are created within your organization and are stored in the repositories that were active when they were created. They can be edited, renamed, and deleted. See also *built-in data test*.

data test – A test that captures the data of an object with the Object Data verification point. See also *built-in data test* and *custom data test*.

datapool – A source of test data that GUI scripts and virtual user scripts can draw from during playback. You can automatically generate datapools using TestManager, or you can import datapool data from other sources such as your database.

dependency – In LoadTest, a method of coordinating an object in a schedule with an event. For example, if the script Query is dependent upon the script Connect, then Connect must finish executing before Query can begin executing. See also *event*.

distributed architecture – Architecture in which computer systems work together and communicate with each other across LAN, WAN, or other types of networks. A client/server system is an example of distributed architecture.

distributed functional test – In LoadTest, a test that uses multiple Agent computers to execute multiple GUI scripts written in the SQABasic language.

dynamic load balancing selector – A type of selector in a LoadTest schedule. Items in the selector, such as scripts, are executed according to a weight that you set.

emulation commands – VU language statements or commands that emulate client activity, evaluate the server’s responses, and perform communication and timing operations. LoadTest stores the results of emulation commands in a log file, which you can view from the LogViewer.

emulation functions – VU language functions that emulate client activity and evaluate the server's responses. Unlike emulation commands, emulation functions do not perform communication and timing operations, and they are not logged.

environment control commands – VU language commands that let you control a virtual user's environment by changing the VU environment variables. For example, you can set the level of detail that is logged or the number of times that virtual users attempt to connect to a server.

event – An item in a LoadTest schedule upon which another item is dependent. For example, if the script Connect sets an event and the script Query depends on this event, Connect must finish executing before Query can begin executing. See also *dependency*.

external script – A script that runs a program created with any tool. You plan and run external scripts in TestManager.

fixed user group – In LoadTest, a group that contains a scalable number of users. When you create a fixed user group, you indicate the maximum number of users that you will run in the group. Typically, you use fixed user groups in functional tests, which do not add a workload to the system.

flow control statements – In the VU and SQABasic languages, statements that let you add conditional execution structures and looping structures to a script.

functional test – A test to determine whether a system functions as intended. Functional tests are performed on GUI objects and objects such as hidden DataWindows and Visual Basic hidden controls.

Grid Comparator – The Robot component for reviewing, analyzing, and editing data files for text and numeric verification points in grid formats. The Grid Comparator displays the differences between the recorded baseline data and the actual data captured during playback.

GUI script – A type of script written in the SQABasic language. It contains GUI actions such as keystrokes and mouse clicks. Typically, a GUI script also contains verification points for testing objects over successive builds of the application-under-test.

GUI user – The type of user that is emulated when a GUI script is executed. Only one GUI user at a time can run on a computer.

hidden object – An object that is not visible through the user interface. Hidden objects include objects with a visible property of False and objects with no GUI component.

IDE – Integrated Development Environment. This environment consists of a set of integrated tools that are used to develop a software application. Examples of IDEs supported by Robot include Oracle Forms, PowerBuilder, Visual Basic, and Java.

Image Comparator – The Robot component for reviewing and analyzing bitmap image files for Region Image and Window Image verification points. The Image Comparator displays differences between the recorded baseline image and the actual image captured during playback. The Image Comparator also displays unexpected active windows that appear during playback.

instrumentation – In TestFactory, the process of inserting code coverage counters into the application-under-test. These counters record how much code is executed during a script run. See also *object code instrumentation* and *source code instrumentation*.

load – See *workload*.

load balancing – See *workload balancing*.

LoadTest – See *Rational LoadTest*.

log – A repository object that contains the record of events that occur while playing back a script or running a schedule. A log includes the results of all verification points executed as well as performance data that can be used to analyze the system's performance.

LogViewer – See *Rational LogViewer*.

low-level recording – A recording mode that uses detailed mouse movements and keyboard actions to track screen coordinates and exact timing. During playback, all actions occur in real time, exactly as recorded.

manual script – A set of testing instructions to be run by a human tester. The script can consist of steps and verification points. You create manual scripts in TestManager.

Master computer – A computer that executes LoadTest. From this computer, you create, run, and monitor schedules. When the run is finished, you use it to analyze test results.

mix-ins – See *Pilot mix-ins*.

network recording – In Robot, a virtual user recording method that records packet-level traffic. This traffic is captured on the wire.

next available selector – In LoadTest schedules, a selector that distributes each item such as a script, delay, or other selector to an available computer or virtual user. This type of selector is used in a GUI schedule. The next available selector parcels out the items sequentially, based on which computers or virtual users are available.

object – An item on a screen, such as a window, dialog box, check box, label, or command button. An object has information (properties) associated with it and actions that can be performed on it. For example, information associated with the window object includes its type and size, and actions include clicking and scrolling. In some development environments, a term other than *object* is used. For example, the Java environment uses *component*, and the HTML environment uses *element*.

object code instrumentation – In TestFactory, the process of inserting code coverage counters into the executable file of the application-under-test. These counters record how much of the program a script tests. See also *instrumentation* and *source code instrumentation*.

Object-Oriented Recording® – A script recording mode that examines objects in the application-under-test at the Windows layer. Robot uses internal object names to identify objects, instead of using mouse movements or absolute screen coordinates.

Object Properties Comparator – The Robot component that you use to review, analyze, and edit the properties of objects captured by an Object Properties verification point. The Object Properties Comparator displays differences between recorded baseline data and the actual data captured during playback.

Object Scripting commands – A set of SQABasic commands for accessing an application's objects and object properties. You add Object Scripting commands manually when editing a script.

Object Testing® – A technology used by Robot to test any object in the application-under-test, including the object's properties and data. Object Testing lets you test standard Windows objects and IDE-specific objects, whether they are visible in the interface or hidden.

OCI – Object Code Insertion. The Rational technology used in TestFactory to instrument object code and measure how much of the application-under-test a script tests. See also *code coverage* and *object code instrumentation*.

performance test – A test that determines whether a multi-client system performs within user-defined standards under varying loads. Performance tests are always run from a schedule in LoadTest.

Pilot – In TestFactory, a tool for generating scripts automatically.

Pilot mix-ins – In TestFactory, a list of Pilots that are executed on a random basis during the run of a lead Pilot. Mix-ins are useful for randomly testing multiple areas of the application-under-test. To make tests more realistic, you can combine mix-ins and scenarios.

Pilot scenario – An ordered list of Pilots that are executed during the run of a Pilot. A Pilot scenario is useful for testing UI objects that need to be exercised in a specific order. To make tests more realistic, you can combine scenarios and mix-ins.

project – A collection of data, including test assets, defects, requirements, and models, that can facilitate the development and testing of one or more software components.

proxy recording – In Robot, a virtual user recording method that captures the client/server conversation on the network wire rather than on the client computer. Proxy recording allows Robot to capture network packets that are not visible to it during network recording — for example, if the client and server are in different network segments.

query – A request for information stored in the repository. A query consists of a filter and several visible attributes — the columns of data to display, the width of the column, and the sort order.

random selector – A type of selector in a LoadTest schedule. Items in the selector, such as scripts, are randomly executed. Random selectors can be with replacement, where the odds are the same, or without replacement, where the odds change with each iteration.

Rational Administrator – The component for creating and maintaining repositories, projects, users, groups, computers, and SQL Anywhere servers.

Rational Agent – The LoadTest software that resides on a shared network drive and runs on each computer where testing occurs. The entries specified in a schedule play back on the Agent computer, which reports on their progress and status as they run. See also *Agent computer*.

Rational ClearQuest – The Rational product for tracking and managing defects and change requests throughout the development process. With ClearQuest, you can manage every type of change activity associated with software development, including enhancement requests, defect reports, and documentation modifications.

Rational LoadTest – The Rational Test component for running performance, stress, scalability, multi-user, and distributed functional tests on multiple Agents connected by a network. With LoadTest, you can initiate test runs and monitor tests from a master computer that manages the test process. LoadTest is available only in Rational Suite PerformanceStudio.

Rational LogViewer – The Robot component for displaying logs, which contain the record of events that occur while playing back a script or running a schedule. Also, the component from which you start the four Comparators.

Rational PerformanceArchitect – The Rational component that lets you test the performance of COM/DCOM applications. With Rational PerformanceArchitect, you can create a Rose sequence or collaboration diagram, convert it to a virtual user script, and then use Rational Suite PerformanceStudio to edit the script and run the performance tests.

Rational repository – A database that stores application testing information, such as test requirements, scripts, and logs. All Rational Suite TestStudio and Rational Suite PerformanceStudio products and components on your computer update and retrieve data from the same connected repository. A repository can contain either a Microsoft Access or a Sybase SQL Anywhere database.

Rational RequisitePro – The Rational product for organizing, managing, and tracking the changing requirements of your system.

Rational Robot – The Rational product for recording, playing back, debugging, and editing scripts.

Rational SiteCheck – The Robot component for managing your intranet or World Wide Web site. You can use SiteCheck to visualize the structure of your Web site, and you can use it with Robot to automate Web site testing.

Rational Synchronizer – The Rational tool that ensures the consistency of data across several Rational products.

Rational TestAccelerator – An agent application that executes scripts. TestFactory uses computers running TestAccelerator as remote machines on which to run automated distributed tests.

Rational TestFactory – The Rational Test component for mapping an application-under-test and generating scripts automatically. TestFactory is available in Rational Suite TestStudio and Rational Suite PerformanceStudio.

Rational TestManager – The Robot component for managing the overall testing effort. You use it to define and store information about test documents, requirements, scripts, schedules, and sessions.

Report Layout Editor – The TestManager component for customizing the layout of reports.

repository – See *Rational repository*.

RequisitePro – See *Rational RequisitePro*.

Robot – See *Rational Robot*.

scalable user group – In LoadTest, a group that contains a varying number of users. When you create a scalable user group, you assign it a percentage of the total workload. Assume you have a scalable user group that is 50 percent of the workload. If you run a test with 10 users, the group will contain 5 users. If you run a test with 100 users, the group will contain 50 users.

scenario – In LoadTest, a modular group of scripts and other items in a schedule that is used by more than one user group. A scenario can contain scripts, delays, and synchronization points.

scenario – See *Pilot scenario*.

schedule – In LoadTest, structure that you create to specify how scripts should be played back. A schedule can contain GUI scripts and virtual user scripts, and can indicate the number of times to repeat a script and the computer on which the script will run. In performance testing, a schedule is used to create a workload. In distributed functional testing, a schedule is used to distribute scripts among various computers.

script – A set of instructions used to navigate through and test an application. You can generate scripts in a variety of ways. You can use Robot to record scripts used in functional testing and performance testing. You can also use TestManager to create and manage manual scripts, and to manage external scripts created with a third-party testing tool. A script can have properties associated with it, such as the purpose of the script and requirements for the script. See also *external script*, *GUI script*, *manual script*, and *virtual user script*.

script outline – In TestFactory, the readable version of a script. A script outline contains a description of the actions that Robot performs while running the script.

script segment – In TestFactory, a section of a script that tests a particular element of product functionality. A Pilot generates a script segment by starting the application-under-test in a base state, navigating through the part of the product that you are testing, and returning the application-under-test to the base state. See also *base state*.

seed – An initial number fed to a random number generator. Using the same seed produces the same series of random numbers. In LoadTest, you use seeds to generate think times.

selector – An item that you insert in a LoadTest schedule to indicate how often and in what order to run scripts.

sequential selector – In a LoadTest schedule, a type of selector that executes each script, delay, or other item in the same order in which it appears in the schedule.

session – In virtual user recording, one or more scripts that you record from the time you begin recording until the time you stop recording. Typically, the scripts in a session represent a logical flow of tasks for a particular user, with each script representing one task. For example, a session could be made up of three scripts: *login*, *testing*, and *logout*. In TestFactory, a session is the period of time that the TestFactory application or a window is open.

shared variable – An integer variable that multiple scripts and multiple virtual users can read and write to. You can see the value of a shared variable while monitoring a LoadTest schedule. For example, you can set a shared variable as a flag to end a playback session. Each script can check the flag to see if the session should end. When that flag is set, exit tasks can be performed.

shell script – A script that calls or groups several other GUI scripts and plays them back in sequence. Shell scripts provide the ability to create comprehensive tests and then store the results in a single log.

SiteCheck – See *Rational SiteCheck*.

source code instrumentation – In TestFactory, the process of inserting code into the source code of the application-under-test. This code measures how much of the source code a script tests. See also *instrumentation* and *object code instrumentation*.

SQABasic – The Robot scripting language for recording GUI actions and verifying GUI objects. SQABasic contains most of the syntax rules and core commands that are contained in the Microsoft Basic language. In addition, SQABasic has commands that are specifically designed for automated testing. See also *VU*.

stable load – In LoadTest, a condition that occurs when a specified number of virtual users have logged on to the system-under-test and are active. When the stable load criterion is met, LoadTest begins measuring the load.

streak – When running a virtual user schedule in LoadTest, a series of successes or failures for emulation commands. You can see a streak while monitoring a schedule.

structural test – A test to determine whether the structure of a Web site is consistent and complete. A structural test ensures that an application's interdependent objects are properly linked together. You perform a structural test using SiteCheck.

synchronization point – In LoadTest, a place where emulated virtual users stop and wait until all other synchronized users reach that point. When all users reach the synchronization point, they are released and continue executing.

Synchronizer – See *Rational Synchronizer*.

system tuning – In LoadTest, the process of optimizing a system's performance by changing hardware resources and software configuration parameters while using a constant workload.

system-under-test – The system being tested. This includes the computers and any software that can generate a load on the system, networks, user interfaces, CPU s, and memory. See also *application-under-test*.

test assets – The resources that facilitate the planning or development phases of the testing effort. Examples of test assets include scripts, schedules, sessions, test documents, and test requirements.

test development – The process of developing tests to verify the operation of a software application. This includes creating scripts that verify that the application-under-test functions properly. Test development lets you establish the baseline of expected behavior for the application-under-test.

test documents – Test plans, project schedules, resource requirements, and any other documents that are important to your project. You develop your test documents using your own word processing or scheduling program; you then reference the name and location of the document in TestManager. This lets members of the test and development team locate documents quickly.

Test Lab – A collection of computers on which TestAccelerator is running. In TestFactory, you can distribute the scripts associated with a Pilot, a Test Suite, or the AutoPilot to run on computers in the Test Lab. See also *Rational TestAccelerator*.

Test Suite – In TestFactory, a tool for running a collection of scripts as a group.

TestAccelerator – See *Rational TestAccelerator*.

TestFactory – See *Rational TestFactory*.

TestManager – See *Rational TestManager*.

Text Comparator – The Robot component for reviewing, analyzing, and editing data files for text and numeric verification points in any format except grids. The Text Comparator displays the differences between the recorded baseline results and the actual results.

think time – In virtual user and GUI scripts, think times are delays that simulate a user's pauses to type or think while using an application. With virtual user scripts, LoadTest calculates the think time at runtime, based on think time VU environment variables that are set in the script. You can set a maximum think time in Robot. With GUI scripts, Robot uses the actual delays captured between keystrokes, menu choices, and other actions.

transaction – In LoadTest, a logical unit of work performed against a server. For example, submitting a search query or submitting a completed form to a Web server are both transactions.

transaction rate – In LoadTest, the playback speed calculated as a function of number of transactions per unit of time. For example, if a script contains one transaction, and each script is started at half-second intervals, your transaction rate would be 2 per second.

transactor – In LoadTest, an item that you insert in a LoadTest schedule to indicate the number of user-defined transactions that a virtual user performs in a given time period.

UI coverage – In TestFactory, the percentage of objects in the application map that are tested by a Pilot-generated script. This percentage is the proportion of UI objects that the script touches, relative to all UI objects available to the Pilot. A Pilot can use UI coverage to determine the best script for a run. See also *code coverage*.

UI object properties – Attributes of object classes and UI objects that TestFactory uses to map applications and generate scripts.

unexpected active window – A window that appears during script playback that interrupts the script playback process and prevents the expected window from being active. For example, an error message generated by the application-under-test is an unexpected active window. You can view unexpected active windows in the Image Comparator.

user group – In LoadTest, a collection of users that execute similar tasks and generate the same basic workload. Accountants and data entry operators are examples of user groups.

verification – The process of comparing the test results from the current build of the software to its baseline results.

verification point – A point in an SQABasic script that confirms the state of one or more objects. During recording, a verification point captures object information from the application-under-test and stores it as the baseline. During playback, a verification point recaptures the object information and compares it to the baseline. In a manual script, a verification point is a question about the state of the application-under-test.

virtual user – In LoadTest, a type of user that is emulated when a virtual user script is executed. A computer can run multiple virtual users simultaneously.

virtual user script – A type of script written in the VU language. Virtual user scripts contain client/server requests and responses as well as user think times.

VU – The Robot scripting language for recording a client's requests to a server. VU provides most of the syntax rules and core commands available in the C programming language. In addition, VU has emulation commands and functions that are specifically designed for automated performance testing. See also *SQABasic*.

wait state – A delay or timing condition that handles time-dependent activities.

workload – In LoadTest, the set of all activities that users perform in an actual production setting of the system-under-test. You can use LoadTest to emulate a workload.

workload balancing – In LoadTest, the act of distributing activities so no one system or device becomes a bottleneck.

workload model – In LoadTest, the workload model is represented as a schedule. You can play back this schedule and analyze the response times.

▶ ▶ ▶ Index

A

Abnormal_term_cnt. *See* schedules, setting runtime information

access order of datapool rows 6-4, 6-17

- starting row number 6-18

addresses data type C-2

Agent computers 1-5, 7-6, 7-7

- changing settings of 7-37
- checking 7-60
- controlling port numbers B-3
- copying compiled VU scripts to 7-37
- copying files to Master 7-37
- location of scripts and datapool files 7-38
- monitoring resources of 8-21, 9-3, 9-27
- monitoring status of 8-21

Oracle 7-41

- preferred user view 8-12
- removing files from 7-37
- running schedule items on next available 1-10, 2-20, 7-15

Sybase 7-41

TUXEDO 7-42

AIX Agents 7-41

Analog reports 3-12, 3-14, 9-3, 9-38

- including passed and failed responses 9-23, 9-25

analyzing results 2-15, 9-2

API recording 3-2, 3-4

- monitoring feedback during 4-7
- starting applications 3-25

applications, starting when recording 3-25

ASCII text files 6-48

Asset Browser

- copying reports with 9-8
- deleting reports with 9-11
- opening schedule through 7-29
- renaming reports with 9-10
- viewing schedule through 7-50

Authentication Datapool 3-29

- features 3-32
- modifying with Robot during recording 3-31
- modifying with TestManager 3-30
- when to modify 3-30

automatic protocol filtering 3-17

automatic timing

- blocks 2-13, 5-4
- emulation commands 5-1

automatically generating values for user-defined data types 6-39

autonaming virtual user scripts and sessions 3-25

axes, inverting in graphs 9-31

B

blocks 2-13, 5-4

- adding during recording 5-5

- nesting 5-6

- reporting average time 9-56

Index

C

cached responses, in HTTP scripts 3-20

checking

- Agent computers 7-60

- schedules 7-59

cities data type C-2

Cleanup_time. *See* schedules, terminating

client computers

- associating with a server 3-8

- defining for network or proxy recording 3-27

- performance tests and 1-8

- removing 3-29

- response times 1-11, 2-1

- selecting for network recording 3-4

- think time distribution and 2-14

client requests

- monitoring during recording 4-7

- recording 1-6, 4-3

client/server pairs

- deleting 3-34, 3-35

- identifying for proxy recording 3-8

- reassociating with a proxy 3-35

columns in datapools

- adding 6-20, 6-22

- assigning data types to 6-28

- assigning values from a text file 6-48, C-8

- configuring through the script 6-18

- defining 6-19

- deleting 6-33

- editing column definitions, in Robot 6-21

- editing column definitions, in TestManager 6-32

- editing values, in Robot 6-23

- editing values, in TestManager 6-34

- example of column definition 6-30

- field values and 6-45

- including or excluding 6-18

- length of 6-29

- maximum number 6-4, 6-13, 6-20, 6-26, 6-48

- names correspond to script variables 6-19, 6-27, 6-48

- setting numeric ranges in 6-29

- setting unique values in 6-28, 6-29

- unique 6-43

- values supplied by data types 6-9

command IDs 3-23

- filtering 9-16, 9-33

- grouping Status report output by 9-23

- in Trace reports 9-50

- prefix 3-10

- prefix for all commands 3-10

- sorting in reports 9-22

- TU XEDO prefixes 3-23

comments 2-13

- adding to scripts during editing 5-12

- adding to scripts during recording 5-12

Compact user view 8-12, 8-13

- preferred with Agent computers 8-12

- sorting users in 8-27

company names data type C-2

Compare reports 9-3, 9-13, 9-40

- absolute comparison 9-40

- defining 9-28

- graphing 9-22, 9-30

- N/A and Undefined responses 9-44

- relative comparison 9-42

- setting response ranges 9-22, 9-24

- weighted absolute comparison 9-41

- weighted relative comparison 9-43

Computer view 8-21

computers

- defining for network or proxy recording 3-27

- distributing users among 7-8, 7-36

- monitoring resources of 2-18, 7-67, 8-21, 9-3, 9-27

- removing 3-29

- Repository entry optional 7-7
- running users on 7-6, 7-7
- selecting a network interface card 3-5
- See also* Agent computers, Master computers
- configuration tests 1-10, 2-10
- connections in virtual user recording 3-17, 4-11
- constant value data type C-8
- contained scripts 4-17, 4-21
- contention tests 1-3, 1-9
- controlling port numbers B-3
- copying
 - compiled VU scripts to Agent computers 7-37
 - datapools 6-35
 - files from Agent 7-37
 - graphs 9-8
 - report output 9-8
 - reports 9-8
 - user-defined data types 6-42
 - virtual user scripts 4-25
- CPU delays, think time 3-16
- creating
 - datapools outside Rational Test 6-44
 - datapools, in Robot 6-13
 - datapools, in TestManager 6-25
 - user-defined data types 6-11
- credit card numbers 6-31
- .csv datapool files 6-3, 6-36
- .csv exported report output 2-18, 9-12
- cursors 6-4
 - disabling wrapping for unique row retrieval 6-43
 - persistent 6-16
 - private vs. shared 6-16
 - starting row number 6-18
 - wrapping 6-16
- custom histograms 8-6
 - adding groups to 8-37
 - assigning states to 8-36

- deleting groups from 8-38
- removing states from 8-37
- customer support xv
- customizing
 - histograms 8-36
 - reports 9-15, 9-22
 - views 8-27

D

- data correlation 3-21
- data types
 - assigning to a datapool column 6-28
 - copying 6-42
 - creating 6-11
 - deleting 6-42
 - determining which data types you need 6-10
 - editing values in 6-38
 - importing user-defined 6-41
 - list of standard data types C-1
 - minimum and maximum values C-9
 - renaming 6-41
 - role of 6-9
 - standard and user-defined 6-9
- DATAPPOOL_CONFIG
 - editing 6-14
 - role of 6-14
- datapools 6-6
 - access order 6-4, 6-17
 - accessing from GUI and virtual user scripts 6-24
 - adding commands to virtual user scripts 6-13
 - assigning data types to 6-28
 - Authentication Datapool 3-29
 - copying 6-35
 - creating in Robot 6-13
 - creating in TestManager 6-25
 - creating outside Rational Test 6-44
 - cursors 6-4

Index

- data types 6-9
- deleting 6-35
- deleting columns from 6-33
- editing column definitions, in Robot 6-21
- editing column definitions, in TestManager 6-32
- editing values, in Robot 6-23
- editing values, in TestManager 6-34
- example of column definition 6-30
- example of value generation 6-32
- exporting 6-37
- files 6-3, 6-36
- finding data types for 6-10
- generating data, in Robot 6-20, 6-22
- generating data, in TestManager 6-26
- importing from another project 6-37
- importing from outside Rational test 6-35
- limits 6-4
- maximum number of columns 6-4, 6-13, 6-20, 6-26, 6-48
- naming 6-16
- numeric ranges in 6-29
- persistent cursors 6-16
- planning 2-13, 2-21, 6-6
- populating with values, in Robot 6-20, 6-22
- populating with values, in TestManager 6-26
- private user access to 6-16
- renaming 6-35
- role of 6-2, 6-5
- row access order 6-4, 6-17
- script generation option 3-10
- setting unique values in 6-28, 6-29
- shared user access to 6-16, 7-63
- starting row number 6-18
- structure 6-45
- unique row retrieval 6-43
- where stored 6-3

- dates
 - Julian C-4, C-5
 - setting ranges C-2, C-3, C-4
- dates data types C-2, C-3, C-4
- debugging virtual user scripts 8-33
- decimal numbers 6-29, C-5
- default reports
 - deleting 9-11
 - restoring 9-12
- default settings
 - for users 7-38
 - monitor 8-31, 8-34
 - recording 3-1
 - reports 9-35
- delays 2-8, 7-19, 7-54
 - deleting from schedules 7-33
 - inserting into script 7-20
 - setting in schedule 7-10, 7-20
 - suppressing 7-63
 - types of 7-20
- deleting
 - client/server pairs 3-34
 - datapool column definitions 6-33
 - datapools 6-35
 - items from schedules 7-33
 - proxy computers 3-35
 - report output 9-11
 - reports 9-11
 - schedules 7-51
 - script from a session 4-22
 - scripts 4-25
 - sessions 4-25
 - user-defined data types 6-42
- dependencies 7-52
 - setting 7-54
- directives in datapools 6-18
 - overriding 6-17

distributed functional tests 1-10, 2-19, 7-7
 identifying requirements 2-20
 scheduling 2-20
 vs. performance tests 2-2

DLB_FREQ. *See* dynamic load balancing selectors

DLB_TIME. *See* dynamic load balancing selectors

dynamic data correlation 3-21

dynamic load balancing selectors 7-17

E

editing

 datapool column definitions, in Robot 6-21
 datapool column definitions, in TestManager 6-32
 datapool values, in Robot 6-23
 datapool values, in TestManager 6-34
 DATAPOOL_CONFIG 6-14
 default monitor settings 8-34
 default user settings 7-38
 in-line 7-33
 schedules 7-31, 7-34
 scripts 7-29
 user group properties 7-34
 user options 7-34
 user-defined data type definitions 6-39
 user-defined data type values 6-38

empty string data type C-8

emulate VU emulation commands 9-56, 9-58

emulated users. *See* GUI users, virtual users, users

emulation commands 3-10

 associating with block and timer names 5-5
 automatically timed 5-1

 displaying success or failure of 8-12, 8-14, 8-27

environment variables. *See* system environment variables, VU environment variables

error files

 copying to Master 7-37
 displaying 8-18, 8-33, 9-4
 location on Agent 7-38
 removing from Agent 7-37

events 7-52

 displaying state of 8-12
 how set with multiple iterations 7-53
 setting 7-10, 7-12, 7-53

Excel, creating datapool files with 6-46

excluding datapool columns 6-18

executables 7-12, 7-54

 deleting from schedules 7-33
 monitoring 8-11
 replacing 7-33

executing schedules 4-15, 7-66

execution order of users 7-62

Execution_list. *See* schedules, setting runtime information

exponential notation data type C-5

exporting

 datapools 6-37
 report output 2-18, 9-12
 schedules 7-57

F

feedback during virtual user script recording 3-26, 4-3, 4-7, 4-9

fields in datapools. *See* columns in datapools

FIELDTBLS system environment variable 7-42

FIELDTBLS32 system environment variable 7-43

file location

 session files 4-2
 virtual user scripts 4-2

Index

file types

- .csv (datapool files) 6-3, 6-36
- .csv (report output) 2-18, 9-12
- .s (scripts) 4-2
- .spc (datapool specification files) 6-3, 6-36
- .wch (session) 4-2

files

- copying to Master 7-37
- datapool file location 6-3
- location on Agent 7-38
- removing from Agent 7-37
- Schedule Log 9-4
- total open B-3
- User Error 8-18, 8-33, 9-4
- User Log 8-18, 8-33, 9-4
- User Output 9-4

filtering

- group views 8-30
- protocols 3-16, 3-17, 4-11
- report data 9-16, 9-33
- user views 8-29

firewalls, controlling port numbers B-3

first names data type C-6

FLDTBLDIR system environment variable 7-42

FLDTBLDIR32 system environment variable 7-43

float data types C-5

floating point numbers 6-29, C-5

floating toolbars

- VU Insert 4-5
- VU Record 4-5

FTP protocol 3-19

Full user view 8-12, 8-16

- sorting users in 8-27

functional tests. *See* distributed functional tests

G

gender data type (M, F) C-5

generating

- values in datapools, example 6-32
- values in datapools, in Robot 6-20, 6-22
- values in datapools, in TestManager 6-26

generating virtual user scripts 4-4

- from a session 4-17
- manual protocol filtering 4-11
- problems with 4-5
- recording options 3-17

global datapool directives 6-17

graphs

- changing formats 9-32
- copying 9-8
- data point information 9-32
- modifying labels 9-33
- resource monitoring 8-23

grids, displaying in graphs 9-31

Group views 8-26

groups

- adding to custom histograms 8-37
- deleting from custom histograms 8-38
- filtering 8-30

GUI histograms 8-5, 8-7

GUI scripts

- datapools and 6-24
- recording 2-22
- setting pass criteria for 7-61
- SQABasic language and 1-4
- See also* scripts

GUI scripts and datapools 6-24

- associating variable names and datapool columns 6-19, 6-48
- datapool access shared with virtual user scripts 6-24

GUI users 1-6, 7-8
 Agent computers and 1-5
 including in performance tests 1-8
 inserting executable in group 7-12
 next available selectors 1-10, 2-20, 7-15
 synchronizing 5-7

H

help desk xv
 hexadecimal data type C-5
 histograms
 custom 8-6, 8-36
 GUI 8-5, 8-7
 HTTP 8-5, 8-8
 IIOP 8-6, 8-9
 SQL 8-5, 8-8
 standard 8-5, 8-6
 zooming in on bars 8-9
 hotline support xv
 HP-UX Agents 7-41
 HTTP histograms 8-5, 8-8
 HTTP scripts
 cached responses 3-20
 dynamic data correlation 3-21
 enabling IP aliasing 7-64
 keep-alives 3-20
 partial responses 3-20
 redirects 3-20
 HTTP VU emulation commands
 reporting 9-24, 9-52, 9-53, 9-54
 response from cache 9-50

I

icon for Virtual User Recorder 3-26, 4-9
 IIOP
 assigning prefix to emulation commands 3-24
 emulation commands in Trace reports 9-24
 including original IORs in `iiop_bind` 3-24
 IIOP histograms 8-6, 8-9
`iiop_bind` emulation command 3-24
 IME 6-11, 6-13, 6-27, 6-30
 importing
 datapools from another project 6-37
 datapools from outside Rational Test 6-35
 sessions 4-16
 user-defined data types 6-41
 including datapool columns 6-18
 Input Method Editor 6-11, 6-13, 6-27, 6-30
 inserting columns in a datapool 6-20, 6-22
 integer data type C-6
 interface card, selecting for network recording 3-5
 IP aliasing 7-64
 iterations
 displaying schedule 8-4
 setting in schedule 7-10

J

Japanese characters 6-10, 6-13
 Java applets 3-19
 job classes. *See* scenarios
 Jolt protocol 3-18
 Julian date data types C-4, C-5

K

Kanji characters 6-13
 Katakana characters 6-13
 keep-alives, in HTTP scripts 3-20
 keys in unique datapool rows 6-42

Index

L

- last names data types C-6
- LD_LIBRARY_PATH system environment variable 7-41, 7-42
- legends, displaying in graphs 9-31
- LIBPATH system environment variable 7-41, 7-42
- library source files 4-20
- literal value data type C-8
- load tests, about 1-8
- LoadTest 1-1
 - basic concepts 1-3
 - hardware and software environment 1-12
 - logging into 1-14
 - total processes B-3
 - uses 1-2
- log levels, setting 2-5, 7-45, 8-19
- log scale, displaying in graphs 9-31
- logging into LoadTest 1-14
- login information
 - Authentication Datapool 3-29
 - automatic detection 3-29
 - user ID and password 3-29
- logs 7-66
 - copying to Master 7-37
 - displaying 8-18, 8-33, 9-4
 - folder name 7-67
 - location on Agent 7-38
 - naming 7-67
 - removing from Agent 7-37
 - running reports against 9-17, 9-18
- LogViewer 2-15, 7-61, 9-4

M

- manual protocol filtering 3-17, 4-11
- mapping
 - clients and servers for proxy recording 3-8
 - illustration of proxy mapping 3-6
 - proxy computer with a server 3-8
 - resource usage onto response time 9-27
- Master computers 1-5, 7-37, B-3
 - monitoring resources of 8-21, 9-3, 9-27
 - monitoring status of 8-21
 - running schedules on 1-6
 - TUXEDO 7-42
- maximum response time 2-18, 9-44
- mean response time 2-18, 9-44
- median response time 2-18, 9-44
- memory
 - minimum shared for large user runs B-2
 - response from cache 9-50
 - total shared for large user runs B-3
- Message user view 8-12, 8-16, 8-28
- Microsoft Excel, creating datapool files with 6-46
- middle initials data type C-7
- middle names data type C-6
- minimum response time 2-18, 9-44
- missing passwords 4-4, 4-5
- monitoring computer resources 2-18, 8-21, 9-3, 9-27
 - setting option to allow 7-67
- monitoring proxies 3-34
- monitoring schedules 8-1
 - changing default settings 8-34
 - Compact user view 8-12, 8-13, 8-27
 - Computer view 8-21
 - Full user view 8-12, 8-16, 8-27
 - Group views 8-26
 - Message user view 8-12, 8-16, 8-28
 - Results user view 8-12, 8-14, 8-27
 - Script view 8-18

- setting update rates 7-67
- Shared Variables view 8-17
- Source user view 8-12, 8-15, 8-28
- Sync Points view 8-19
- Transactor views 8-25

multi-byte characters 6-10, 6-11, 6-13, 6-27, 6-30

N

N_users. *See* schedules, running

names data types

- company names C-2
- first names C-6
- last names C-6
- middle initials C-7
- middle names C-6
- titles (Mr, Ms) C-6

naming

- report output 9-10
- reports 9-10
- scripts 3-25
- sessions 3-25

nesting blocks in virtual user scripts 5-6

network interface card, selecting for network recording 3-5

network recording 3-2, 3-4

- defining a client or server computer for 3-27
- filtering protocols 3-16
- identifying the client and server 3-4
- selecting a network interface card 3-5
- starting applications 3-25, 3-26

network services B-3

next available selectors 1-10, 2-20, 7-15

NLSPATH system environment variable 7-42

Normal_term_cnt. *See* schedules, setting runtime information

numbers data type C-6

NuTCRACKER settings, changing when running large numbers of users B-1

O

Oracle

- client name required 3-23
- login information 3-31
- proxy recording and 3-7
- setting system environment variables 7-41

ORACLE_HOME system environment variable 7-41

outliers 2-18, 9-22, 9-24

output files

- copying to Master 7-37
- displaying 9-4
- location on Agent 7-38
- removing from Agent 7-37

P

packed decimal data type C-7

partial responses, in HTTP scripts 3-20

passwords

- modifying in Robot 3-31
- modifying in TestManager 3-30
- supplying 3-29, 4-4, 4-5

PATH 7-41

PATH system environment variable 7-41

Performance reports 2-15, 2-17, 9-3, 9-44

- automatically run 9-4
- comparing output of 9-40
- filtering data 9-16
- graphing 9-22, 9-30
- setting response ranges 9-22, 9-24
- setting response time calculation 9-23
- setting response time percentiles 9-23
- setting response type 9-22
- setting stable loads 2-5, 9-23, 9-25

Index

- setting time period for 9-23
- sorting command IDs 9-22
- performance tests 1-7, 2-2
 - clients 1-11, 2-1
 - database servers 1-12
 - including GUI users in 1-8
 - recording scripts for 4-3
 - TUXEDO servers 1-12
 - vs. distributed functional tests 2-2
 - Web servers 1-12
- PerformanceStudio Authentication 3-30
- PERMUTE. *See* random without replacement
- selectors
- persistent datapool cursors 6-4, 6-16
 - starting row number 6-18
- phone numbers data types C-7
- planning
 - datapools 6-6
 - distributed functional tests 2-19
 - performance tests 2-2
 - virtual user recording 3-1
 - virtual user scripts 2-11
- playing back virtual user scripts
 - pacing 3-15
 - verifying SQL return codes 3-13
- populating datapools
 - example 6-32
 - in Robot 6-20, 6-22
 - in TestManager 6-26
- port numbers B-3
- prefixes
 - autonaming virtual user scripts and sessions 3-25
 - command ID 3-10, 3-23
 - TUXEDO command ID 3-23
- printing
 - report output 9-8
 - reports 9-7
 - schedules 7-57
- private datapool cursors 6-16
- processes
 - total LoadTest B-3
- properties
 - report 9-30, 9-34
 - report output 9-34
 - schedule 7-32
 - session 4-18
- properties of scripts 1-4, 7-30
 - accessing from session properties 4-19
 - defining 2-11, 4-21
 - status when re-recording scripts 4-24
 - status when re-recording sessions 4-23, 4-24
- protocols
 - converting 4-14
 - filtering 3-16, 3-17, 4-11
 - Java applets 3-19
 - Jolt 3-18
 - selecting for virtual user scripts 3-18
 - socket 3-19
 - TUXEDO definition 3-19
- proxy computers 3-6
 - associating with a server 3-8
 - computer shutdown and 3-32
 - creating 3-8
 - deleting 3-35
 - monitoring 3-34
 - reassociating with a client/server pair 3-35
 - redefining after proxy service is stopped 3-33
 - status 3-34

- proxy recording 3-2, 3-6, 3-7
 - client/server relationships 3-6
 - computer shutdown and 3-32
 - creating a proxy computer 3-8
 - defining a client or server computer for 3-27
 - filtering protocols 3-16
 - identifying client/server pairs 3-8
 - starting applications 3-25, 3-26
- proxy service
 - computer shutdown and 3-32
 - starting 3-32
 - stopping 3-32

R

- random alphabetic string data type C-7
- random alphanumeric string data type C-7
- random datapool access 6-4, 6-17
- random numbers 7-48
- random value seed 6-30
- random with replacement selectors 7-16
 - generating random numbers for 7-63
- random without replacement selectors 7-16
 - generating random numbers for 7-63
- ranges in dates C-2, C-3, C-4
- Rational technical support xv
- Read From File data type 6-48, C-8
 - unique values 6-50
- record levels
 - for Trace reports 9-50
 - setting 2-5, 7-46
- recording GUI scripts 2-22
- recording methods
 - API recording 3-4
 - network 3-4
 - proxy 3-6
 - setting 3-2

- recording options 3-1
 - changing 4-18
 - filtering protocols 3-16
 - proxy 3-7
 - script generation 3-17
 - setting 3-1, 3-7
 - setting recording method 3-2
- recording virtual user scripts 2-12, 4-3
 - cancelling 4-9
 - feedback 3-26, 4-3, 4-7, 4-9
 - recording a single script in a session 4-3
 - recording methods 3-2
 - values recorded 6-19
- records in datapools. *See* rows in datapools
- redirects, in HTTP scripts 3-20
- regenerating scripts from a session 4-17
- release times 5-7, 5-10
 - ranges 5-10
 - staggering 2-14, 7-27
- renaming
 - datapools 6-35
 - report output 9-10
 - reports 9-10
 - schedules 7-51
 - user-defined data types 6-41
- report output
 - Analog 9-39
 - Compare 9-40
 - comparing 9-13
 - copying 9-8
 - default names of 9-2
 - deleting 9-11
 - exporting 2-18, 9-12
 - Performance 9-44
 - printing 9-8
 - properties of 9-34
 - renaming 9-10

Index

- Response 9-46
- Status 9-48
- Trace 9-49
- Usage 9-52
- reports 9-2
 - Analog 3-12, 3-14, 9-3, 9-38
 - changing default settings 9-35
 - changing graph formats 9-32
 - changing reports that run automatically 9-35
 - Compare 9-3, 9-13, 9-40
 - copying 9-8
 - customizing 9-15, 9-22
 - deleting 9-11
 - displaying grids 9-31
 - displaying legends 9-31
 - displaying log scales 9-31
 - filtering data 9-16, 9-33
 - filtering with block names 5-5
 - inverting axes 9-31
 - Performance 2-15, 2-17, 9-3, 9-44
 - printing 9-7
 - properties of 9-30, 9-34
 - renaming 9-10
 - Response 2-16, 2-17, 9-46
 - restoring default 9-12, 9-38
 - running from menu bar 9-6
 - running from report bar 9-6
 - saving 8-39
 - setting response time calculations 9-23
 - setting response type 9-22
 - setting stable loads 2-5, 9-23, 9-25
 - setting time period for 9-23
 - sorting command IDs 9-22
 - Status 9-3, 9-48
 - Trace 9-4, 9-49
 - types of 9-3
 - Usage 9-4, 9-52
- requests
 - cancelling recorded 4-9
 - monitoring during recording 4-7
 - recording 1-6, 4-3
- re-recording
 - sessions 4-22
 - virtual user scripts 4-24
- resource monitoring 2-18, 8-21, 9-3, 9-27
 - setting option to allow 7-67
- Response reports 2-16, 2-17, 9-46
 - graphing 9-22, 9-30
 - including passed or failed responses 9-23
 - resource monitoring 9-3, 9-27
 - setting response ranges 9-22, 9-24
 - setting response time calculations 9-23
 - setting response type 9-22
 - setting stable loads 9-23, 9-25
 - setting time period for 9-23
 - sorting command IDs 9-22
- response times
 - reporting on 2-18, 9-23
 - standard deviation 2-18, 9-44
- restoring default reports 9-12, 9-38
- restoring Robot during virtual user recording 4-2
- result files
 - changing information in 7-46
 - copying to Master 7-37
 - location on Agent 7-38
 - removing from Agent 7-37
- Results user view 8-12, 8-14
 - sorting users in 8-27
- rewinding the datapool cursor 6-16
- Robot, restoring during virtual user recording 4-2
- row access order 6-4, 6-17
- rows in datapools
 - access order 6-4, 6-17
 - maximum number 6-4

- records and 6-45
 - starting row number 6-18
 - unique 6-43
 - Run_time. *See* schedules, setting runtime information
 - running
 - applications 3-25
 - reports 9-6
 - schedules 1-6, 2-15, 2-23, 4-15, 7-66
 - schedules. *See also* monitoring schedules
- S**
- saving schedules 7-58
 - scenarios 7-10, 7-54
 - deleting from schedules 7-33
 - replacing 7-33
 - Schedule Log files, displaying 9-4
 - schedules 7-2
 - changing logs in 7-67
 - checking 7-59
 - creating 7-3
 - deleting 7-51
 - deleting items from 7-33
 - displaying views of 8-4
 - editing 7-31, 7-32, 7-34
 - execution order of users 7-62
 - exporting 7-57
 - inserting an executable 7-12
 - inserting scripts 7-8
 - inserting selectors 7-13, 7-18
 - inserting synchronization point 2-14
 - inserting transactors 7-21, 7-22
 - minimum requirements for running 7-10, 7-59
 - monitoring 8-1
 - opening 7-28
 - percent done 8-3
 - printing 7-57
 - renaming 7-51
 - replacing items in 7-33
 - reporting on portion of 9-23
 - results 7-66
 - running 2-15, 2-23, 4-15, 7-66
 - saving 7-58
 - setting delays in 7-10, 7-20
 - setting maximum time for run 7-63
 - setting number of users 7-66
 - setting pass or fail criteria 7-61
 - setting runtime information 7-60
 - synchronization points and 5-9
 - synchronizing items in 5-10, 7-19, 7-25, 7-52
 - terminating 7-64, 7-67, 8-39, 9-57
 - time in run 8-3
 - scientific notation data type C-5
 - scope of a synchronization point 5-11
 - script generation options 3-17
 - adding SQL return codes to scripts 3-13
 - changing 4-18
 - command ID prefix 3-10
 - datapool commands 3-10
 - display number of rows retrieved 3-12
 - display retrieved rows in scripts 3-11
 - playback pacing 3-15
 - think time vs. CPU delays 3-16
 - script properties 1-4, 7-30
 - accessing from session properties 4-19
 - defining in Robot 4-21
 - status when re-recording scripts 4-24
 - status when re-recording sessions 4-23, 4-24
 - Script view 8-18
 - emulation commands temporarily unavailable in 8-19

Index

- scripts 1-4
 - as runtime users 1-4
 - changing number 7-47
 - deleting from schedules 7-33
 - editing 7-29
 - grouping into scenarios 7-10
 - initializing timestamps for 7-63
 - inserting delays 7-10, 7-20
 - inserting into schedule 7-8
 - inserting synchronization point 2-14
 - iterations 7-10
 - limiting number 7-48
 - replacing 7-33
 - reporting active times of 9-52
 - reporting inactive times of 9-52
 - runtime errors in 2-15
 - setting dependencies 7-54
 - setting events 7-52
 - syntax errors in 2-15
 - types of 1-4
 - variable names and datapool column names 6-27
 - where stored on Agent 7-38
- Seed flags 7-49
- seeds 7-49
 - base 7-63
 - for random selectors 7-63
 - for users 7-48
 - random datapool values 6-30
- selectors 7-13
 - deleting from schedules 7-33
 - dynamic load balancing 7-17
 - inserting into schedule 7-18
 - next available 1-10, 2-20, 7-15
 - random 7-16
 - sequential 7-15
- semaphores, changing maximum number of B-1, B-3
- sequential datapool access 6-4, 6-17
 - unique row retrieval and 6-44
- sequential selectors 7-15
- servers
 - associating with a client 3-8
 - associating with a proxy 3-8
 - changing 2-10
 - defining for network or proxy recording 3-27
 - performance tests 1-12
 - removing 3-29
 - response times 1-11, 2-1
 - selecting for network recording 3-4
 - testing 2-6
 - think time distribution and 2-14
- session files 4-1
 - regenerating scripts from 4-17
 - where stored 4-2
- session ID 3-21
 - where stored 3-21
- sessions 1-5
 - autonaming 3-25
 - contents of 4-15
 - creating schedule from 7-4
 - deleting 4-25
 - finding the session name for a script 4-22
 - importing 4-16
 - properties of 4-18
 - recording 4-1
 - regenerating scripts from 4-17
 - removing a script from 4-22
 - re-recording 4-22, 4-23
 - scripts within 4-17, 4-21
 - splitting into multiple scripts 2-12, 4-15, 4-16
 - uses for 4-15
 - where stored 4-2
- shared datapool cursors 6-4, 6-16
- shared memory B-2, B-3

- shared variables
 - changing value of 8-31
 - displaying users waiting on 8-32
 - initializing 7-55
 - viewing values of 8-17
- SHLIB_PATH system environment variable 7-41, 7-42
- shuffle datapool access 6-4, 6-17
 - unique row retrieval and 6-44
- socket protocols 3-19
 - changing 4-14
- socket VU emulation commands, reporting 9-52, 9-53, 9-55
- Solaris Agents 7-41
- sorting users 8-27
- Source user view 8-12, 8-15
 - sorting users in 8-28
- space data type C-8
- .spc datapool specification files 6-3, 6-36
- SQABasic language 1-4
 - in GUI scripts 1-4
 - monitoring 8-11
- SQL histograms 8-5, 8-8
- SQL return codes, comparing during virtual user recording and playback 3-13
- SQL Server login information 3-31
- SQL VU emulation commands, reporting 9-24, 9-52, 9-53
- stable loads
 - planning 2-5, 2-16, 2-17
 - setting in reports 2-5, 9-23, 9-25
- staggering release times 2-14, 7-27
- standard data types
 - list of C-1
 - minimum and maximum values C-9
 - role of 6-9
 - when to use 6-10
- standard deviation of response times 2-18, 9-44
- standard histograms 8-5, 8-6
- start scripts 7-47
 - setting maximum initialization time for 7-63
- Start_group_size. *See* schedules, setting runtime information
- start_time VU emulation commands 2-14
 - reporting on 9-58
- Start_time. *See* schedules, setting runtime information
- starting applications 3-25
- starting proxy service 3-32
- state abbreviations data type C-8
- states
 - assigning to custom histograms 8-36
 - removing from custom histograms 8-37
- Status reports 9-3, 9-48
 - automatically run 9-4
 - data summary style 9-23
 - graphing 9-22, 9-30
 - setting response ranges 9-22, 9-24
 - setting response type 9-22
 - setting stable loads 9-23, 9-25
 - setting time period for 9-23
 - sorting command IDs 9-22
- stop_time VU emulation commands 2-14
 - reporting on 9-56
- stopping
 - proxy service 3-32
 - recording 4-4
- street names data type C-2
- stress tests 1-8, 2-9, 7-26
- string constant data type C-8
- structure of datapools 6-45
- support, technical xv
- survey utility. *See* schedules, monitoring
- suspending virtual users 8-34, 8-38
- Sybase
 - login information 3-31
 - setting system environment variables 7-41

Index

synchronization points 2-14, 5-7, 5-10, 7-25, 7-27, 8-19

- deleting from schedules 7-33
- displaying state of 8-19
- example of 5-8
- inserting into schedules 2-14, 5-9, 5-10, 7-25
- inserting into script 2-14, 5-9, 5-10
- multiple 5-9
- number of users waiting 8-19, 8-20
- release time ranges 5-10
- releasing 8-20
- releasing users from 5-7, 5-10, 7-27, 8-19
- replacing 7-33
- scope of 5-11
- timeout 5-10, 7-28, 8-20

synchronizing items in schedules

- delays 7-19
- events and dependencies 7-52
- synchronization points 5-10, 7-25

system environment variables 7-41, 7-44

T

`Task_dir`. *See* schedules, running

`Task_term_cnt`. *See* schedules, terminating

`Task_ts_init`. *See* schedules, setting runtime information

tasks. *See* scripts

technical support xv

Telnet protocol 3-19

terminating schedules 7-64, 7-67, 8-39, 9-57

- planning 2-5

terminating users 8-34

- abnormal 8-3, 8-13, 8-14, 8-27
- normal 8-3, 8-27

`TEST7_LTMASTER_NTUSERLIMIT` system environment variable B-2

`TEST7_LTMASTER_SHM_MINSZ` system environment variable B-2

testcase VU emulation commands 9-56, 9-58

tests 1-7

- configuration 1-10
- contention 1-3, 1-9
- distributed functional 1-10, 2-19
- load 1-8
- performance 1-7, 2-2
- stress 1-8, 2-9, 7-26

text files, assigning values to a datapool column 6-48, C-8

think time

- CPU delays and 3-16
- distribution 2-14
- maximum 3-16
- setting 3-15

threshold between think time and CPU delays 3-16

throughput 9-53, 9-57

time data types C-8

timeout values, for synchronization points 5-10

timers 5-1

- adding during editing 5-3
- adding during recording 5-2

times

- client response 1-11, 2-1
- reporting active 9-52
- reporting inactive 9-52
- server response 1-11, 2-1
- setting maximum initialization 7-63
- setting maximum schedule run 7-63
- standard deviation of response 2-18, 9-44
- suppressing delays 7-63
- think 2-14, 3-15, 3-16
- think time distribution 2-14

timestamps

- in Trace reports 9-50
- initializing for scripts 7-63
- omitting from Trace report 9-24

`TNS_ADMIN` system environment variable 7-41

- toolbars
 - VU Insert 4-5
 - VU Record 4-5
 - working with A-1
 - Total_term_cnt. *See* schedules, setting runtime information
 - TPINIT request message 3-19
 - Trace reports 9-4, 9-49
 - including command types 9-24
 - including passed or failed responses 9-23
 - including VU environment variables 9-24
 - omitting timestamps 9-24
 - Transaction reports. *See* Response reports
 - transactions 2-11
 - automatically timing in blocks 2-13, 5-4
 - performing during recording 4-4
 - transactors 7-21, 7-54, 8-25
 - displaying information about 8-25
 - inserting into schedule 7-22
 - truncating emulation command ID prefixes 3-11
 - TUXEDO
 - command ID prefixes 3-23
 - defining a connection 3-19
 - emulation commands in Trace reports 9-24
 - emulation commands in Usage report 9-52, 9-53, 9-56
 - performance testing 1-12
 - setting system environment variables 7-42
- ## U
- U.S. cities data type C-2
 - U.S. state abbreviations data type C-8
 - unique datapool rows
 - guidelines for 6-42
 - Read From File data type and 6-50
 - setting unique values 6-28
 - user-defined data types and 6-12
 - update rates, when monitoring schedules 7-67
 - usage directives for datapools 6-18
 - Usage reports 9-4, 9-52
 - data summary style 9-23
 - User Error files
 - displaying 8-18, 8-33, 9-4
 - user groups 2-11, 7-5
 - changing information about 7-35
 - changing number of users 7-36
 - deleting from schedules 7-33
 - displaying information about 8-26
 - editing properties of 7-34
 - fixed 7-6
 - inserting into schedules 7-5
 - planning 2-4
 - replacing 7-33
 - scalable 7-6
 - temporarily disabling 7-48, 7-62
 - user IDs
 - modifying list of, in Robot 3-31
 - modifying list of, in TestManager 3-30
 - supplying 3-29
 - User Log files
 - displaying 8-18, 8-33, 9-4
 - User Output files
 - displaying 9-4
 - user settings, changing 7-38
 - User_term_mode. *See* schedules, terminating
 - user-defined data types
 - automatically generating values for 6-39
 - copying 6-42
 - creating 6-11
 - deleting 6-42
 - editing definitions of 6-39
 - editing values in 6-38
 - importing 6-41
 - renaming 6-41

Index

- role of 6-9
- unique values 6-12
- when to use 6-10

userlists. *See* user groups

users 8-3

- abnormal termination 8-3, 8-14, 8-27
- active 8-3
- changing number of 7-36
- combining GUI and virtual 7-8
- displaying information about 8-5
- distributing among computers 7-8, 7-36
- execution order 7-62
- filtering 8-29
- GUI 1-6
- limiting number of scripts run by 7-48
- normal termination of 8-3, 8-27
- percentage executing schedule 8-5
- reporting active times of 9-52
- reporting inactive times of 9-52
- seeds for 7-48
- setting number of 7-6
- setting termination counts for 7-65
- sorting 8-27
- starting at different times 7-47
- terminating 8-34
- total number in run 8-3
- virtual 1-6

users. *See also* GUI users, virtual users

V

values supplied during recording 6-19

variable names and datapool column names 6-19

variable names, and datapool column names 6-27, 6-48

verifying SQL return codes 3-13

VIEWDIR system environment variable 7-43

VIEWDIR32 system environment variable 7-43

VIEWFILES system environment variable 7-43

VIEWFILES32 system environment variable 7-43

viewing

- datapool values, in Robot 6-23
- datapool values, in TestManager 6-34
- user-defined data type values 6-38

views

- Compact user 8-12, 8-13, 8-27
- Computer 8-21
- customizing 8-27
- Full user 8-12, 8-16, 8-27
- Message user 8-12, 8-16, 8-28
- restoring default 8-31
- Results user 8-12, 8-14, 8-27
- Script 8-18
- Shared Variables 8-17
- Source user 8-12, 8-15, 8-28
- Sync Points 8-19

Virtual User Recorder 4-7

- after recording 4-9
- during API recording 4-7
- icon 3-26, 4-9

virtual user scripts

- adding datapool commands to 3-10
- adding number of rows retrieved to 3-12
- adding retrieved rows to 3-11
- autonaming 3-25
- blocks 2-13, 5-4
- cancelling 4-9
- changing recording options 4-18
- command ID prefixes in 3-10
- comments in 2-13, 5-11
- copying 4-25
- copying code from one script to another 4-20
- datapool access shared with GUI scripts 6-24
- debugging 8-33
- deleting 4-23, 4-25

- distinguishing think time and CPU delays 3-16
 - generating after recording 4-4
 - generating from a session
 - improving readability 3-10, 5-5
 - list of, in a session 4-17, 4-21
 - manual protocol filtering 4-11
 - maximum think time 3-16
 - methods of recording 3-2
 - modifying connect line of 2-10
 - overwriting 4-24
 - planning 2-11
 - playback rate 3-15
 - properties 4-20
 - recording 2-12, 4-3
 - recording options for 3-1
 - regenerating from a session 4-17
 - removing from a session 4-22
 - re-recording 4-24
 - script generation options 3-17
 - session associated with 4-22
 - splitting 2-12, 4-15, 4-16
 - SQL return code verification 3-13
 - stopping recording 4-4
 - think time in 3-15
 - updating on Agent 7-37
 - user groups 2-11
 - variable names and datapool column names 6-19
 - VU language and 1-4
 - where stored 4-2
 - See also* scripts
 - virtual users 1-6, 7-8
 - changing log level 7-45
 - changing record levels of 7-46
 - determining number supported 2-6
 - incrementally loading 2-7
 - pacing 3-15
 - resuming suspended 8-34
 - running large numbers B-1
 - setting limit for large user runs B-2
 - setting number of 7-66
 - suspended 8-3, 8-27
 - suspending 8-34, 8-38
 - synchronizing 5-7
 - think time 3-15
 - think time vs. CPU delays 3-16
 - VU display library routine 8-12, 8-16, 8-28
 - VU environment variables
 - in Trace reports 9-24
 - Log_level 7-39
 - Record_level 7-46
 - VU Insert toolbar 4-5
 - VU language 1-4
 - in virtual user scripts 1-4
 - VU Record toolbar 4-5
- ## W
- watch files. *See* session files
 - Web
 - performance testing 1-12
 - structural testing of 1-7
 - Windows NT Agents 7-41
 - workloads 2-2
 - adding to system-under-test 1-5, 1-8
 - designing 2-4
 - planning stable 2-5, 2-16, 2-17
 - reporting on stable 2-5, 9-23, 9-25
 - WorkStation Listener information 3-19
 - wrapping the datapool cursor 6-16
 - WSDEVICE system environment variable 7-43
 - WSLHOST system environment variable 7-42
 - WSLPORT system environment variable 7-42
 - WSN ADDR system environment variable 7-42

Index

Z

Zap_mode. *See* schedules, setting runtime information

zip code data types C-9

zoned decimal data type C-9