Getting Started with Rational Suite PerformanceStudio

Version 2000.02.10

Rational®
the e-development company™

Getting Started with Rational Suite PerformanceStudio

Copyright © 1999-2000 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. GOVERMENT 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 800-023368-000

Contents

3 Tutorial

▸ ▸ ▸ Preface

Rational Suite PerformanceStudio is a sophisticated tool for automating performance and distributed functional tests on client/server systems.

This manual introduces Rational Suite PerformanceStudio and provides a tutorial for first-time users. The manual is intended to help application developers and testers use PerformanceStudio.

Other Resources

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

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

- All manuals for this product are available online in PDF format. These manuals are on the *Rational Solutions for Windows* Online Documentation CD.

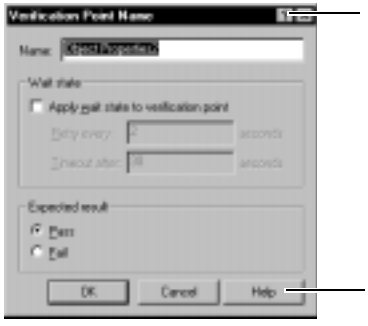
- For information about training opportunities, see the Rational University Web site: http://www.rational.com/university.

Using Help

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

Dialog Box Help

Most dialog box Help includes overviews and detailed item information.



Click this and then click an item to see information about the item.

Click this to see an overview of the dialog box.

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: – Your name, telephone number, and company name – Computer make and model – Operating system and version number – Product release number and serial number – Your Case ID number (if you are calling about a previously reported problem)
Europe	Telephone: +31 (0) 20 4546 200 E-mail: support@europe.rational.com	
Asia Pacific	Telephone: +61-2-9419-0111 E-mail: support@apac.rational.com	
World Wide Web	http://www.rational.com	Click the Technical Support link.

▸▸▸ CHAPTER 1

About Rational Suite PerformanceStudio

If you've installed Rational Suite PerformanceStudio, you're on your way to ensuring that your business-critical client/server and Web applications perform exactly as you want. With PerformanceStudio, you can test the capacity, performance, and stability of your systems under real-world, multi-user workloads.

Here are just some of the things that PerformanceStudio lets you do:

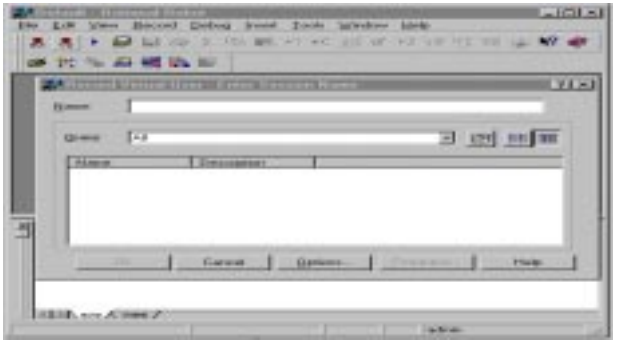
- Model complex workloads without programming.
- Monitor and analyze the response times that users actually experience under different workloads.
- Automatically maintain accurate user emulation during playback.
- Verify correct playback of changing Web page content.
- Automatically replace literal data in scripts with variables, and then associate the variables with test data.

The Path to Successful Performance Testing

1. Install Rational Suite PerformanceStudio. For information about installing PerformanceStudio, see *Installing Rational Suite*.

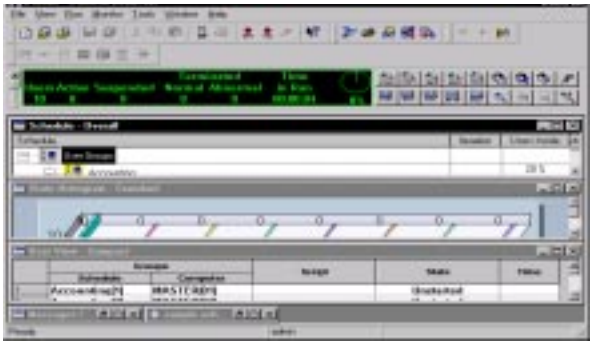
2. Create a Rational repository in the Rational Administrator to store your test assets. For information about creating a Rational repository, see the *Using the Rational Administrator* manual.

3. Record scripts in Rational Robot that emulate client/server conversations. For information about recording GUI scripts, see the *Using Rational Robot* manual. For information about recording virtual user scripts, see the *Using Rational LoadTest* manual.

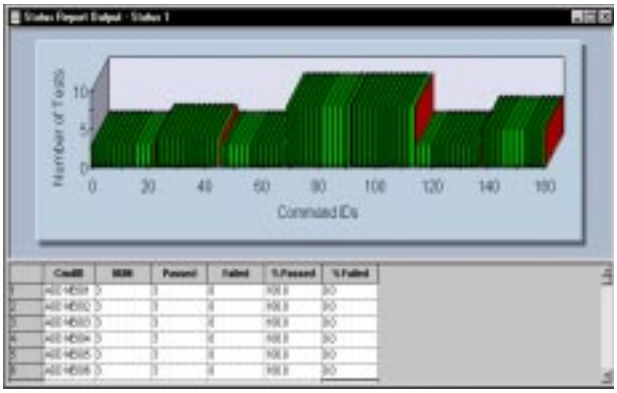


4. Create schedules in Rational LoadTest that emulate clients sending requests to a server. For information about creating schedules, see the *Using Rational LoadTest* manual.

5. Run and monitor the schedules as they add load to your database and Web servers. For information about running and monitoring schedules, see the *Using Rational LoadTest* manual.



6. Analyze the results of your tests. For information about analyzing results, see the *Using Rational LoadTest* manual.

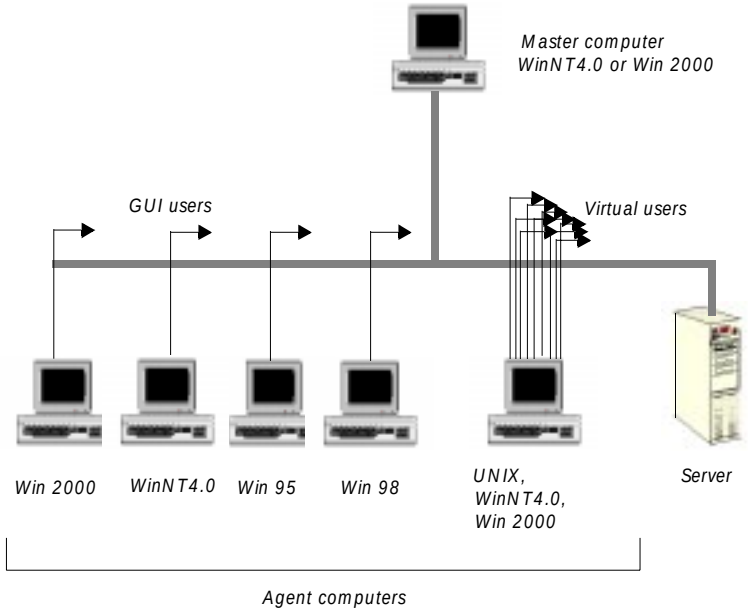


Setting Up Your PerformanceStudio Test Lab

Before you deploy your business applications, you want to know if they perform as you expect them to. With Rational Suite PerformanceStudio, you need only a few computers to emulate the real-world environment of complex client/server interactions and extensive Web commerce.

You can transform your test lab into an activity center that emulates hundreds or thousands of users sending and receiving information between your client applications and their database, application, or Web servers.

In your PerformanceStudio test lab, the point of control is the computer where the PerformanceStudio software is installed. From this Master computer, you can schedule, coordinate, monitor, and analyze the playback of GUI users and virtual users.



GUI users test the client application response time, stability, and functionality of live clients. **Virtual users** test application, database, or Web servers as well as the network between the virtual users and these servers. Virtual users add load by emulating hundreds or thousands of client application sessions, or users connected to a server, without requiring an actual computer for each user.

In your test lab, you can use the Master computer by itself to play back scripts, or you can distribute scripts to various Agent computers that extend the computer resources of the Master. Using the Master and its Agents to play back scripts creates large workloads that test server performance. Agents can play back numerous combinations of virtual user and GUI user scripts, providing you with the flexibility and scalability you need to meet your testing requirements.

Types of Tests

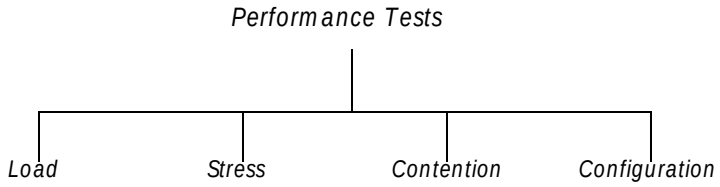
PerformanceStudio lets you answer the following questions before you deploy your applications:

- How will the system perform in a production environment?
- How long does it take for the database transaction to complete?
- How quickly was the server able to respond under heavy workload conditions?
- What are the usage patterns that impact performance?
- Where do bottlenecks exist?
- Where is the application's break point and how can it be avoided?

To help you discover the answers to these questions, Rational Suite PerformanceStudio provides you with a variety of tests for use throughout the testing lifecycle.

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. Performance tests typically 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. The term "performance testing" includes load, stress, contention, and configuration tests. These tests let you accelerate the performance test cycle and achieve meaningful, accurate results.



Load Tests

Load testing lets you test client or server response times under heavy workloads. These tests help you compute the maximum number of transactions a server can handle over a given time period. Also, when a client/server system uses load balancing or a distributed architecture, load testing helps ensure that the load balancing or distribution methods work as designed.

Stress Tests

Stress testing runs your client application under extreme conditions to see whether it or the server “breaks.” You can perform a stress test by continuously running a client application for many hours, performing a large number of transactions, or by having hundreds of emulated users perform the same operation at virtually the same moment.

In addition, as you stress the server, you may discover insufficient memory, unavailable services or hardware, or diminished shared resources.

Stress testing helps you make sure that your server or client application can handle production conditions.

Contention Tests

Contention testing involves running tests to simulate a live user environment—for example, users accessing the same database record which can 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. With Rational Suite PerformanceStudio, you can synchronize user tasks. You can also conduct multi-user scenarios in which emulated users wait for conditions to be satisfied before they continue running. For instance, while one virtual user adds a record to a database, a second virtual user waits for the first user to finish its transaction before attempting to read the record.

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 will run on some machines and not on others. With configuration testing, you can ensure that your product will continue to run on multiple platforms. You also determine the minimal and optimal configuration of hardware and software that you need.

Recording Scripts

In Rational Suite PerformanceStudio's point-and-click environment, you can record scripts that emulate the way users will actually use your system.

Types of Scripts

PerformanceStudio provides two types of scripts—virtual user and GUI scripts.

Virtual user scripts test the performance of your client applications as they interact with your application, database, and Web servers. A virtual user script represents requests to a server, such as SQL and HTTP, and checks the server responses to those requests.

GUI scripts test the response time and stability of live clients as well as testing the properties of the client application.

Methods of Recording

Rational Suite PerformanceStudio ensures that the scripts you record accurately reflect the traffic between clients and servers, regardless of the transmission rate.

PerformanceStudio lets you record user activity in three ways:

- **API** – Records API calls between the client software and the client libraries that communicate with the server. API recording is the recommended approach for all Windows-based clients. PerformanceStudio and the client application are both installed on the client computer in API mode.
- **Network** – Records TCP/IP traffic transmitted across the network. Network recording is recommended when the client computer does not support API recording.
- **Proxy** – Records the same kind of traffic as network recording, but the client routes the traffic through a proxy computer. Proxy recording, a specialized form of network recording, is used on high-speed or switched networks.

Generating Scripts

With PerformanceStudio, you can create scripts simply, *without any coding*. PerformanceStudio's DataSmart™ recording lets you instantly generate test-ready scripts and test data. With this unique type of recording, PerformanceStudio automatically replaces literal data in a script with variables, and then associates the variables with data from your databases or from PerformanceStudio data-ready datapools. By using realistic data from datapools, PerformanceStudio ensures that your virtual user scripts accurately emulate the activities of different users sending requests to a server.

With PerformanceStudio, you can create, extend, or change a script manually. PerformanceStudio offers two scripting languages **VU** (similar in syntax to C) and **SQABasic** (similar in syntax to Microsoft Basic). With PerformanceStudio, you can also call external C libraries or programs.

As you record, PerformanceStudio lets you enhance your scripts. You can add **timers** to bracket a series of activities, **comments** to annotate your scripts, and **blocks** to identify transactions. PerformanceStudio's TransactionSmart™ Analysis automatically gathers transaction times for these blocks as a whole and for each individual command that comprises these transactions. With this information, you can immediately look inside a transaction to diagnose any problems with individual server requests.

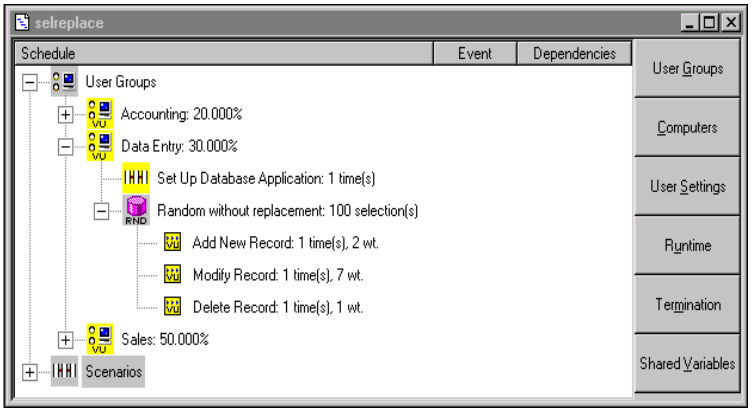
All of these features make your recorded scripts easy to record, read, and maintain.

Creating Schedules

A Rational Suite PerformanceStudio **schedule** provides a controlled, flexible way to create realistic user workloads that closely model actual user workloads.

With PerformanceStudio's LoadSmart™ scheduling, you can model complex workloads without programming. For instance, you can model hundreds of users applying stress to an application's database servers or thousands of hits on a Web site.

The building blocks of every PerformanceStudio schedule are emulated users organized into **user groups,** such as Data Entry, Accounting, and Sales, and the user activities they perform, encapsulated into scripts.

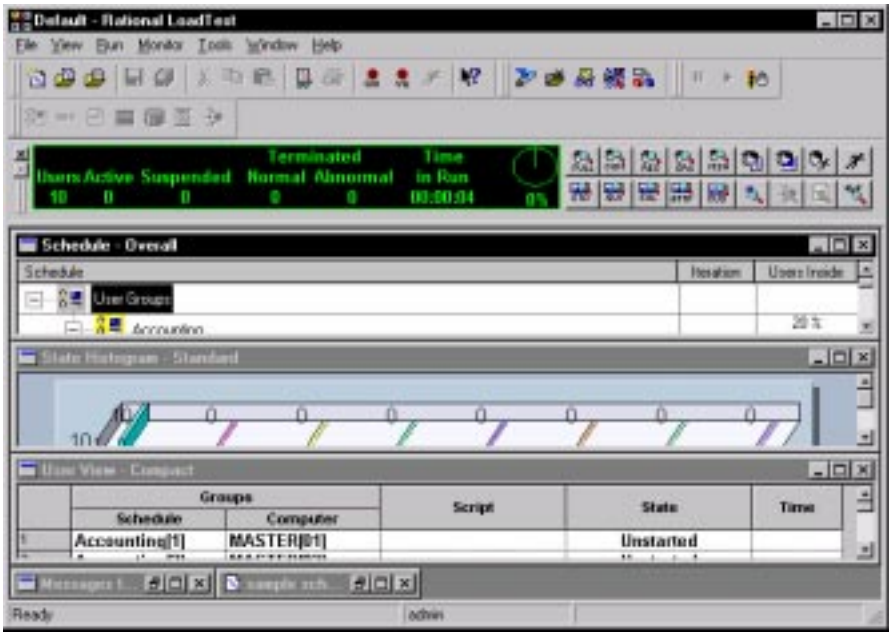


In the schedule, you can specify a weight for each user group to create realistic usage models. For example, you can specify that 20 percent of the users belong to the Accounting group, 30 percent to the Data Entry group, and 50 percent to Sales. Then you can create scripts that represent the activities of these groups and add these scripts to a schedule to emulate realistic workloads.

Running and Monitoring a Schedule

When a schedule is running, you can monitor its progress. You don't need to wait until the end of a schedule run to confirm that the schedule is progressing successfully or to discover that something went wrong. Instead, you can view information that is dynamically updated as each schedule executes.

Monitoring a schedule as it plays back lets you view all aspects of that schedule—its users, user groups, computers, and scripts. There are more than 15 different ways to view and monitor a schedule. And, you can use filters to customize each view to get just the data you want.

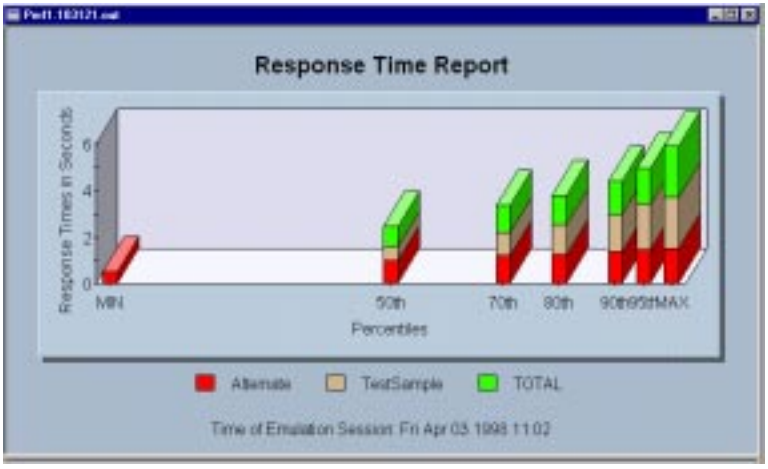


If you discover potential problems, you can suspend, resume, or terminate the schedule run, and you can view information about any of your emulated users. In addition, you can step through every transaction and every response to find out whether the schedule is progressing successfully.

Analyzing Results

A variety of PerformanceStudio reports deliver accurate metrics at the end of a schedule run, providing you with insight into your server's performance and stability under large user workloads. If a report does not meet your testing needs, you can easily customize its content and appearance.

You can display a report as a graph — line, bar, pie, stack, or area. You can also view the data in the form of a histogram, scatter, or Gantt chart. You can modify the labels of the *x* and *y* axes, zoom in on a particular area, display information on specific data points, and much more. You can also export a graph to a spreadsheet or third-party application for further analysis.



In addition to displaying reports graphically, you can also display them in grid format. This lets you see the statistics for the run represented by a particular graph.

	CmdID	NBR	MEAN	STDDEV	MIN	50th	70th	80th	90th	95th	MAX
1	wcd009	12	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.03
2	wcd010	12	1.41	0.80	0.40	1.05	2.12	2.29	2.40	2.52	2.65
3	wcd011	12	1.32	0.76	0.06	1.55	1.90	1.98	2.22	2.25	2.27
4	wcd012	11	1.46	0.83	0.03	1.35	1.99	2.50	2.72	2.73	2.74
5	wcd013	11	1.28	0.95	0.02	1.22	1.54	1.98	2.86	2.90	2.93
6	wcd014	9	1.43	0.79	0.17	1.35	1.90	2.04	2.22	2.54	2.67
7	wcd015	9	1.93	0.82	0.73	1.69	2.65	2.73	2.81	2.91	3.01
8	wcd016	7	2.04	0.71	0.89	2.03	2.55	2.67	2.79	2.84	2.89
9	wcd017	7	0.85	0.57	0.06	0.78	0.98	1.27	1.81	1.79	1.86
10	wcd018	7	0.96	0.69	0.03	0.77	1.10	1.43	1.78	1.96	2.14
11	wcd019	7	1.13	0.65	0.12	1.23	1.60	1.74	1.81	1.83	1.85
12	TOTAL	104	1.23	0.82	0.00	1.10	1.86	2.10	2.52	2.76	3.01

Where to Look Next

You've just been introduced to some of the ways that Rational Suite PerformanceStudio predicts and measures client/server and Web system testing. But there's still a lot more to discover.

Here's where you can find the information you need:

To find out	Read this
How to create a Rational repository and set up user access rights.	*Using the Rational Administrator*
How to plan tests, record and play back GUI scripts, and analyze functional testing results.	*Using Rational Robot*
How to create performance and distributed functional tests. Also, how to record virtual user scripts, how to schedule, run, and monitor tests, and then analyze the results.	*Using Rational LoadTest*
How to use the VU scripting language to enhance your virtual user scripts.	*VU Language Reference*
How to use the SQABasic scripting language to enhance your GUI scripts.	*SQABasic Language Reference*
How to use all of the functionality in Rational Suite PerformanceStudio.	Online Help
How to use Rational LoadTest to test the performance of your SAP applications.	*Rational LoadTest: Try it! for Virtual User Testing of SAP Applications*
How to use Rational Robot to test the functionality of your SAP applications.	*Rational Robot: Try it! for GUI Testing of SAP Applications*
How to use Rational SiteCheck to test the structural integrity of your intranet or World Wide Web site.	*Rational Robot: Try it! with Rational SiteCheck*
How to use Rational Robot to test the functionality of your HTML pages.	*Rational Robot: Try it! with HTML*
How to use Rational Robot to test the functionality of Java applets and applications.	*Rational Robot: Try it! with Java*
How to use Rational Robot to test the functionality of objects in your Oracle applications.	*Rational Robot: Try it! with Oracle Forms*
How to use Rational Robot to test the controls in your PowerBuilder applications.	*Rational Robot: Try it! with PowerBuilder*
How to use Rational Robot to test the controls in your Visual Basic applications.	*Rational Robot: Try it! with Visual Basic*

▸▸▸ CHAPTER 2

Before You Begin

This chapter tells you what you have to do before you start the Rational Suite PerformanceStudio tutorial. It includes the following topics:

- Tutorial requirements
- Copying the tutorial files
- Locating Internet Explorer
- Specifying Internet Explorer settings
- Finding out the name of the computer running the Web server
- Setting up a repository for the tutorial

Tutorial Requirements

This tutorial requires the following software:

- Rational Suite PerformanceStudio installed on the Master computer.

  For instructions, see *Installing Rational Suite*.
- Windows NT 4.0 server.
- Microsoft Internet Information Server.

  IIS is installed from the Windows NT 4.0 Option Pack.
- Browser software installed on the Master computer.

  Rational Suite PerformanceStudio requires Microsoft Internet Explorer 4.0 SP1 or greater.

Before you begin working with the tutorial, you need to:

- Find out the path of Internet Explorer. For information, see *Locating Internet Explorer* on page 2-2.

- Specify Internet Explorer settings. For information, see *Specifying Internet Explorer Settings* on page 2-3.
- Set up a Rational repository for the tutorial. For information, see *Setting Up a Repository for the Tutorial* on page 2-3.

Copying the Tutorial Files

Before you run the tutorial, you need to copy the Rational Suite PerformanceStudio tutorial files.

To copy the tutorial files:

1. Choose the Web server that you will use to serve the sample application. To run the tutorial, you must be able to access the directory structure of this server.

   NOTE: Make sure you know the network name of the Web server, because you will need this information for the tutorial.

2. Identify the root directory of the Web server that you will be using. For Microsoft Internet Information Server, the default location of the Web server root directory is:

   `C:\InetPub\wwwroot`

3. Copy the entire Rational Test 7\ClassicsTutorial folder from your installation CD into the root directory of the Web server. With Microsoft IIS this creates:

   `C:\InetPub\wwwroot\ClassicsTutorial`

Locating Internet Explorer

Before you begin the tutorial, you need to know the path of the application that you will be testing. In this case, the application is Microsoft Internet Explorer, and Classics Online is run within Internet Explorer.

Microsoft Internet Explorer is typically installed in the directory C:\Program Files\Plus!\Microsoft Internet\ iexplore.exe. However, if Internet Explorer is installed in a different location, you can use the Windows Find function to locate the file.

To locate the Internet Explorer executable:

1. Click **Start** → **Find** → **Files or Folders**.
2. Type **iexplore.exe** in the **Named** box.
3. Choose **Local Hard Drives** from the **Look in** list.

4. Make sure that **Include subfolders** is selected.
5. Click **Find Now**.

   Windows searches all hard drives and displays all matches in the Find dialog box.
6. Click the file you want. Then, right-click and choose **Properties** from the context menu.

   Write down the **Location** path of the file. You will need this path when you record a session.

Specifying Internet Explorer Settings

For the tutorial to run correctly, you need to set the Internet Explorer home page to blank and clear the Internet Explorer cache. To do this:

1. Start Internet Explorer.
2. If you are using Internet Explorer 5, select **Tools** → **Internet Options**. If you are using Internet Explorer 4, select **View** → **Internet Options.**
3. Make sure that the **General** tab is selected.
4. In the Home Page section, click **Use Blank**.
5. In the Temporary Internet Files section, click **Delete Files**.

   This clears the cache, so that during the tutorial you will be recording the actual traffic between the client and the server, rather than reading a Web page stored in cache. This does not harm your bookmarks or other settings; it only clears temporary files so that the tutorial runs correctly.
6. Click **OK** to confirm that you want to clear the cache.
7. Click **OK** to close the dialog box.

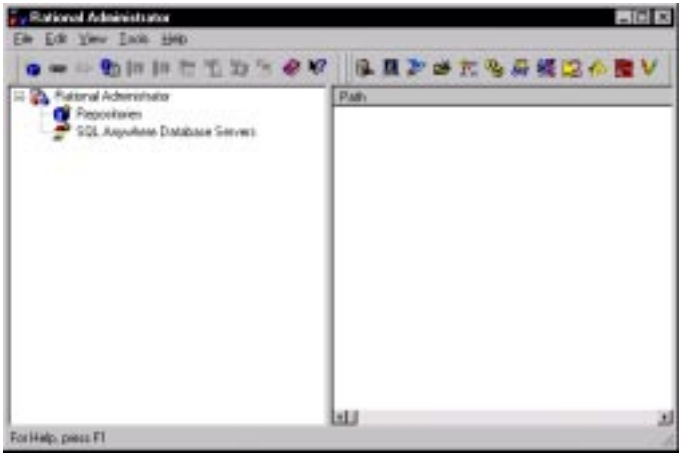
Setting Up a Repository for the Tutorial

Before you record a session, you must create a repository — a place to store all your test data.

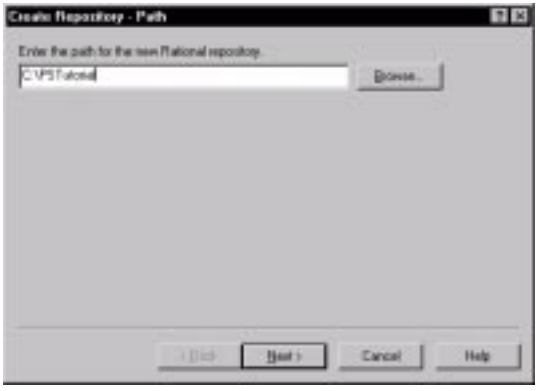
A **Rational repository** stores software testing and development information for your projects. All Rational test components on your computer update and retrieve data from the same active repository. The type of data in a Rational repository depends on the Rational software that you have installed. The data you create during this tutorial is stored in the repository you are about to create.

To create a Rational repository:

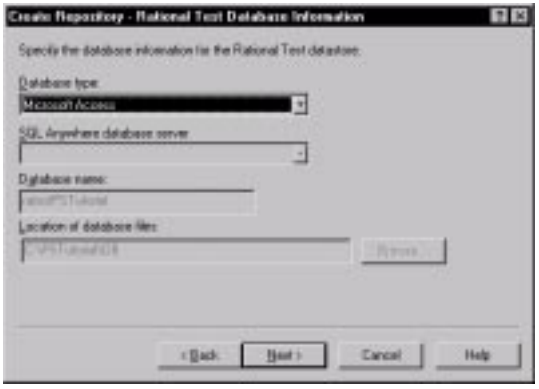
1. Click **Start** → **Programs** → **Rational Suite PerformanceStudio** → **Rational Administrator**.



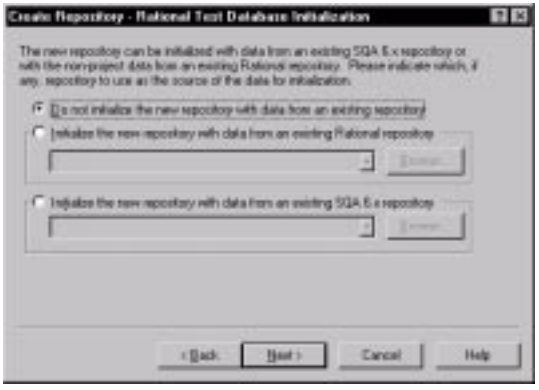
2. From the Rational Administrator, click **File** → **Create Repository**.
3. Type the drive and pathname for the repository, **C:\PSTutorial**, and click **Next**.



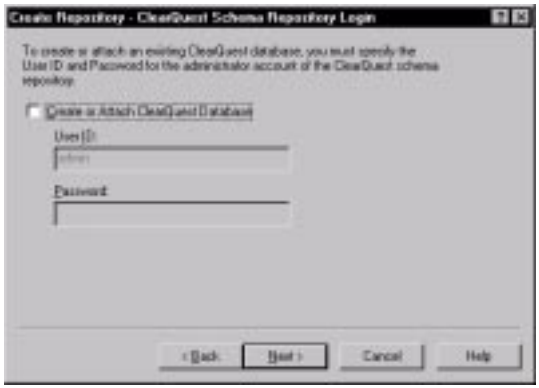
4. In the **Database type** list, select **Microsoft Access**, and click **Next**.



5. Click **Do not initialize the new repository using data from an existing repository**, and click **Next**.



6. Clear the **Create or Attach ClearQuest Database** check box, and click **Next**.



7. A summary page confirms the type and location of your new repository. Click **Finish** to create the new repository.

   The Rational Administrator will spend a few moments creating the repository structure.

8. When the Administrator completes its tasks, click **OK** in the confirmation dialog box.

9. Click **File** → **Exit** to exit from Rational Administrator.

▸▸▸ CHAPTER 3

Tutorial

This chapter provides a step-by-step tutorial for using Rational Suite PerformanceStudio. The tutorial covers the basic tasks that you follow when using Rational Suite PerformanceStudio, from planning the initial test to analyzing test reports. The tasks that this tutorial covers are as follows:

- Planning a test
- Setting recording options
- Recording scripts
- Examining the recorded scripts
- Generating realistic data to send to the server
- Designing a schedule to represent your workload
- Running the schedule
- Running reports

After you install all the required software elements, you can complete the various steps of the tutorial at your own pace. At any time while working in the tutorial, you can refer to other Rational documentation for more information.

Is This Tutorial for You?

Before you begin this tutorial, ask yourself the following:

- Are you new to PerformanceStudio, or to testing?
- Do you want to know more about the specific advantages of using PerformanceStudio?
- Do you need a reminder about how some of the basic components of PerformanceStudio fit together?

If you answered ‘yes’ to one or more questions, then this tutorial is for you.

Purpose of the Tutorial

This tutorial shows you how to test the performance of a Web-based e-commerce application. You will use PerformanceStudio tools, and Classics Online — a demonstration application developed by Rational Software — to perform the basic tasks involved in testing an application.

The tutorial enables you to adapt the testing process to your own needs. While the tutorial's initial steps walk you through each action required for setting up and running a performance test, the final steps encourage you to think independently and to apply what you have learned to your own testing environment.

Step 1: Planning a Test

The first step of the testing process is to clearly define what you want to test. In this section, you will:

- Identify the hardware and software required for testing.
- Map out your testing plan.

Identifying the Hardware and Software Required for Testing

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 servers

  This tutorial requires Microsoft Internet Information Server as your Web server. This server is installed from the Windows NT 4.0 Option Pack.

- **Client computers** – Windows NT, Windows 2000, Windows 98, or Windows 95 computers; network computers; or Macintosh or UNIX workstations

  This tutorial assumes that the test is set up and run from a Windows NT 4.0 server. Windows NT service pack 4 or later must be installed with the operating system.

- **Databases** that will be accessed

  This tutorial uses the default database that is set up when repositories are registered.

- **Applications** that will be running

  This tutorial assumes that the following applications are running: an Internet browser (specifically, Internet Explorer) and Classics Online.

Mapping Out Your Testing Plan

The Classics Online application simulates a music retailer selling classical CDs over the Web. In the tutorial, you use PerformanceStudio to test the performance of the Classics Online as customers search for and order CDs.

When you plan your tests, you need to think about the criteria that determine whether performance is acceptable, the types of users that access the server, and the types of activities that the users perform. For example:

- What are the pass and fail criteria for your tests? This tutorial assumes that when 10 users access your Web site, 90% of them must have a response time of 2 seconds or less. In the busy season, when you expect 20 users at a time to access your Web site, 80% of the users must have a response time of 6 seconds or less. No response time can exceed 8 seconds.
- What kinds of users access the server? This tutorial assumes that three groups of users access your Web site: 'window shoppers', 'regular buyers', and 'big spenders'.
- What kinds of activities do your groups of users perform? This tutorial assumes that **Window shoppers** browse through 15 Web pages and may order one CD. **Regular buyers** browse through 3 Web pages, and then order one CD. **Big spenders** browse through 15 Web pages and order 10 CDs.
- What are the proportions of your user groups? This tutorial assumes that 50 percent of your users are window shoppers, 40 percent are regular buyers, and only 10 percent are big spenders.

Where to Go for More Information

To learn more about planning tests, see Chapter 2 in the following manuals: *Using Rational Load Test* and *Using Rational Robot*.

Step 2: Setting Recording Options

In this section, you will:

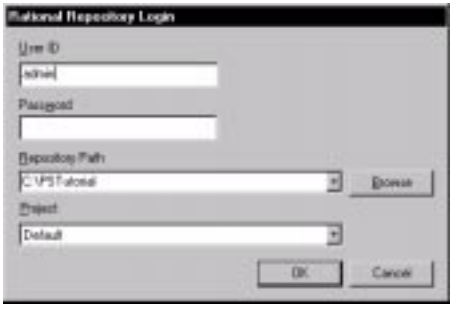
- Start Rational Robot.
- Set up recording options.

Starting Rational Robot

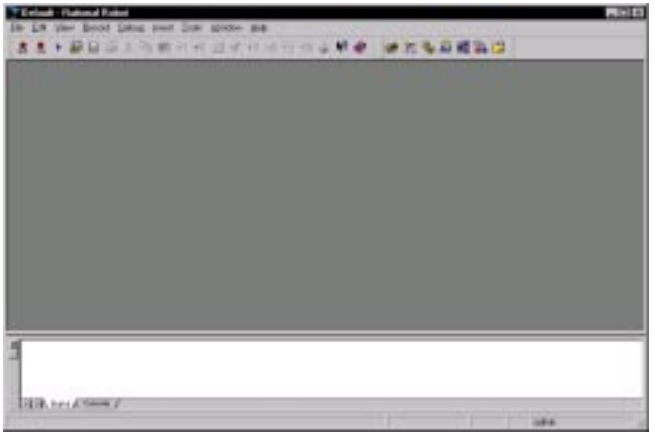
Rational Robot is the tool you use to record a script.

To start Rational Robot:

1. From the Start menu, click **Start** → **Programs** → **Rational Suite PerformanceStudio** → **Rational Robot**.



2. In the Rational Repository Login dialog box, type **admin** as your user ID. Make sure that the password is blank, that **PSTutorial** is selected as the repository, and that the project **Default** is selected. Then click **OK**.



Setting Recording Options

Before you begin recording, you'll set up some specific recording options. These options include the recording method that you want to use and the basic characteristics of the generated script.

To set virtual user recording options:

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

   API recording records application program interface calls between the specific client application — in this case, the browser — and the Web server.

NOTE: LoadTest may display a Virtual User Method Network Unavailable dialog box. This dialog box applies only to Network recording. You are using the API method to record, so click **OK** to dismiss the dialog box.

3. Click the **Generator** tab.
   a. In the **General** section, select **Use datapools**.

      A datapool lets a script send a different set of values to the server each time the script is played back in a test run. The **Use datapools** option initializes your script for datapool access. Later (in *Step 5: Generating Realistic Data to Send to the Server* on page 3-24), you'll modify the initialized datapool section of the script, and then create a datapool.

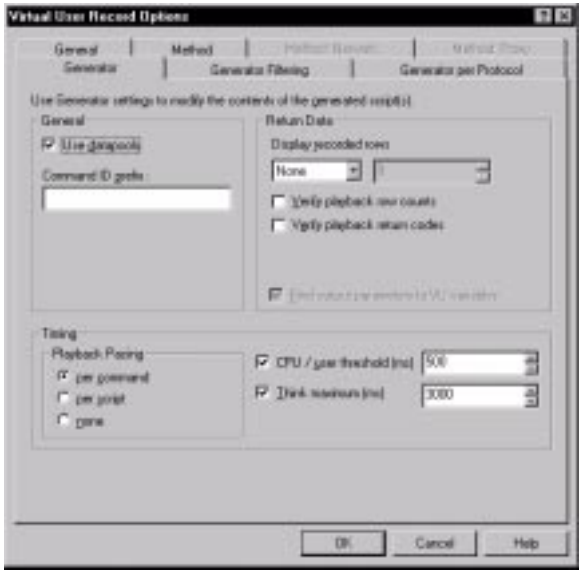
b. In the **Timing** section, click the **per command** button for **Playback Pacing**, and then at **CPU/user threshold (ms)**, enter **500**.

This option allows you to differentiate between delays in CPU processing time and delays in user think time. Setting the CPU/User threshold to 500 milliseconds (ms) means that LoadTest considers any delays over 500 milliseconds as user think time rather than CPU delays.

c. Also in the **Timing** section, select **Think maximum (ms)** and enter **3000** in the box.

Although this option does not change the think times that you will see in the generated script, it affects how LoadTest plays back the script. Setting the think maximum to 3000 milliseconds means that the script pauses for a maximum of 3 seconds during playback.

After you have finished setting options, the Generator tab should look like this:



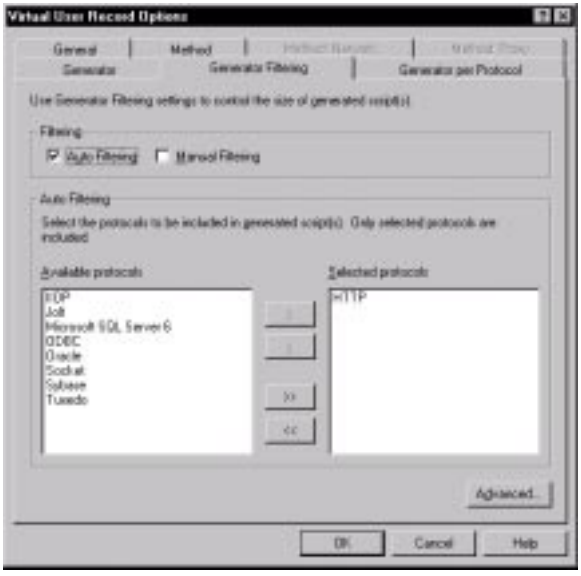
4. Click the **Generator Filtering** tab.

a. In the **Filtering** section, make sure that the **Auto Filtering** option is selected and the **Manual Filtering** option is cleared.

b. In the **Auto Filtering** section, the only selected protocol should be **HTTP**. If necessary, hold down the **CTRL** key while selecting all the other protocols in the **Selected protocols** list, and click **<** .

Robot will automatically capture requests for the selected protocols; in this case, HTTP.

After you have finished setting options, the Generator Filtering tab should look like this:



5. Click the **Generator per Protocol** tab. This tab lets you select options specific to the protocol you are recording.

In the **HTTP** section, select **Allow partial responses**, **Allow cache responses**, and **Allow redirects**.

These options let your script play back successfully if:

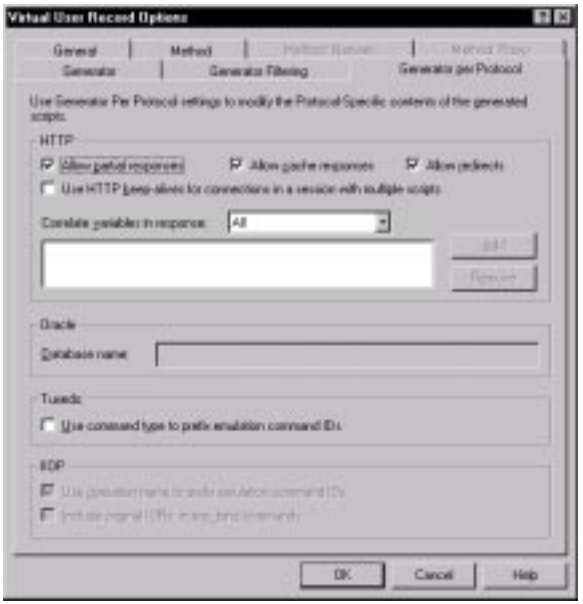
– The server responds with partial data during recording or playback.

– The response is cached during recording or playback.

– Your script is directed to another http server during playback.

Clear **Use http keep-alives for connections in a session with multiple scripts**.

After you have finished setting options, the Generator Per Protocol tab should look like this:



6. Click **OK**. You have finished setting the recording options.

Where to Go for More Information

LoadTest provides many recording options. This tutorial discusses the more widely used options. For more information about recording options, see Chapter 3 of the *Using Rational LoadTest* manual.

Step 3: Recording Scripts

You are ready to record scripts. In this section, you will:

- Start recording.
- Split the recording session into four modular scripts:
  - an Open script, which will contain the client/server traffic that occurs when you open the Classics Online home page.
  - a Browse script, which will contain the client/server traffic that occurs when you search through Classics Online.

  - an Order script, which will contain the client/server traffic that occurs when you order a CD.
  - a Close script, which will contain the client/server traffic that occurs when you close the Classics Online application.

- Add comments and other items to the scripts.
- Stop recording.

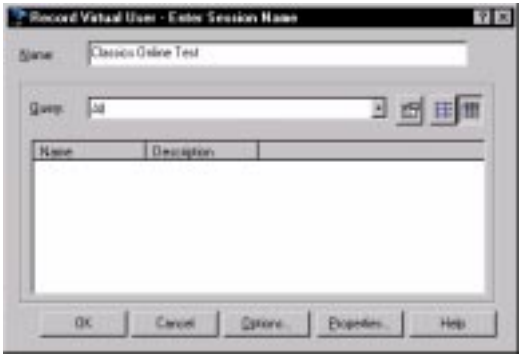
Although you shouldn't press the stop recording button until the tutorial instructs you to do so, you can take a break in the middle of your recording session, while the recording is still running. For example, even if you leave your desk for an hour during recording, when you play back the script, it will pause for only 3 seconds. This is because you have set the Think Maximum to 3 seconds.

Starting the Recording Session

To record a session of Classics Online:

1. In Robot, click **File** → **Record Virtual User**.
2. Type the session name, **Classics Online Test**.

   You will specify script names within the session as you finish recording each script.



3. Click **OK**.

   – Robot is minimized (default behavior).

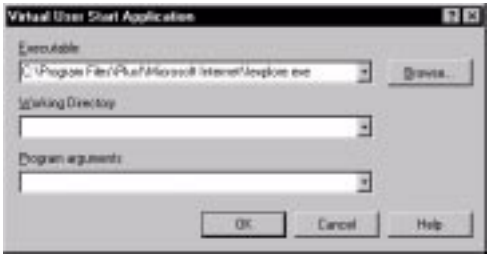
   – The floating VU Record toolbar appears (default behavior). You will use this toolbar to stop recording, redisplay Robot, split a script, and insert features into a script.

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

   – Since the client application is not running, the Virtual User Start Application dialog box appears.

4. Specify the path of the executable file for the browser, as noted earlier in *Locating Internet Explorer* on page 2-2.

   The default location is:
   `C:\Program Files\Plus!\Microsoft Internet\Iexplore.exe`



5. Click **OK**. Robot starts the browser.

6. From the browser, open the Classics Online application.

   The default URL is:

   `http://`*`servername`*`/ClassicsTutorial/default.htm`

   where *servername* is the network name of the computer running the Web server.

NOTE: Be sure to type this URL in the **Address** box of Internet Explorer.

7. The browser displays the Classics Online home page:

Splitting a Script

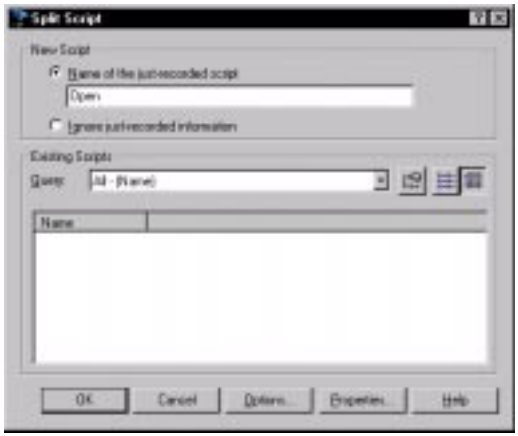
After the sample application Classics Online starts, you'll 'split the script'. Splitting scripts makes your scripts more flexible. For example, instead of recording one long session, you will split the session into four scripts: an Open script, a Browse script, an Order script, and a Close script. Then, when you design your schedule, you can rearrange these modular scripts to emulate the work that each user does.

The first script that you'll split is the Open script. To split the script:

1. On the opening page of Classics Online, wait until the Virtual User Recorder icon on the toolbar stops blinking. This indicates that Robot has stopped capturing requests and responses.

   Then, click the **Split Script** button on the VU Record toolbar.

2. In the Split Script dialog box, type **Open**. This is the name of the script that you just recorded.



3. Click **OK**. The focus returns to Classics Online.

Adding a Comment

Comments are helpful additions to scripts. This is because one user action — such as clicking a search button on a Web page — can translate into more than one command in a script. Adding comments to a script during recording makes the script easier to understand and makes the commands in the script easier to map to user actions.

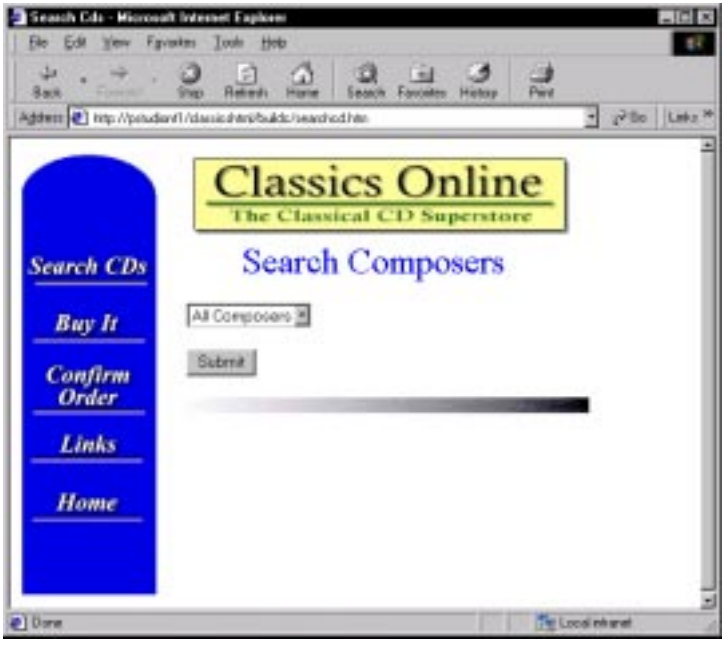
In this section, you'll add a comment, and then you'll browse through Classics Online. To add a comment:

1. Click the **Display VU Insert Toolbar** button on the VU Record Toolbar.

2. Click the **Comment** button.
3. In the Comment dialog box, type **Browsing All CDs**, and then click **OK**.

NOTE: It is a good idea to place relevant comments throughout the script. As you go through the rest of these steps for recording a session, feel free to add comments where you feel they are appropriate.

4. From the Classics Online application, click **Search CDs**.
5. Select **All Composers** from the **Search Composers** list, and then click **Submit**.



Splitting the Script

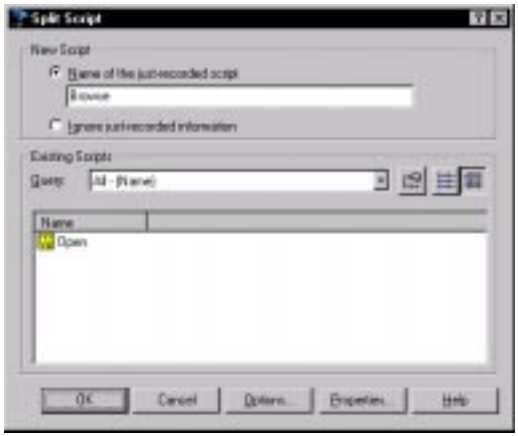
Once again, splitting the script differentiates this task from the remaining tasks you will record. This time, you'll split the Browse script.

To split the script:

1. Wait until the Virtual User Recorder icon on the toolbar stops blinking. This indicates that Robot has stopped capturing requests and responses.

   Then, click the **Split Script** button in the VU Record toolbar.

2. In the Split Script dialog box, type **Browse** for the script that you just recorded.



3. Click **OK**. The focus returns to the Classics Online application.

Adding a Block

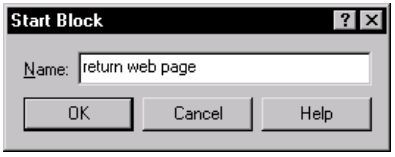
In this section, you'll order a CD. You'll fill in a name and address, add a block, and then fill in credit card information.

A block allows you to set off a certain amount of code from the rest of the script. A block can be used to set off a transaction, but in this instance, you'll use a block to determine how long it takes the server to return a Web page.

1. From the Classics Online application, click the back arrow to return to the Search page.
2. Click the **Buy It** button to order a CD.

3. Type a name, address, and phone number, but *do not* click Submit:

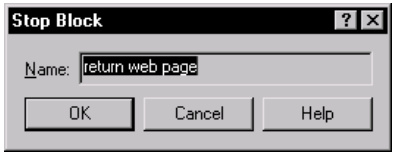
4. From the VU Insert Toolbar, click the **Start Block** button.
5. In the Start Block dialog box, type **return web page**, and then click **OK**.



6. Click **Submit**.

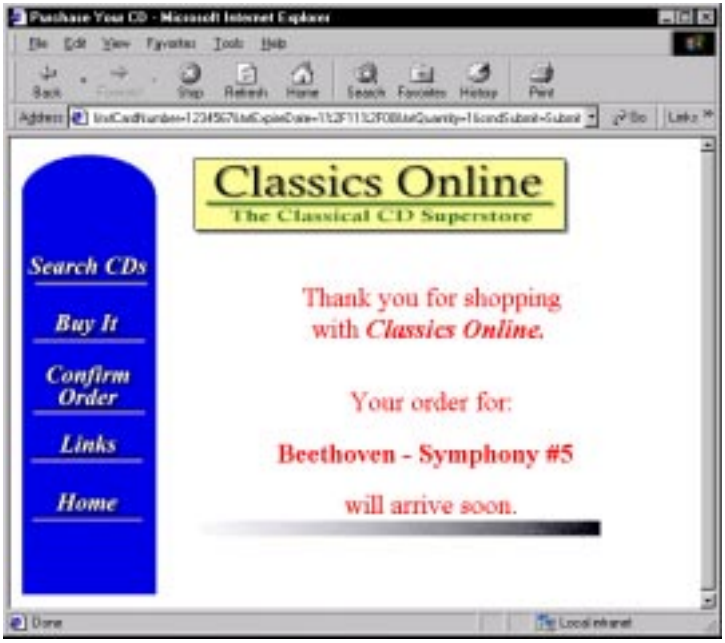
7. To stop the block, from the VU Insert Toolbar, click the **Stop Block** button.

The name of the block, **return web page**, is displayed in the Stop Block dialog box. Click **OK**.



8. Fill in the credit card information and quantity, and then click **Submit**. You have just ordered a CD from Classics Online.

9. Classics Online displays the confirmation screen:



Splitting the Script

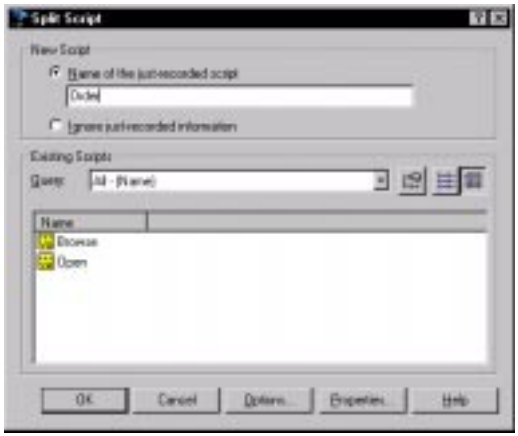
Once again, you'll split the script. This time, you'll split the Order script.

To split the script:

1. Wait until the Virtual User Recorder icon on the toolbar stops blinking. This indicates that Robot has stopped capturing requests and responses.

   Then click the **Split Script** button in the VU Record toolbar.

2. In the Split Script dialog box, type **Order** for the script that you just recorded.



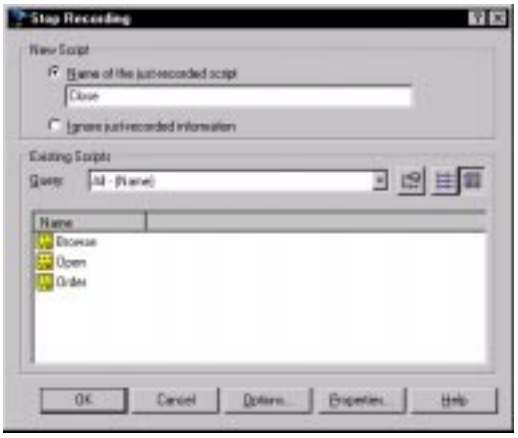
3. Click **OK**. The focus returns to the Classics Online.

Finishing the Session

1. Exit Classics Online. From the Classics Online Web page, click **File** → **Close**.
2. Robot displays a dialog box that asks if you want to stop recording.

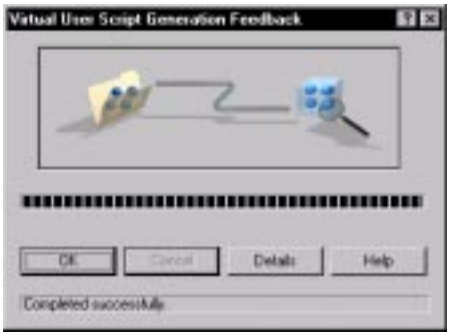
   Click **Yes**.

3. In the Stop Recording dialog box, type **Close** for the name of the script that you just recorded.

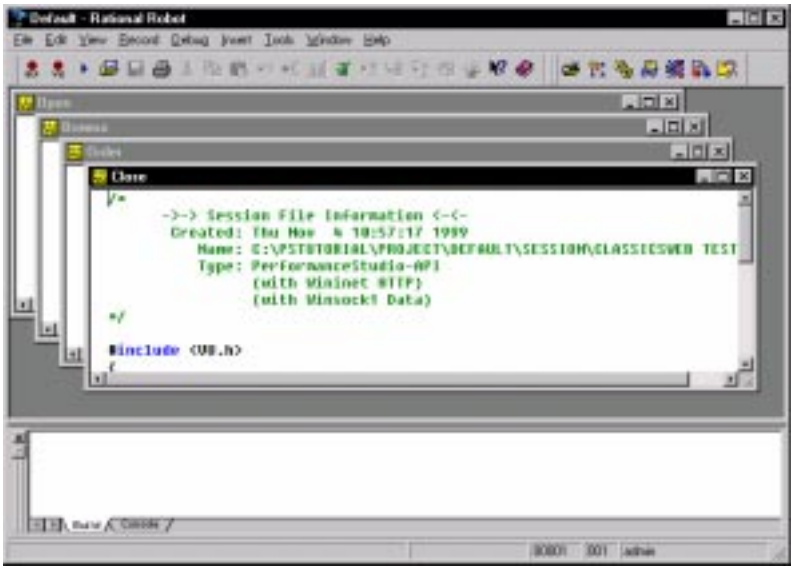


4. In the Stop Recording dialog box, click **OK**. Robot generates the scripts for the session.

5. When the script generation is complete, click **OK**.



Robot displays the scripts that you recorded in the session — Open, Browse, Order, and Close.



6. In the next section, you'll examine these scripts more closely. But for now, click **File** → **Close** to close each open script.

Where to Go For More Information

For more information about setting recording options, see Chapter 3 of the *Using Rational LoadTest* manual.

For more information about recording a virtual user script in Rational Robot, see Chapter 4 of the *Using Rational LoadTest* manual.

If you would like to take a break while performing the tutorial, now is a good time. You can come back and finish the remainder of the tutorial later.

Step 4: Examining the Recorded Scripts

Now that you have finished recording the session, you can view the generated scripts. This section shows some of the important features that you'll see in a VU (Virtual User) script.

To examine a script:

1. Click **File** → **Open** → **Script.**
2. From the Open Script dialog box, select **Open**, and click **OK**.

   At the top of the script, Robot has added some general comments, such as when you created the session and the recording method that you set. These comments — in fact, all comments — appear in green.

   Scroll down through the script. The VU language reserved words appear in blue.
3. You'll also see, grouped together near the beginning of the Open script, VU environment variables. Robot set most of them by default, and they are set with the VU `push` command.

   You'll also see some environment variables that you set before you started recording. When you selected a maximum think time of 3000 ms, you actually set a VU environment variable — `Think_max` — that shows up in your script. To find `Think_max`:

   a. Click **Edit** → **Find**.

   b. In the Find dialog box, type **Think_max.**

   c. Click **Find Next**.

   Recall that you also selected HTTP options and a Think/CPU threshold. These selections also set VU environment variables — in this case, `Http_control` and `Think_cpu_threshold`.
4. When you opened the Classics Online page, you sent requests to the server and received information back from the server. These requests and responses are captured in VU emulation commands, specifically `http_request` and `http_nrecv`.

   To find `http_request`:

   a. Click **Edit** → **Find**.

   b. In the Find dialog box, type **http_request**.

   c. Click **Find Next**.

5. Notice the bracketed string of text that immediately follows the `http_request` command. This bracketed text is called a **command ID**. These command IDs uniquely identify each VU emulation command. Because each emulation command is uniquely identified, you can report on it and isolate the source of any performance problem. Command IDs are a central concept of performance testing.

   Robot adds a command ID to each VU emulation command in the script in the form of `["Open`*n*`.00`*n*`"]`.

   – The first part of the command ID is the name of the script — in this case, Open.

   – The *n* is a number to further guarantee uniqueness.

   – the `00`*n* is incremented with each emulation command.

   Thus, in the Open script, the first emulation command is identified by the command ID `Open1.001`, the second by command ID `Open1.002`, and so on.

   NOTE: A command ID may not always follow this exact format. For example, if your script names are very long, the command ID shows an abbreviated script name and a tilde. The important concept to remember is that the command ID always uniquely identifies an emulation command.

   To find the command IDs in the Open script:

   a. Click **Edit** → **Find**.

   b. In the Find dialog box, type **Open**.

   c. Click **Find Next**.

6. During recording, you added a comment to the Browse script. To see this comment, open the Browse script, and search for `Browsing All CDs`.

   a. Click **File** → **Open** → **Script.**

   b. From the Open Script dialog box, select **Browse**, and then click **OK**. The Browse script opens.

   c. Click **Edit** → **Find**.

   d. In the Find dialog box, type **Browsing All CDs**, and then click **Find Next**.

   Both the comments that you enter and the comments that Robot enters appear in green, and are set off from the rest of the script with a leading `/*` and a closing `*/`.

7. Recall that you added a block during the recording session. When you added the block, you actually changed the command IDs in the script. To see how they have changed, open the Order script and search for **Start_Block**.

   The command IDs within the `return web page` block are named according to the block — `return 001`, `return 002` — and so on, rather than the script. This enables you to report a transaction that you define instead of an individual command ID.

   In addition, Robot added a `start_time` command at the beginning off the block, and a `stop_time` command at the end of the block. Think of this timer as a stopwatch that you clicked on just before you left the first Web page, and clicked off after the server returned the next Web page. Later on, you'll be able to report on this block.

8. When you set recording options before you recorded the session, you clicked **Use datapools**. During recording, Robot automatically generated a DATAPOOL_CONFIG section in the script. To see the DATAPOOL_CONFIG section of the Order script, scroll down to the end of the script. You see the following lines:

```
DATAPOOL_CONFIG "Order"OVERRIDE DP_NOWRAP DP_SEQUENTIAL DP_SHARED
{
EXCLUDE, "cboCreditCard", "string", "Visa";
EXCLUDE, "cmdSubmit", "string", "Submit";
EXCLUDE, "txtAddress", "string", "123 Pleasant Street";
EXCLUDE, "txtCardNumber", "string", "1234567";
EXCLUDE, "txtCity", "string", "Any Town";
EXCLUDE, "txtExpireDate", "string", "1/11/00";
EXCLUDE, "txtName", "string", "John Smith";
EXCLUDE, "txtPhone", "string", "123-4567";
EXCLUDE, "txtQuantity", "string", "1";
EXCLUDE, "txtState", "string", "USA";
EXCLUDE, "txtZip", "string", "12345";
}
```

   The DATAPOOL_CONFIG section sets the framework of the datapool. It contains information such as:

   – The names of script variables and the literal values that were supplied to the variables during recording.

   – Instructions on whether each variable should use data from the datapool during script playback.

   – Directives that describe how data should be retrieved from the datapool during script playback.

   The datapool is dormant until you set it up. The next section shows how to set up a datapool, so that when the script runs, it will send realistic data to the server.

Where to Go For More Information

For more information about script structure, emulation commands, and VU environment variables, see the *VU Language Reference*. For more information about blocks, see the *Using Rational LoadTest* manual.

Step 5: Generating Realistic Data to Send to the Server

The actual data that you send to the server has a great effect on performance.

In the real world, where users have different names, the server has to retrieve and store each different name in some back-end database. The database has to do a lot of work to retrieve and store each different name, which is why Web applications may start to get slow.

Recall that when you recorded the session, you filled in the name of one user who ordered a CD. If you run a performance test and every user has the same name, the backend database has to retrieve and store only one name. Because the database can put this single name in a very fast area of memory known as a **cache**, the reports you run will tell you that you have a very fast system. (This is known as a "false positive" test result, since the test is falsely telling you that the system is performing as quickly as expected.)

To fix the problem, LoadTest uses **datapools**. Datapools supply different names to the server, which give a realistic idea of an application's performance under real-world conditions.

This section tells you how to use datapools to emulate many users with different names ordering CDs from Classics Online.

1. If you have closed Rational Suite PerformanceStudio, do the following:
   a. Click **Start** → **Programs** → **Rational Suite PerformanceStudio** → **Rational Robot**.
   b. Click **File** → **Open** → **Script**, select the Order script from the Open Script dialog box, and then click **OK**.
2. Click **Edit** → **Datapool Information**.

   The Configure Datapool in Script dialog box appears. This dialog box lets you modify the `DATAPOOL_CONFIG` section of the script.

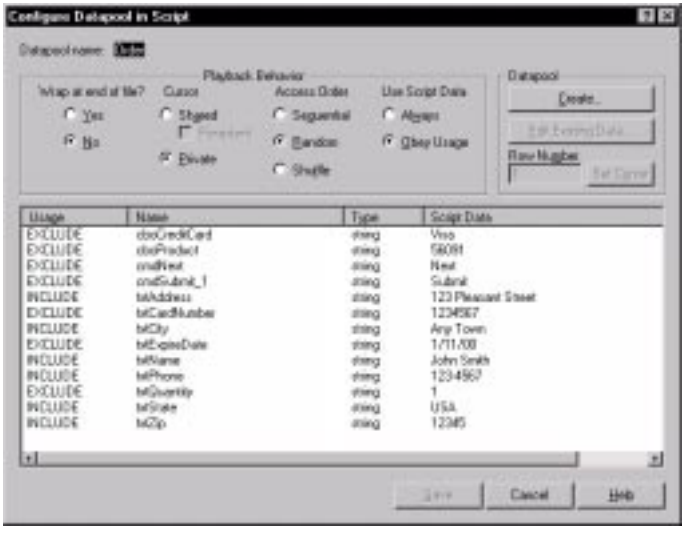
3. Select the following options for **Playback behavior**:
   - Under **Cursor**, click **Private**. Each user will have its own cursor.
   - Under **Access Order**, click **Random**. Each row of the data will be accessed in any order, and a row can be accessed multiple times or not at all.
   - Under **Use Script data**, click **Obey Usage.** This option tells the script to obey the command in the Usage column in the grid below to determine the source of each variable's data.

4. You'll notice that, by default, the Usage column lists **EXCLUDE** for each variable. This is because Robot has generated a datapool framework in your script but hasn't yet instructed LoadTest to supply the variables with data from a datapool. This default setting lets you run the script before the datapool is ready for use.

   In this tutorial, you'll supply script variables with the name and address data that Classics Online displayed just after you clicked the **Buy It** button: the name, address, city, state, zip code, and phone number. You set this up in the script as follows:

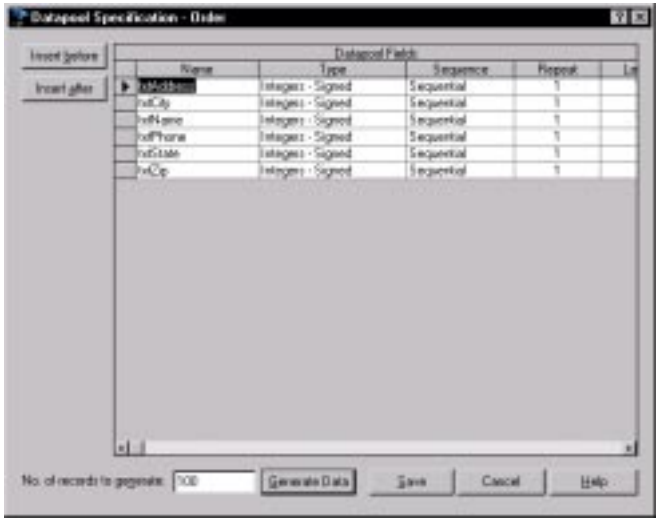
   a. To supply the variable **txtName** with names from the datapool, right-click on the word **EXCLUDE** in the **txtName** row, and then select **INCLUDE**.

   b. Continue selecting **INCLUDE** for the address, city, phone number, state, and zip code rows. The Configure Datapool in Script dialog box should look like this:



5. You are now ready to create the datapool. In the Datapool section, click **Create**. A dialog box appears with the message **The script has been saved to reflect any changes you have made**. Click **OK**.

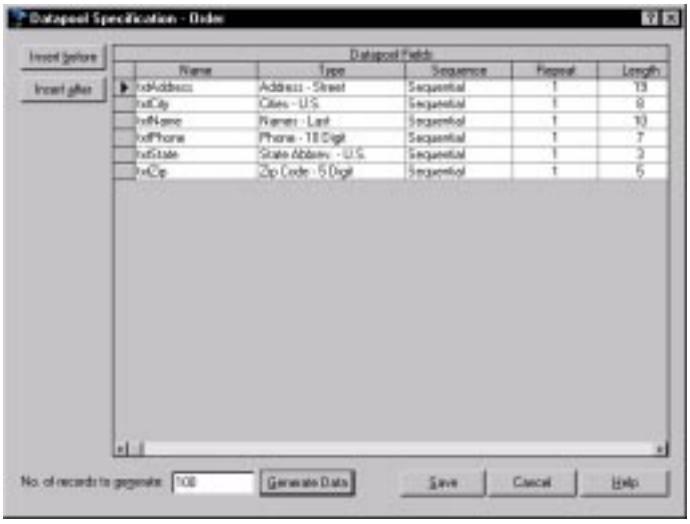
6. The following dialog box appears:



7. In the first row of the grid (which defines the datapool column **TxtAddress**), select the contents of the **Type** cell, and then click the down arrow to see a list of available data types.

8. Scroll through the list and select **Address-Street**.

9. Continue selecting data types for each datapool column name. When you have finished, the dialog box should look like this:



10. At **No. of records to generate**, type **100**, and then click **Generate Data**.
11. Click **Yes** to confirm that data generation is complete.
12. Click **Cancel** in the Data Generator Results Detail dialog box.
13. Click **Close** in the Datapool Specification dialog box.
14. Click **Close** in the Configure Datapool in Script dialog box.

Where to Go For More Information

The previous section showed how to add a datapool so that LoadTest will provide realistic values to the server during script playback. Thus, if script playback emulates 100 users accessing a database on the server, each user can send a different record to the server. Without a datapool, the same record (the values captured during recording) would be sent to the server 100 times.

Note that the datapool that you just created used values that LoadTest automatically generated. You can also create a datapool with user-defined data. For example, if your application allowed you to order CDs from different composers, you could create a datapool and populate it with the names of different composers, so that your test could emulate users ordering different CDs.

See Chapter 5 of the *Using Rational LoadTest* manual for more information about:

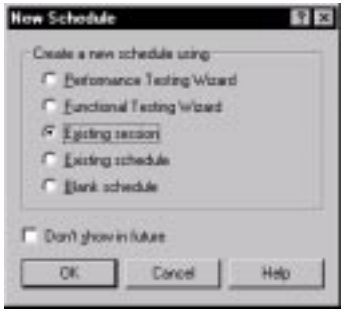
- Defining and populating datapools with user-defined data types
- Creating and managing more complex datapools within virtual user scripts

Step 6: Designing a Schedule to Represent Your Workload

Now that you have finished recording the session, you are ready put your scripts into a schedule in such a way that they emulate your user workload.

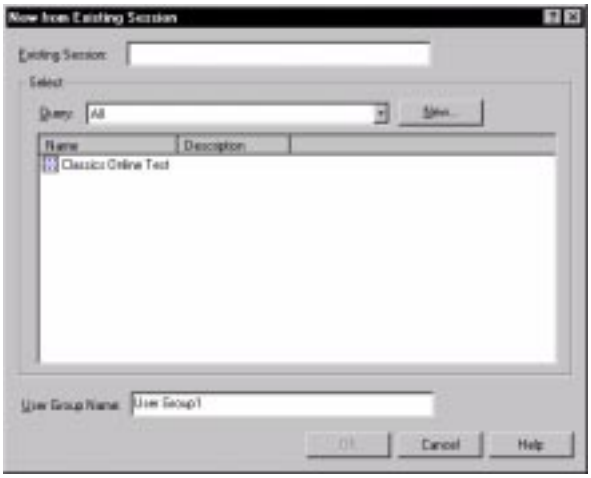
To create a schedule:

1. Open Rational LoadTest.
   a. If you have closed Rational Suite PerformanceStudio, click **Start** → **Programs** → **Rational Suite PerformanceStudio** → **Rational LoadTest**.
   b. If you are still in Robot, click **Tools** → **Rational Test** → **Rational LoadTest**.
2. In LoadTest, click **File** → **New** → **Schedule**. The New Schedule dialog box appears.



3. Click **Existing Session**, and then click **OK**.

   In the New from Existing Session dialog box, LoadTest displays a list of sessions that you have recorded.



4. Select **Classics Online Test**, and click **OK**.
5. LoadTest displays a schedule, which contains the scripts that you recorded in the session.

LoadTest automatically creates a schedule that is ready to run. This schedule has one user group associated with it, **User Group1**, and one scenario.

The scenario has the same name as the session, and it contains the scripts in the order that you recorded them. In general, a scenario lets you group scripts and other elements together so that they can be shared among user groups. If a schedule is complicated and uses many scripts, grouping the scripts under a scenario makes the schedule easier to read and maintain.

Although you can run this schedule — and in a real test, it's a good idea to do so to confirm that your scripts play back correctly — in the interest of brevity, this tutorial will skip this step.

But even though the schedule is ready to run, it does not reflect your environment. If you ran this schedule, one user group, **User Group1**, would open Classics Online, browse through the composers, place one order, and close Classics Online.

However, your environment contains three groups of users that access your Web site: 'window shoppers,' 'regular buyers,' and 'big spenders.'

In addition, each user group behaves differently:

- Window Shoppers, which comprise 50 percent of your total users, browse through 15 Web pages and may or may not order any CDs.
- Regular Buyers, which comprise 40 percent of your total users, browse through three Web pages, and then order a CD.
- Big Spenders, which comprise only 10 percent of your total users, browse through 15 Web pages and place 10 orders at random intervals.

The following section shows how to design a schedule that emulates this workload.

Inserting User Groups into Your Schedule

In this section, you will insert three user groups — Window Shoppers, Regular Buyers, and Big Spenders — into the schedule.

To add these user groups:

1. Click the User Groups icon in the schedule.
2. Click **Insert** → **User Group**.
3. In the User Group dialog box:
   - Type **Window Shoppers**, the name of the user group.
   - Click **Scalable**, and enter the percentage of your customers that are window shoppers, **50**.

4. Click **OK**. The user group **Window Shoppers** appears in the schedule.
5. Continue adding the two remaining user groups.
   - Click the **Window Shoppers** user group, and repeat steps 2–4 to create a user group called Regular Buyers with a scalable percentage of 40%.
   - Click the **Regular Buyers** user group, and repeat steps 2–4 to create a user group called Big Spenders with a scalable percentage of 10%.
6. Remove **User Group1** by right-clicking on it, and then clicking **Delete**.

After you've added user groups, your schedule should look like this:

You are now ready to add scripts and selectors to the schedule.

Inserting Scripts and Selectors into Your Schedule

Now that you have inserted user groups into the schedule, you'll add the scripts that each user group will run. You will insert the Open, Browse, Order, and Close scripts so that they emulate the behavior of each user group. You will also add some random selectors, which will select the scripts to run, because not all scripts will run sequentially.

In your environment:

- All users begin by opening Classics Online and end by closing Classics Online. Therefore, each user will run the Open script first, and will run the Close script last.
- Window Shoppers browse through 15 Web pages and may or may not order any CDs. Therefore, they will run the Browse script 15 times, and may or may not run the Order script.
- Regular Buyers browse through three Web pages, and then order a CD. Therefore, they will run the Browse script three times, and will then run the Order script once.
- Big Spenders browse through 15 Web pages and place 10 orders at random intervals. Therefore, they will randomly run the Browse script 15 times and randomly run the Order script 10 times.

Copying the Open and Close Scripts into the User Groups

You'll start by copying the Open and Close scripts into each user group. To copy these scripts:

1. Select both the Open and Close scripts by pressing the **CTRL** key and clicking the left mouse button.
2. Right-click, and then click **Copy.**
3. Right-click on the **Window Shoppers** user group, and click **Paste**. The Open and Close scripts are inserted into the Window Shoppers user group. Expand the the tree so that you can see them in the schedule.
4. Right-click on the **Regular Buyers** user group, and click **Paste**. The Open and Close scripts are inserted into the Regular Buyers user group.
5. Right-click on the **Big Spenders** user group, and click **Paste**. The Open and Close scripts are inserted into the Regular Buyers user group.

In the following sections, you'll add the Browse and Order scripts. However, each user group runs these scripts differently. Therefore, you need to add a selector to randomly choose which script to run, and to then weight each script to increase or decrease its chances of being chosen.

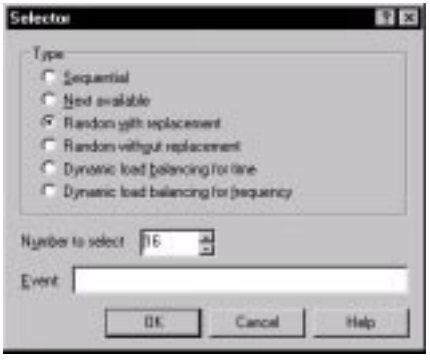
Emulating the Window Shoppers Workload

Recall that customers in the Window Shoppers user group browse through 15 Web pages and may or may not order any CDs. If they order a CD, they sometimes continue browsing.

To reflect this behavior, you insert a random with replacement selector into your schedule, and place the Browse and Order scripts under that selector. A random with replacement selector runs the items under it — in this case, the Browse and Order scripts — in random order. Each time an item is selected, the odds of it being selected again remain the same.

To insert a selector:

1. Click the **Open** script in the Window Shoppers user group.
2. Click **Insert** → **Selector**.



   In the Selectors dialog box, click **Random with replacement**.

   In **Number to select**, enter 16. This means that the selector will select a script randomly 16 times during the schedule run.

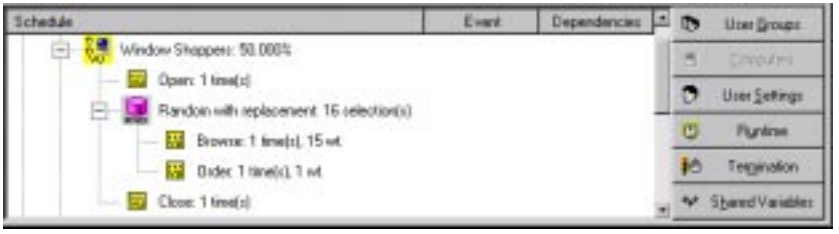
3. Click **OK.** LoadTest adds the selector under the Open script.

Next, you'll add the Browse and Order scripts under the selector.

1. Click the selector that you just added in the Window Shoppers user group.
2. Click **Insert** → **Script**, and select the Browse script.

3. At **Weight**, enter **15**. This is because Window Shoppers browse through 15 Web pages. This increases the chance that this script will be chosen by the selector.
4. Click **OK**.
5. Click the **Browse** script in the Window Shoppers user group.
6. Click **Insert** → **Script**, and then select the **Order** script.
7. At **Weight**, enter **1**, and then click **OK**. Because the weight of this script is 1 (and the weight of the

The Window Shoppers user group should look like this:



Emulating the Regular Buyers Workload

Recall that customers in the Regular Buyers user group browse through three Web pages, and then place one order.

To reflect this behavior, you'll add the Browse and Order scripts under the Open script:

1. Click the **Open** script in the Regular Buyers user group.
2. Click **Insert** → **Script**, and select the **Browse** script.

3. At **Iterations**, enter **3**. This is because Regular Buyers will browse through three Web pages.
4. Click **OK**.
5. Click the **Browse** script in the Regular Buyers user group.
6. Click **Insert** → **Script**, select the **Order** script, and then click **OK**.

Emulating the Big Spenders Workload

Recall that customers in the Big Spenders user group browse through 15 Web pages and order 10 CDs — a ratio of 3 'browses' to 2 'orders'. They browse and order the CDs in any order.

To reflect this behavior, you insert a random without replacement selector into your schedule, and place the Browse and Order scripts under that selector. A random without replacement selector runs the items under it — in this case, the Browse and Order scripts — in random order. However, each time an item is selected, the odds of it being selected again change.

To insert a selector:

1. Click the **Open** script in the Big Spenders user group.
2. Click **Insert** → **Selector**.

   In the Selectors dialog box, click **Random without replacement**.

   In **Number to select**, enter **25**. This means that the selector will select a script randomly 25 times during the schedule run.

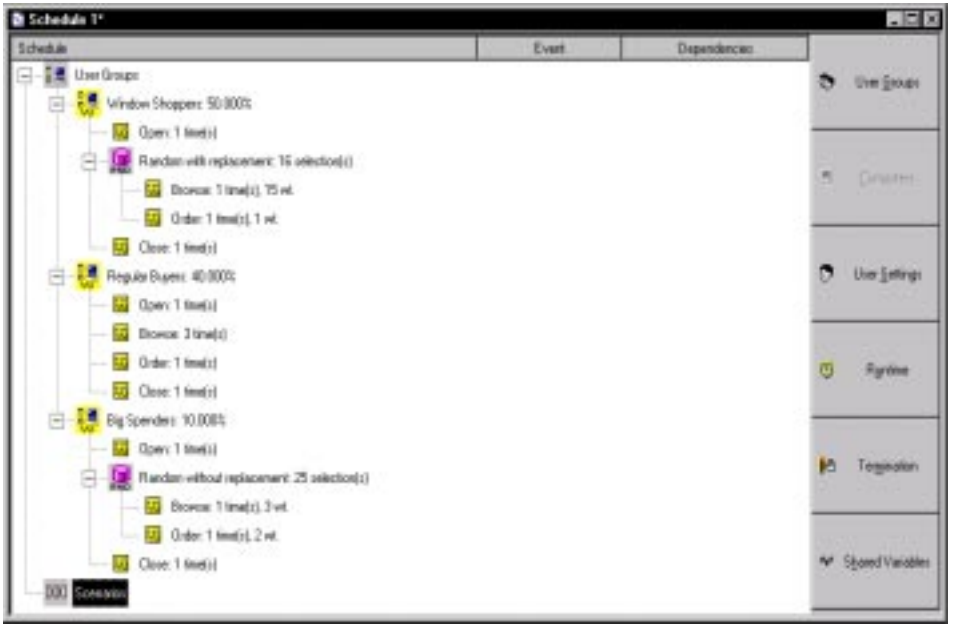
3. Click **OK.** LoadTest adds the selector under the Open script.

Next, you'll add the Browse and Order scripts under the selector.

1. Click the selector that you just added in the Big Spenders user group.
2. Click **Insert** → **Script**, and select the **Browse** script.

3. At Weight, enter **3**. This is because Big Spenders browse through 15 Web pages for every 10 orders, which is a ratio of 3:2.
4. Click **OK**.
5. Click the Browse script in the Big Spenders user group.
6. Click **Insert** → **Script**, and then select the Order script.
7. At Weight, enter **2**, and then click **OK**.

After you have added scripts and selectors, your schedule should look like this:

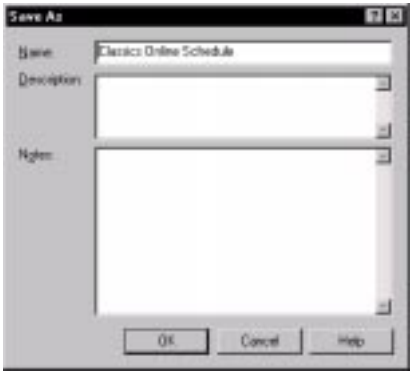


Saving the Schedule

After you have finished setting up the schedule, save it.

To save a schedule

1. Click **File** → **Save**.
2. Enter a name for the schedule, **Classics Online Schedule**, in the Save As dialog box.



3. Click **OK**.

Where to Go For More Information

For more information about designing schedules, see Chapter 7 of the *Using Rational LoadTest* manual. The chapter includes detailed information about user groups, selectors, and other elements that you can insert into a schedule.

Step 7: Running the Schedule

Now that your schedule reflects the browsing and ordering patterns of Classics Online customers, it is time to actually run it.

In this section, you'll do the following:

- Set the information that LoadTest displays when the schedule runs.
- Run the schedule with 10 users.
- Briefly examine the default reports that LoadTest generates after a schedule run.

Setting the Information You See During a Schedule Run

Before you run the schedule, you'll set up LoadTest so that it displays information specific to HTTP testing. To set the information that LoadTest displays:

1. Click **Tools** → **Options**, and then click the **Monitor** tab.
2. Under **User**, select the **Results** box, and clear any other boxes that are selected. This displays information geared to the success or failure of each emulation command.

3. Under **State Histogram**, select the **HTTP** box, and clear any other boxes that are selected. This displays a bar chart oriented to scripts that access Web data.
4. Under **Misc**, clear any boxes that are checked.

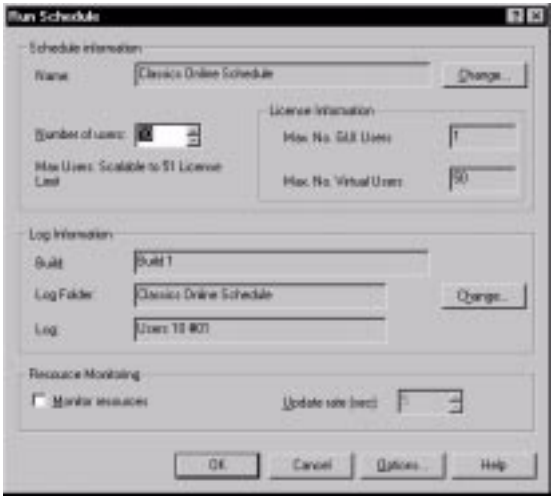
   When you are finished, the dialog box should look like this:

5. Click **OK** to save these settings.

Running a Schedule

To run a schedule with 10 users:

1. Click **Run** → **Schedule**.



   In the Run Schedule dialog box, the schedule name is displayed in the **Name** box.

2. In the **Number of Users** box, enter **10**.

   This will run the schedule with 10 virtual users and distribute them among the user groups in the percentages that were set when you created the user groups: 5 Window Shoppers, 4 Regular Buyers, and 1 Big Spender.

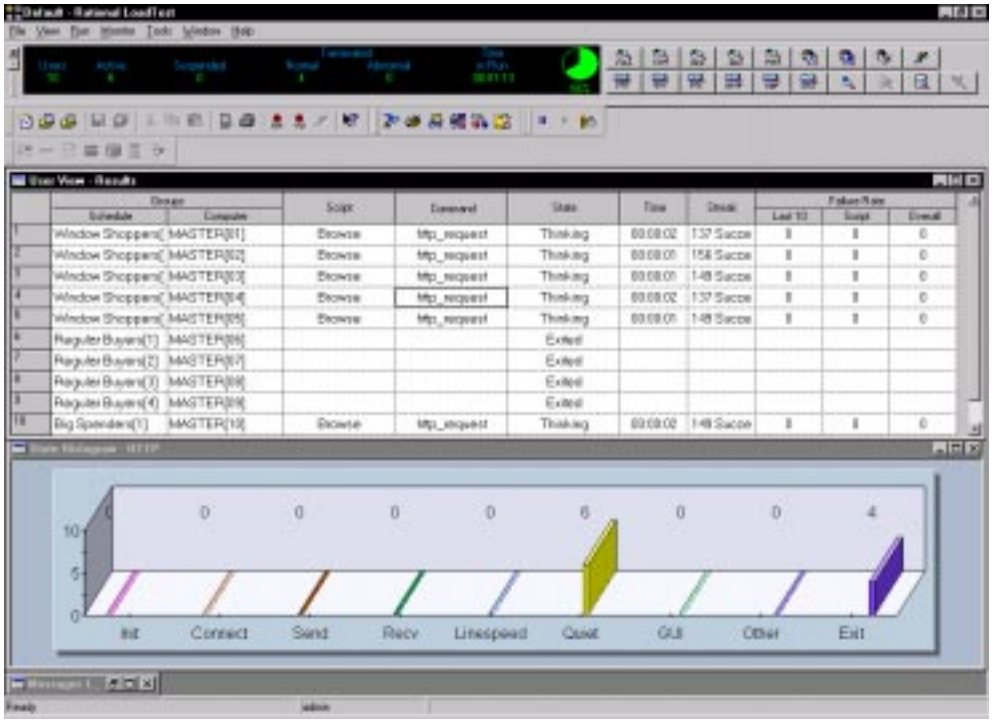
3. Click **OK**. If necessary, LoadTest compiles your scripts and then begins to run the ClassicsWeb schedule.

Monitoring the Schedule

While the schedule is running, it automatically displays the two views that you set — the Results User view and the HTTP Histogram view.

To monitor the schedule:

- Maximize the LoadTest window and click **Windows** → **Tile horizontal** to make sure you'll be able to see the information.



- The dark area at the top of the schedule is the Progress bar. This lets you quickly asses how successfully the schedule is running. In this figure, the circle shows that you are about two-thirds (66%) through with the schedule run.
- The Results User view displays information about the state of each script that is running. The runtime scripts are also called **virtual users**. In this figure, you can see all 10 virtual users. Six are running the Browse script, and four have already exited.

  When you run your schedule, you will notice that the Regular Buyers will probably be the first virtual users to exit. These virtual users perform the fewest iterations of the scripts. The Window Shopper will be the last virtual user to exit, because it performs the most iterations.
- The HTTP Histogram view is a bar chart that gives specific information about the state that each virtual user is in.

In the HTTP Histogram view, data is grouped appropriately for tests that access Web servers. The following table shows the information displayed in this view:

Bar Name	Description
Init	Users that are initializing.
Connect	Users that are waiting to connect to the Web server.
Send	Users that are sending data to the Web server.
Recv	Users that are waiting for data from the Web server.
Linespeed	Users that are being artificially delayed to achieve a specific network linespeed.
Quiet	Users that are thinking, delaying, or suspended, or are 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.

Where to Go For More Information

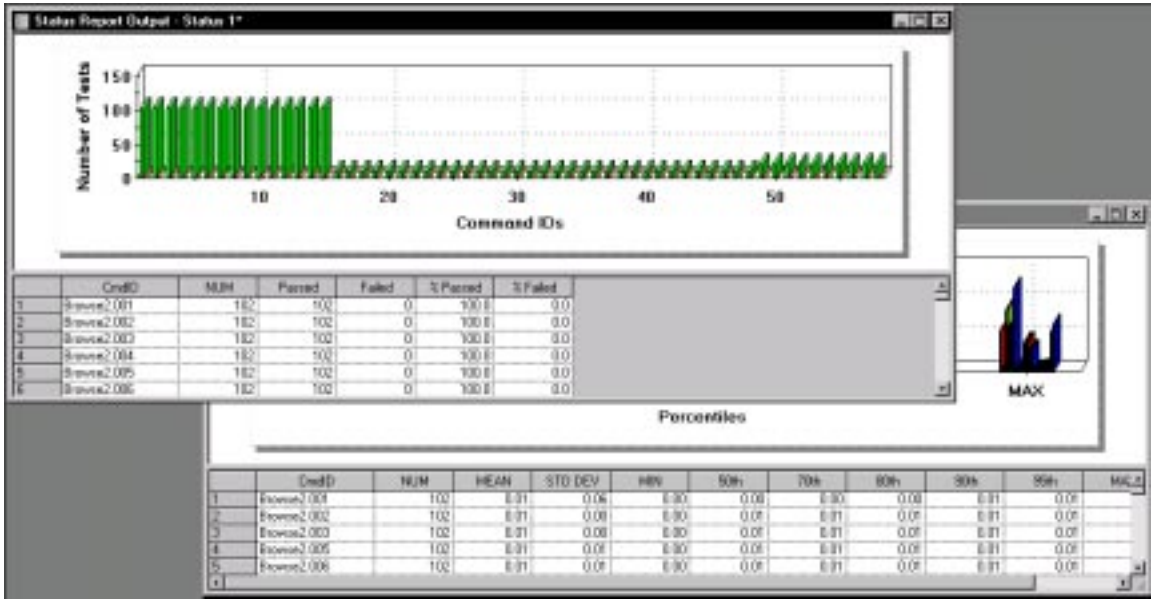
You can select a number of views when you monitor a schedule. See Chapter 8 of the *Using Rational LoadTest* manual for more information about:

- Viewing a schedule
- Types of histograms and views
- Zooming in on histograms for more detailed information
- Pausing or aborting a schedule run

Step 8: Running Reports

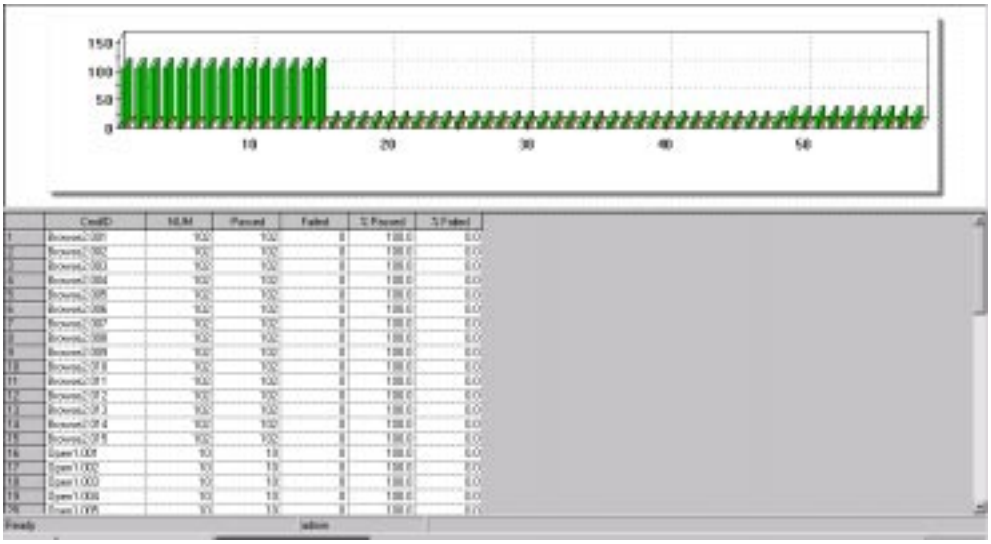
After a schedule run is complete, LoadTest displays the Status and the Performance report output.

NOTE: The LogViewer may appear, which indicates that at least one virtual user terminated abnormally. In a real test, you would investigate this failure. For this tutorial, however, dismiss the LogViewer, and from the LoadTest window, click the **Perf** and **Status** buttons to run the Performance and Status reports.



The Status report output shows which command IDs passed and which failed. To examine the output more closely:

1. Click the square in the upper-right corner of the Status report output to enlarge it.



In the Status report output, you'll see a bar chart and a table. The table lists the command IDs in alphabetical order.

Look at the Open command IDs, which came from the Open script. You will see that each Open command ran 10 times, which is as expected, because 10 virtual users opened Classics Online.

Look at the Browse command IDs. Each Browse command ran 102 times. This happened because of the way you designed the schedule and the number of users you ran. That is:

- Four Regular Buyers executed the Browse command three times (4 x 3 = 12).
- Five Window Shoppers ran a selector that executed the Browse command 15 times (5 x 15 = 75).
- One Big Spender ran a selector that executed the Browse command 15 times.

You'll notice the Status report output doesn't list any command IDs from the Close script, even though you recorded a Close script. This is because in this script, you closed the connection, no server traffic was generated, and therefore, there are no emulation commands to report on.

2. Click **File** → **Save As** to save the Status report output.

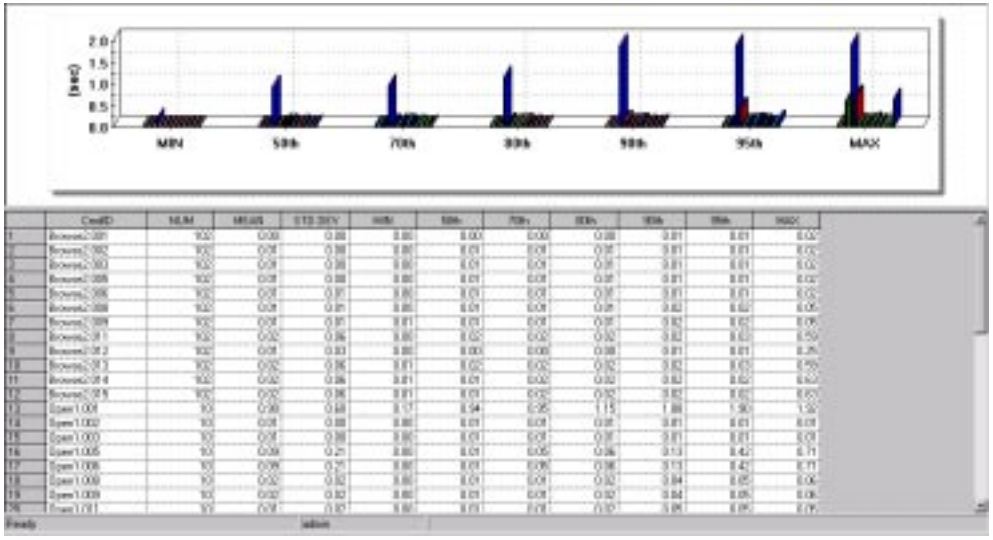
3. In the Save As dialog box, save the Status report with the name **Status--10 users**, and then click **OK**.

The Performance report output displays the response times that occurred during the schedule run, grouped by command ID.

To examine the output more closely:

1. Click the square in the upper-right corner of the Performance report output to enlarge it.

> **NOTE:** Your Performance report output will not be the same as that shown in the tutorial. This is because the performance characteristics of your environment differs from those used in the tutorial. Use the Performance report information in this tutorial as a guideline only.



2. Recall that when you planned your test, you set the pass and fail criteria as follows:

   When 10 users access your Web site, 90% of them must have a response time of 2 seconds or less.

   When 20 users access your Web site, 80% of them must have a response time of 6 seconds or less.

   No response time can exceed 8 seconds.

Now that you have run schedule with 10 users, you can check whether you have met some of your goals.

- Scroll down the 90% column, and check whether any response time is greater than 2 seconds. If none is greater, Classics Online has met this goal.
- Scroll down the MAX column to see whether any maximum response time exceeds 8 seconds. If none exceeds 8 seconds, Classics Online has met this goal for 10 users.

3. Click **File** → **Save As** to save the Performance report output.
4. In the Save As dialog box, save the output with the name **Performance--10 users**, and then click **OK**.

Running and Monitoring a 20-User Schedule

Now that you have examined these reports, you'll run the schedule with 20 users, and compare the difference in your server's response time.

Running a schedule with 20 users is almost identical to running one with 10 users. The only difference is that you'll enter **20** instead of 10 in **Number of Users**. And, when you run the schedule, the run should take longer, because it contains more users.

To run a schedule with 20 users:

1. Click **Run** → **Schedule**.
2. In the **Number of Users** box, enter **20**, and then click **OK**. This runs the schedule with 20 virtual users —10 Window Shoppers, 8 Regular Buyers, and 2 Big Spenders.
3. Monitor the schedule, as you did with 10 users.
4. At the end of the schedule run, LoadTest again displays Status and Performance report output.
5. Optionally, examine the Status report output.
6. Examine the Performance report output.
   - Scroll down the 90% column, and check whether any response time is greater than 2 seconds. If none is greater, Classics Online has met this goal.
   - Scroll down the MAX column to see whether any maximum response time exceeds 8 seconds. If none exceeds 8 seconds, Classics Online has met this goal.
7. Then click **File** → **Save As** to save the Status report output.

8. In the Save As dialog box, save the Status report output with the name **Status--20 users**, and then click **OK**.
9. Click **File** → **Save As** to save the Performance report output.
10. In the Save As dialog box, save the Performance report output with the name **Performance--20 users**, and then click **OK**.

Where to Go For More Information

The previous sections showed you how to run Performance reports to compare two schedule runs. These are not the only types of reports you can run, however. LoadTest provides additional reports that enable you to:

- Run one report to compare the results of up to six schedule runs.
- Filter command IDs — both before and after running a report.
- Eliminate renegade response times — called **outliers** — from the report output.
- Report on a few commands — or even on one command ID.
- Report on a few user groups — or even on one user.
- Customize reports and save your customizations.

For more information about LoadTest reports and analyzing results, see Chapter 9 of the *Using Rational LoadTest* manual.

Wrapping Up

Now that you've completed the performance testing tutorial, it's time to think about what you've learned and how you can apply this knowledge to your testing environment.

- In Step 1, you learned about planning performance tests. You considered the tasks of the application in terms of transactions, the people who perform those transactions, and the frequency and volume of those transactions. How do these concepts apply to your server?
- In Step 2, you learned about setting recording options, and saw the kinds of options that you could set for the different recording methods. How will you set the recording options for your session?
- In Step 3, you learned about recording a session. How will you apply this to your application? What are the logical ways that you can break up the user's tasks to modularize testing later? Will adding blocks help you test specific tasks?

- In Step 4, you learned about examining the script. What features will you add to your recorded scripts?
- In Step 5, you learned about generating realistic data to send to the server. What other ways can you use datapools to further refine your testing?
- In Step 6, you created a schedule in Rational LoadTest based on the session that you recorded. You learned about user groups, selectors, and scripts, and how to model user behavior by manipulating these items. How can you use user groups, selectors, and scripts to emulate user actions? Will your schedule be more complex and require additional items?
- In Step 7, you ran the schedule that you created and saw how LoadTest plays back the user actions and allows you to monitor the test as is it happening. You also saw how the test itself uses system resources. How will this affect how you choose to run a test? Do your servers have enough resources to run large numbers of users in the test schedule that you designed?
- In Step 8, you looked at the default reports created by LoadTest to analyze application and system performance. What reports will be most useful in your test environment? What kind of information do you most need to report on?

To help you in your testing, Rational Software provides several ways to learn more about this product and performance testing. For example:

- **Documentation** – Rational provides comprehensive documentation for all its products. You can find this documentation on the *Rational Solutions for Windows* Online Documentation CD.
- **Training –** Rational offers training for all of its products — including PerformanceStudio — through Rational University. For more information, see the Rational University Web site: http://www.rational.com/university.

  Rational can also provide custom on-site training tailored to your situation. Contact your Rational account manager to request this type of training.
- **Support Services –** Rational provides a full range of technical support services, from product maintenance to mentoring and consulting. For more information, see Rational's Web site: www.rational.com/services. In addition, you can contact your account manager for more information.

Good luck!

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, CPUs, 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

H

I

L

M

N

P

R

S

T

U

V

W