

# Using Rational TestManager

VERSION 2001.03.00

PART NUMBER 800-023807-000

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

**Rational**<sup>®</sup>  
the **e-development** company™

## **IMPORTANT NOTICE**

### **COPYRIGHT**

Copyright © 2000, Rational Software Corporation. All rights reserved.

Part Number: 800-023807-000

### **PERMITTED USAGE**

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION WHICH IS THE PROPERTY OF RATIONAL SOFTWARE CORPORATION ("RATIONAL") AND IS FURNISHED FOR THE SOLE PURPOSE OF THE OPERATION AND THE MAINTENANCE OF PRODUCTS OF RATIONAL. NO PART OF THIS PUBLICATION IS TO BE USED FOR ANY OTHER PURPOSE, AND IS NOT TO BE REPRODUCED, COPIED, ADAPTED, DISCLOSED, DISTRIBUTED, TRANSMITTED, STORED IN A RETRIEVAL SYSTEM OR TRANSLATED INTO ANY HUMAN OR COMPUTER LANGUAGE, IN ANY FORM, BY ANY MEANS, IN WHOLE OR IN PART, WITHOUT THE PRIOR EXPRESS WRITTEN CONSENT OF RATIONAL.

### **TRADEMARKS**

Rational, Rational Software Corporation, the Rational logo, Rational the e-development company, ClearCase, ClearQuest, Object Testing, Object-Oriented Recording, Objectory, PerformanceStudio, PureCoverage, PureDDTS, PureLink, Purify, Purify'd, Quantify, Rational Apex, Rational CRC, Rational PerformanceArchitect, Rational Rose, Rational Suite, Rational Summit, Rational Unified Process, Rational Visual Test, Requisite, RequisitePro, SiteCheck, SoDA, TestFactory, TestMate, TestStudio, and The Rational Watch are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

Microsoft, the Microsoft logo, the Microsoft Internet Explorer logo, DeveloperStudio, Visual C++, Visual Basic, Windows, the Windows CE logo, the Windows logo, Windows NT, the Windows Start logo, and XENIX are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

FLEXlm and GLOBEtrotter are trademarks or registered trademarks of GLOBEtrotter Software, Inc. Licensee shall not incorporate any GLOBEtrotter software (FLEXlm libraries and utilities) into any product or application the primary purpose of which is software license management.

### **PATENT**

U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,835,701. Additional patents pending.

Purify is licensed under Sun Microsystems, Inc., U.S. Patent No. 5,404,499.

### **GOVERNMENT RIGHTS LEGEND**

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational Software Corporation license agreement and as provided in DFARS 277.7202-1(a) and 277.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct. 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 227-14, as applicable.

### **WARRANTY DISCLAIMER**

This document and its associated software may be used as stated in the underlying license agreement. Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability or fitness for a particular purpose or arising from a course of dealing, usage, or trade practice.

# Contents

- Preface . . . . . xvii**
- Audience. . . . . xvii
- Other Resources . . . . . xvii
- Contacting Rational Technical Publications . . . . . xvii
- Contacting Rational Technical Support . . . . . xviii
  
- Part 1: Using TestManager to Manage Testing Projects**
  
- 1 Introducing Rational TestManager. . . . . 1**
- What Is Rational TestManager . . . . . 1
- TestManager Workflow . . . . . 2
  - Planning Tests . . . . . 3
  - Designing Tests . . . . . 5
  - Implementing Tests. . . . . 6
  - Executing Tests. . . . . 7
  - Evaluating Tests . . . . . 7
- TestManager and Other Rational Products . . . . . 8
  - The Rational Unified Process . . . . . 8
  - Projects and the Rational Administrator . . . . . 8
  - Test Scripts and Rational Robot . . . . . 9
  - Component Testing and Rational QualityArchitect . . . . . 9
  - Requirements and Rational RequisitePro . . . . . 10
  - Model Elements and Rational Rose . . . . . 10
  - Defects and Rational ClearQuest . . . . . 10
  - Reports and Rational SoDA . . . . . 11
- TestManager and Extensibility . . . . . 11
  - Defining Extensible Test Input Types . . . . . 11
  - Defining Extensible Test Script Types. . . . . 12
- Virtual Testers and Types of Tests . . . . . 12
  - About Virtual Testers . . . . . 12
  - Functional and Performance Testing . . . . . 13

Test Script Services .....	15
Starting TestManager .....	17
Logging into TestManager .....	17
Starting Other Rational Products and Components from TestManager .....	18
The TestManager Main Window .....	19
Test Asset Workspace .....	19
Other TestManager Windows .....	22
<b>2 Planning Tests .....</b>	<b>25</b>
About Test Planning .....	25
Defining What to Test by Using Test Inputs .....	26
Built-in Test Input Types .....	26
Custom Test Input Types .....	28
Creating a Test Plan .....	29
Editing and Creating Test Plans .....	29
Properties of a Test Plan .....	30
Customizing the Properties of a Test Plan .....	31
Organizing Test Cases with Folders .....	31
Creating Test Cases .....	33
Properties of a Test Case .....	33
Specifying the Owner .....	34
Defining the Configurations to Test .....	35
Specifying When to Run Tests .....	40
Setting up Traceability Using Test Inputs .....	42
<b>3 Designing Tests .....</b>	<b>45</b>
About Designing Tests .....	45
Specifying the Testing Steps and Verification Points .....	46
Specifying Conditions and Acceptance Criteria of Test Cases .....	48
Example of a Test Design .....	50
<b>4 Implementing Tests .....</b>	<b>53</b>
About Implementing Tests .....	53
Implementing Built-in Test Scripts Types and Suites .....	54
Implementing Extensible Test Script Types .....	55
Creating Manual Test Scripts .....	56
Starting Rational ManualTest .....	57
Example of a Manual Test Script .....	57

Setting the Default Editor for Manual Test Scripts . . . . .	58
Including an External File in a Manual Test Script . . . . .	59
Creating Script Queries . . . . .	59
Customizing Test Assets . . . . .	60
Associating an Implementation with a Test Case . . . . .	60
Implementing Tests as Suites . . . . .	62
Defining Computers and Computer Lists . . . . .	64
Opening a Suite . . . . .	67
Editing a Test Script . . . . .	67
Editing a Suite . . . . .	68
Setting Shared Variables . . . . .	86
Printing and Exporting a Suite . . . . .	87
Saving a Suite . . . . .	87
<b>5 Executing Tests . . . . .</b>	<b>89</b>
About Running Tests . . . . .	89
Built-in Support for Running Test Scripts . . . . .	89
Running Automated Test Scripts . . . . .	90
Running Manual Test Scripts . . . . .	91
Example of Running a Manual Test Script . . . . .	92
. . . . . Viewing the Results of Running a Manual Test Script	92
Running Test Cases . . . . .	92
Viewing the Associated Implementations. . . . .	92
Running a Test Case . . . . .	93
Running Suites . . . . .	94
Checking a Suite . . . . .	95
Checking Agent Computers . . . . .	95
Controlling Runtime Information of a Suite . . . . .	96
Controlling How a Suite Terminates. . . . .	99
Monitoring Suites . . . . .	102
About Monitoring Suites. . . . .	102
Displaying the Suite Views . . . . .	104
Displaying the Histogram Views. . . . .	107
Displaying the User/Computer Views. . . . .	114
Displaying the Shared Variables View . . . . .	119
Displaying the Script View . . . . .	120
Displaying the Sync Points View . . . . .	120

Displaying the Computer View . . . . .	122
Displaying the Transactor View. . . . .	125
Displaying the Group Views . . . . .	127
Filtering and Sorting Views. . . . .	127
Changing the Value of a Shared Variable. . . . .	130
Debugging a Test Script . . . . .	130
Changing Monitor Defaults . . . . .	131
Configuring Custom Histograms. . . . .	132
Controlling the Suite During a Run. . . . .	133
<b>6 Evaluating Tests . . . . .</b>	<b>137</b>
About Test Logs . . . . .	137
Opening a Test Log in TestManager. . . . .	138
The Test Log Main Window . . . . .	138
About Log Filters . . . . .	141
Viewing Test Log Results . . . . .	143
Viewing Test Case Results . . . . .	143
Viewing Events Details . . . . .	143
Viewing a Test Script . . . . .	146
Working with Test Logs. . . . .	146
About Test Logs . . . . .	146
Entering and Modifying Defects . . . . .	148
Printing a Test Log . . . . .	150
Managing Log Event Property Types . . . . .	151
Viewing Test Script Results Recorded with Rational Robot. . . . .	152
Reporting Results . . . . .	154
About Reports . . . . .	154
Selecting Which Reports to Use. . . . .	156
Additional Reports . . . . .	158
Creating Reports . . . . .	158
Opening a Report. . . . .	161
Running Reports . . . . .	161
Print, Save, or Copy a Test Case Trend or Distribution Report. . . . .	163
Print, Export, or Zoom a Listing Report . . . . .	163
Print, Save, Copy, Delete, or Export a Performance Report. . . . .	164
Copying Reports to a New Project . . . . .	164
Creating a Query . . . . .	164

## Part 2: Functional Testing with Rational TestManager

<b>7</b>	<b>About Functional Tests</b> .....	<b>169</b>
	Planning Functional Tests .....	169
	Identifying Functional Testing Requirements .....	169
	Setting Pass and Fail Criteria for Functional Tests .....	169
	Distributed Functional Testing .....	170
	Distributing Tests Among Different Computers .....	170
	Running Tests on a Specific Computer .....	170
	Example of a Distributed Functional Test .....	171
	Recording Considerations for Functional Tests .....	171
<b>8</b>	<b>Creating Functional Testing Suites</b> .....	<b>173</b>
	About Suites .....	173
	Creating a Suite .....	174
	Inserting Computer Groups into a Suite .....	176
	Inserting Test Scripts into a Suite .....	177
	Preconditions .....	178
	Inserting Other Items into a Suite .....	179
	Inserting a Test Case into a Suite .....	179
	Inserting a Suite .....	181
	Inserting a Selector .....	182
	Advanced Functional Testing .....	184
	Inserting a Scenario .....	184
	Inserting a Delay .....	187
	Inserting a Synchronization Point .....	187
	Using Events and Dependencies to Coordinate Execution .....	191
	Executing Suites .....	193
<b>9</b>	<b>Using the Comparators</b> .....	<b>195</b>
	About the Four Comparators .....	195
	Starting a Comparator .....	196
	Using the Object Properties Comparator .....	196
	The Main Window .....	197
	The Objects Hierarchy and the Properties List .....	198
	Loading the Current Baseline .....	200

Locating and Comparing Differences . . . . .	200
Viewing Verification Point Properties . . . . .	201
Adding and Removing Properties . . . . .	201
Editing the Baseline File . . . . .	202
Saving the Baseline File . . . . .	202
Using the Text Comparator . . . . .	203
The Main Window . . . . .	203
The Text Window . . . . .	203
Locating and Comparing Differences . . . . .	204
Viewing Verification Point Properties . . . . .	204
Editing the Baseline File . . . . .	205
Saving the Baseline File . . . . .	205
Using the Grid Comparator . . . . .	205
The Main Window . . . . .	206
The Grid Window . . . . .	206
Differences List . . . . .	207
Setting Display Options . . . . .	207
Locating and Comparing Differences . . . . .	208
Viewing Verification Point Properties . . . . .	208
Using Keys to Compare Data Files . . . . .	209
Editing the Baseline File . . . . .	210
Saving the Baseline File . . . . .	210
Using the Image Comparator . . . . .	210
The Main Window . . . . .	211
Locating and Comparing Differences . . . . .	213
Changing How Differences are Determined . . . . .	214
Changing the Color of Masks, OCR Regions, or Differences . . . . .	214
Moving and Zooming An Image . . . . .	215
Viewing Image Properties . . . . .	215
Working with Masks . . . . .	216
Working with OCR Regions . . . . .	216
Saving the Baseline File . . . . .	217
Viewing Unexpected Active Window . . . . .	217



## Part 3: Performance Testing with Rational TestManager

<b>10 Planning Performance Tests</b> .....	<b>221</b>
About Performance Testing .....	221
Performance Testing Basics .....	222
Types of Tests .....	223
Local and Agent Computers .....	226
Suites .....	226
Rational TestManager and Performance Testing .....	227
Why Use TestManager for Performance Testing? .....	227
The TestManager Environment .....	228
Planning Performance Tests .....	229
Testing Response Times .....	230
Setting Pass and Fail Criteria for Performance Tests .....	230
Identifying Performance Testing Requirements .....	230
Designing a Realistic Workload .....	231
Implementing Performance Tests .....	232
Examples of Performance Tests .....	233
Number of Virtual Testers Supported Under Normal Conditions .....	233
Incrementally Increasing Virtual Testers .....	234
How a System Performs Under Stress Conditions .....	236
How Different System Configurations Affect Performance .....	237
Analyzing Performance Results .....	237
Comparing Results of Multiple Runs .....	238
Comparing Specific Requests and Responses .....	238
Determining the Cause of Performance Problems .....	239
<b>11 Creating Performance Testing Suites</b> .....	<b>243</b>
About Suites .....	243
Creating a Suite .....	245
Inserting User Groups into a Suite .....	246
Inserting Test Scripts into a Suite .....	249
Preconditions .....	250
Inserting Other Items into a Suite .....	251
Inserting a Test Case into a Suite .....	252
Inserting a Suite .....	253

Inserting a Scenario .....	255
Inserting a Selector .....	257
Inserting a Delay .....	263
Inserting a Transactor .....	265
Inserting a Synchronization Point .....	269
Using Events and Dependencies to Coordinate Execution .....	276
Executing Suites .....	278
<b>12 Working with Datapools .....</b>	<b>279</b>
What Is a Datapool? .....	280
Datapool Tools .....	280
Datapool Cursor .....	282
Datapool Limits .....	282
What Kinds of Problems Does a Datapool Solve? .....	282
Planning and Creating a Datapool .....	284
Data Types .....	286
Standard and User-Defined Data Types .....	287
Finding Out Which Data Types You Need .....	288
Creating User-Defined Data Types .....	288
Generating Unique Values from User-Defined Data Types .....	289
Generating Multi-Byte Characters .....	290
Managing Datapools .....	290
Creating a Datapool .....	290
Editing Datapool Column Definitions .....	298
Editing Datapool Values .....	299
Renaming or Copying a Datapool .....	299
Deleting a Datapool .....	300
Importing a Datapool .....	300
Exporting a Datapool .....	301
Managing User-Defined Data Types .....	302
Editing User-Defined Data Type Values .....	302
Editing User-Defined Data Type Definitions .....	302
Importing a User-Defined Data Type .....	303
Renaming or Copying a User-Defined Data Type .....	303
Deleting a User-Defined Data Type .....	304
Generating and Retrieving Unique Datapool Rows .....	304
What You Can Do to Guarantee Unique Row Retrieval .....	305

Creating a Datapool Outside Rational Test . . . . .	306
Datapool Structure . . . . .	307
Using Microsoft Excel to Create Datapool Data . . . . .	308
Matching Datapool Columns with Test Script Variables . . . . .	309
Maximum Number of Imported Columns . . . . .	309
Creating a Column of Values Outside Rational Test . . . . .	310
Step 1. Create the File . . . . .	310
Step 2. Assign the File's Values to the Datapool Column . . . . .	310
Generating Unique Values . . . . .	311
<b>13 Reporting Performance Testing Results . . . . .</b>	<b>313</b>
About Reports . . . . .	313
Running a Report . . . . .	316
Running a Report from the Report Bar . . . . .	316
Running a Report from the Menu Bar . . . . .	316
Customizing Reports . . . . .	317
Filtering Report Data . . . . .	317
Setting Advanced Options . . . . .	318
Changing a Graph's Appearance or Type . . . . .	324
Editing the Properties of a Report . . . . .	328
Managing Reports . . . . .	328
Printing a Report . . . . .	328
Copying a Report . . . . .	328
Renaming a Report . . . . .	329
Deleting a Report . . . . .	329
Exporting Reports . . . . .	330
Changing Report Defaults . . . . .	330
Changing the Reports that Run Automatically . . . . .	330
Changing the Reports that Run from the Report Bar . . . . .	331
Types of Reports . . . . .	331
Performance Reports . . . . .	332
About Percentiles in Performance Reports . . . . .	334
Compare Performance Reports . . . . .	335
Response vs. Time Reports . . . . .	341
Command Status Reports . . . . .	344
Command Usage Reports . . . . .	346

<b>A</b>	<b>Configuring Local and Agent Computers</b>	<b>355</b>
	Running More Than 245 Virtual Testers	355
	Running More Than 1000 Virtual Testers	355
	Running More Than 1000 Virtual Testers on One NT Computer	356
	Running More Than 24 Virtual Testers on a UNIX Agent	357
	Controlling TCP Port Numbers	358
	Setting Up IP Aliasing	359
	Assigning Values to System Environment Variables	360
<b>B</b>	<b>Standard Datapool Data Types</b>	<b>365</b>
	Standard Data Type Table	365
	Data Type Ranges	372
	<b>Index</b>	<b>375</b>

# Preface

Rational TestManager is an open and extensible framework that unites all of the tools, artifacts, and data both related to and produced by the testing effort. Under this single umbrella, all stakeholders and participants in the testing effort can define and refine the quality goals they are working toward.

This manual describes how to use Rational TestManager to support the five testing activities defined in the Rational Unified Process, and how to use TestManager for functional testing and performance testing.

## Audience

---

This manual is intended for project analysts, project architects and developers, quality assurance team members, project managers, and any other stakeholders involved in the testing effort.

## Other Resources

---

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

**Note:** This manual contains conceptual information. For detailed procedures, see the TestManager Help.

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

## Contacting Rational Technical Publications

---

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

## Contacting Rational Technical Support

---

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

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

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

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

# **Part 1: Using TestManager to Manage Testing Projects**





# Introducing Rational TestManager

# 1

This chapter introduces you to Rational TestManager. It includes the following topics:

- What is Rational TestManager
- TestManager workflow
- TestManager and other Rational products
- TestManager and extensibility
- Virtual testers and types of tests
- Starting TestManager
- The TestManager main window

## What Is Rational TestManager

---

Testing is the feedback mechanism in the software development process. It tells us where corrections need to be made to stay on course at any given iteration of a development effort. It also tells us about the current quality of the system being developed.

Everyone involved in the project is a stakeholder in the process of defining how system quality will be assessed and in taking actions to correct problems. For example:

- Project analysts need to know about the availability, completeness, and quality of use cases, features, and requirements supported by the system.
- Project architects and developers need to understand the state of components and subsystems they have designed or developed.
- Quality assurance team members need to develop a plan to test the system so that the answers required by the rest of the project team can be provided. Further, they need to understand and define the relationships between the elements of their testing plan and the other artifacts of the development effort. These traceability

relationships allow the QA team to understand how changes elsewhere in the project affect their work, and to define how they will test the elements of the system to provide the required answers to the entire team.

- Project managers will use the information the testing effort provides to make decisions about the acceptability and readiness of the system for release. Their decisions will be based on input from other team members such as analysts and developers who will, at least in part, be drawing their knowledge of the state of the system from these same measurements.

What has been described here is no small effort. This is why testing efforts often represent 25–50% of the overall project effort. Making these tasks more difficult is the fact that collecting the required data, tracking the relationships amongst artifacts, and providing a common presentation of the output of the testing effort often involves use of several tools with disparate artifacts and data. This can make it nearly impossible to efficiently track the effects of dependencies and to get a concise, consistent view of the state of a system.

Rational TestManager is the open and extensible framework that unites all of the tools, artifacts, and data both related to and produced by the testing effort. Under this single umbrella, all stakeholders and participants in the testing effort can define and refine the quality goals they are working toward. It is where the team defines the plan it will implement to meet those goals. And, most importantly, it provides the entire team with one place to go to determine the state of the system at any given point in time as measured against any stakeholder's quality requirements.

By supporting all testing activities, quality assurance professionals use TestManager as the central point from which all of those activities are coordinated and from which the outputs of activities are managed and tracked. It tells testers what work needs to be done by whom and by what date. It also informs the testers what areas of their work are affected by changes happening elsewhere in the development effort. TestManager is the one place to go for the answers to all questions related to system quality.

## TestManager Workflow

---

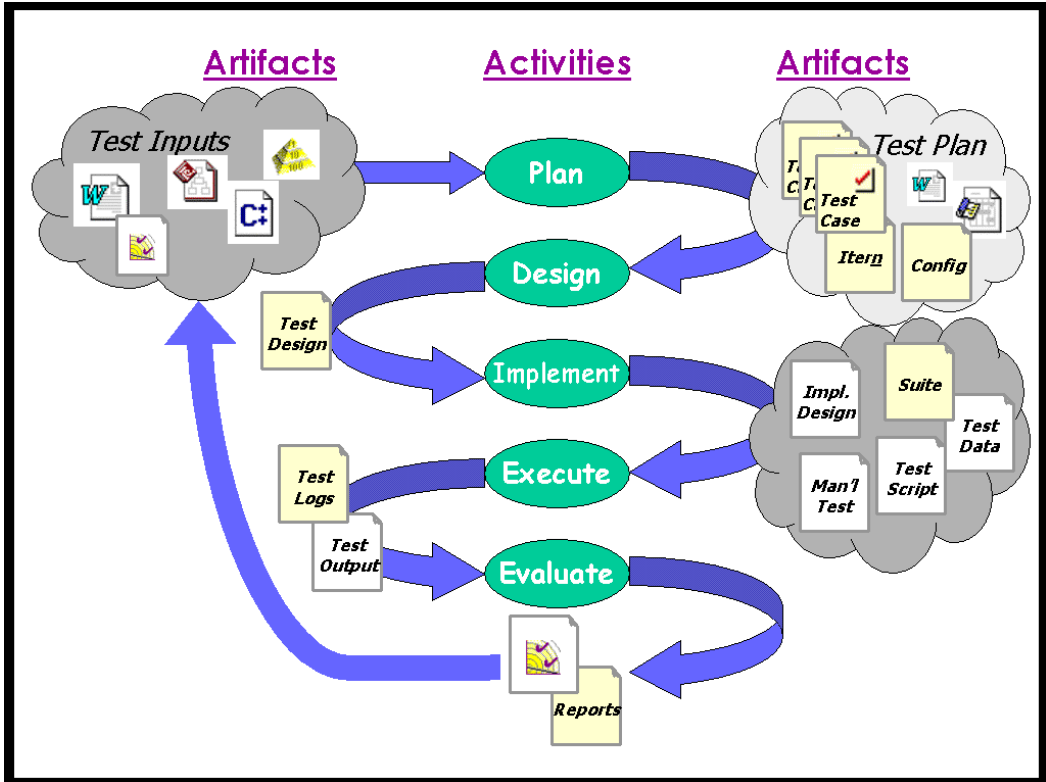
The TestManager workflow supports the five major testing activities defined by the Rational Unified Process:

- Planning tests
- Designing tests
- Implementing tests

- Executing tests
- Evaluating tests

Each of these activities has input and output artifacts, as shown in the following figure.

### Testing Workflow



### Planning Tests

The activity of planning tests primarily involves collecting and organizing information that answers the following questions about testing:

- What and Where? – Requirements, visual models, and other test inputs tell you what to test and where to run the tests.
- When? – Iteration plans tell you when the tests need to be run and to pass.

- Who? – Test plans, iteration plans, or project plans tell you who will perform the testing activities.

## Test Inputs

Test inputs help you answer the four questions about testing. The first step in planning your testing effort is to identify the test inputs. A *test input* is anything that the tests depend on or anything that needs validation. Test inputs help you decide what you need to test. They also help you determine what tests might need to change based on changes in the development process. This is extremely important in iterative development where change is a frequent, necessary part of the process.

TestManager has two built-in test input types:

- Requirements created in a Rational RequisitePro project
- Elements in a Rational Rose visual model

These built-in test input types give you easy access to requirements and model elements, and let you associate these inputs with other test artifacts for traceability purposes.

TestManager also supports test inputs that are not generated by Rational tools. To use any external test inputs within TestManager, you'll need to write test input adapters or use adapters provided by Rational's partners. For more information, see *Defining Extensible Test Input Types* on page 11.

## Test Plans

Once you have identified your test inputs, you can use TestManager to create a test plan. The *test plan* contains information about the purpose and goals of testing within the project, and the strategies to be used to implement and execute testing.

The test plan can contain a varied collection of information and addresses many issues including:

- What tests will need to be performed?
- When will the tests need to be performed and be expected to pass?
- Who is responsible for each test?
- Where will tests need to be performed? In other words, on what hardware and software configuration must they be run?

Projects can contain multiple test plans. You may have a plan for each phase of testing. Different groups may have their own plans. Generally, each plan should have a single high-level testing goal.

Each test plan can contain test case folders and test cases.

## Test Case Folders

Use *test case folders* to organize test cases. Within a test plan, you can create test case folders to organize your test cases hierarchically.

You can organize your test cases in many ways. Common organizations may reflect system architecture, major use cases, requirements, or combinations of these.

## Test Cases

Use *test cases* to validate that the system is working the way that it's supposed to work and is built with the quality you need before you can ship it. The test case is the artifact in TestManager that answers the question, "*What* do I need to test?" You develop test cases to validate particular behaviors.

Each test case is owned by or assigned to a team member. This answers the question, "*Who* will do the testing?"

## Iterations

Use *iterations* to specify *when* a test case must pass. An iteration is a defined span of time during a project. The end of an iteration is a milestone. An iteration says that at some point in time, the product has to meet a certain quality standard to reach a milestone. The quality standard is defined by the test cases that must pass.

## Configurations

Use *configurations* to specify *where* test cases must be run—on what hardware and software configurations. For example, if you need to make sure that your test case passes when run under four different operating systems, you could create a configuration for each operating system. Then you could associate those four configurations with the test case, to create configured test cases. In order for the main test case to pass, all of its configured test cases need to pass.

## Designing Tests

The activity of designing tests is primarily answering the question, "*How* am I going to do the testing?" When you complete your test designing, you end up with a design that helps you understand how you are going to perform the test case.

Designing tests is an iterative and ongoing process. You should be able to start before any system implementation by basing the test design on use case specifications, requirements, prototypes, and so on. As the system becomes more clearly specified, the design can become more detailed along with it.

In TestManager, you can design your test cases by indicating the actual steps required to interact with the application and the system in order to perform the test, and how to validate that the features are working properly. You also specify pre-conditions, post-conditions, and acceptance criteria for the test.

## Implementing Tests

The activity of test implementation primarily involves creating reusable test scripts that implement your test case. You can then associate the implementation with the test case.

Implementation is different in every testing project. In one project, you might decide to build both automated test scripts and manual test scripts. In an other project, you might need to write modular pieces of software using a combination of tools.

TestManager provides built-in support for implementing the following test script types:

Test Script Type	Description
GUI	A functional test script written in SQABasic, a Rational proprietary Basic-like scripting language. Created in Rational Robot. (Available only if Rational Robot is installed.)
VU	A performance test script written in VU, a Rational proprietary C-like scripting language. Created in Rational Robot. (Available only if Rational Robot is installed.) <b>Note:</b> When you choose to record a VU test script, you actually begin recording a session. You can choose to generate VU or VB test scripts from the recorded session, depending on selected recording options.
Manual	A set of testing instructions to be run by a human tester. Created in Rational ManualTest.
Command Line	An executable file, including arguments and initial directory, detailed when inserted into a suite or specified at test case implementation.

TestManager also supports implementation of other test script types that you have registered. See *Defining Extensible Test Script Types* on page 12.

## Executing Tests

The activity of executing your tests is primarily running the implementations of test cases to make sure that the system functions correctly. In TestManager, you can run any of the following: (1) an individual test script; (2) one or more test cases, which run the implementations of the test cases; (3) a suite, which runs any combination of test scripts and test cases (and their implementations) across multiple computers and users

TestManager provides built-in support for executing the following test script types:

Test Script Type	Description
GUI	A functional test script written in SQABasic, a Rational proprietary Basic-like scripting language.
VU	A performance test script written in VU, a Rational proprietary C-like scripting language.
Manual	A set of testing instructions to be run by a human tester.
Visual Basic	A test script written in the Visual Basic language.
Java	A test script written in the Java language.
Command line	A file (for example, an .exe file, a .bat file, or a UNIX shell script) including arguments and an initial directory that can be executed from the command line.

TestManager also supports execution of other test script types that you have registered. See *Defining Extensible Test Script Types* on page 12.

## Evaluating Tests

The activity of evaluating tests is necessary to:

- Determine the validity of the actual test run. Did it complete? Did it fail because pre-conditions weren't met?
- Analyze test output to determine the result. In performance testing, you look at reports on the generated data to see if performance is acceptable.
- Look at aggregate results to check coverage against plan, test inputs, configurations, and so on. This can also be used to measure test progress and to do trend analysis.

# TestManager and Other Rational Products

---

TestManager can be purchased standalone, or as part of other Rational packages. When installed with other Rational products, it is tightly integrated with those products.

## The Rational Unified Process

The Rational Unified Process is a software engineering process that enhances team productivity and delivers software best practices via guidelines, templates, and tool mentors for all critical activities.

To quickly view the areas of the Rational Unified Process that are directly related to testing:

- In TestManager, click **Help > Extended Help**.

To view the complete online version of the Rational Unified Process if you have installed Rational Suite:

- Click **Start > Programs > Rational Suite > Rational Unified Process**.

## Projects and the Rational Administrator

When you work with TestManager, the information you create is stored in projects. You use the Rational Administrator to create and manage Rational projects.

A *Rational project* stores software testing and development information. All Rational components on your computer update and retrieve data from the same project.

**Note:** The types of data in a Rational project depend on the Rational software that you have installed.

A Rational project can contain several different datastores and databases:

- A *database* contains information that is related to only one purpose or subject.
- A *datastore* can contain one or more databases, as well as other types of files such as schemas. Thus a database is one element of a datastore.

A Rational project can consist of the following types of datastores:

- Rational Test datastore – Stores application testing information such as test assets, logs, reports, and builds.
- Rational RequisitePro datastore – Stores product or system requirements, software and hardware requirements, user requirements, quality assurance procedures, and test plans. Each datastore consists of project documents and a dynamically linked database.



- Rational ClearQuest database – Stores change-request information for software development, including enhancement requests, defect reports, and documentation modifications. Each ClearQuest database consists of one schema and one user database.
- Rational Rose models – Stores visual models for business processes, software components, classes and objects, and distribution and deployment processes.

## Test Scripts and Rational Robot

Rational Robot lets you develop automated test scripts for functional testing and performance testing. Use Robot to:

- Perform full functional testing. Record test scripts that navigate through your application and test the state of objects through verification points.
- Perform full performance testing. Record test scripts that help you determine whether a multi-client system is performing within user-defined response-time standards under varying loads.
- Test applications developed with IDEs such as Java, HTML, Visual Basic, Oracle Forms, Delphi, and PowerBuilder. You can test objects even if they are not visible in the application's interface.
- Collect diagnostic information about an application during test script playback. Robot is integrated with Rational Purify®, Rational Quantify™, and Rational PureCoverage™. You can play back test scripts under a diagnostic tool and see the results in the log.

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

## Component Testing and Rational QualityArchitect

Rational QualityArchitect is a powerful collection of integrated tools for testing middleware components built with technologies such as Enterprise JavaBeans and COM.

QualityArchitect lets you reverse-engineer a component into Rational Rose, generate test scripts, and finally edit and run the test scripts right from your Java development environment or from Rational TestManager.

With QualityArchitect, you can:

- Generate test scripts that unit-test individual methods or functions in a component-under-test.

- Generate test scripts that drive the business logic in a set of integrated components. Scripts can be generated directly from Rose interaction diagrams or from live components using the Session Recorder.
- Generate stubs that allow you to test components in isolation, apart from other components called by the component-under-test.
- Track code coverage through Rational PureCoverage and model-level coverage through Rational TestManager.

## Requirements and Rational RequisitePro

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

When you create a project using the Rational Administrator, you can associate a RequisitePro project with the Administrator project. You can then use the requirements in the RequisitePro project as test inputs to your test plan in TestManager, and you can easily associate the requirements with test cases.

## Model Elements and Rational Rose

Rational Rose helps you visualize, specify, construct, and document the structure and behavior of your system's architecture. With Rose, you can provide a visual overview of the system using the Unified Modeling Language (UML), the industry-standard language for visualizing and documenting software systems.

When you create a project using the Rational Administrator, you can associate Rose models with the project. You can then use the elements in that model as test inputs to your test plan in TestManager, and you can easily associate the elements with test cases.

## Defects and Rational ClearQuest

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

With TestManager and ClearQuest, you can submit defects directly from a test log. TestManager automatically fills in some of the fields in the ClearQuest defect form with information from the test log.

## Reports and Rational SoDA

Rational SoDA generates up-to-date project reports of data extracted from one or more tools in Rational Suite. SoDA can work with one Rational tool, such as RequisitePro, or combine information from more than one tool, such as Rational RequisitePro, Rational Rose, Rational TestManager, and ClearQuest. These reports provide a way for your team to communicate more efficiently and consistently.

For example, with SoDa you can create a report with the following information on a software development project:

- Software requirements from RequisitePro.
- Software models from Rose.
- Testing criteria from TestManager.
- Defect tracking information from ClearQuest.

## TestManager and Extensibility

---

By exporting various application programming interfaces (APIs), TestManager is not limited to supporting test artifacts generated by Rational tools. The APIs provide hooks into the TestManager software, enabling implementers to plug in functionality that suits specific testing purposes. In addition to the pre-defined test inputs (RequisitePro requirements and Rose models), TestManager supports test inputs defined by the customer. The other type of test artifact that can be extended to include types defined by the customer are test scripts.

### Defining Extensible Test Input Types

Any kind of intermediary object needed for testing can be defined and managed as a test input type—for example, objects like Microsoft Project files or Excel spreadsheets. You could also define C++ -language project files as a test input type. These files might contain project-specific language restrictions and metrics, such as permitted depth of inheritance. The test designer might create test cases that determine whether specific source modules adhere to these restrictions.

For TestManager to support a new test input type, there must be a user-implemented dynamic link library called a Test Input Adapter (TIA). The adapter includes functions for tasks such as connecting to and disconnecting from the test input source, testing whether an input element has been modified, and setting various kinds of filters. You cannot make modifications to test input elements through TestManager. Changes to test input elements must be made through the application native to the test input type.

## Defining Extensible Test Script Types

There are two ways of defining your own test script type:

- Using the built-in command-line adapter. This adapter is for file-based test scripts that have operations driven from a command line—for example, a PERL script. A new PERL script can be created and later edited by using a text editor. You can define the command for the creation and edit operations for test scripts of a particular test type, as—for example—`Notepad test`, where *test* is replaced with the name of the test script before the command line is executed.

Defining a test script type in this way requires no additional programming, but only entering some information about the type through the TestManager user interface. From the TestManager Tools menu you can select **Manage > Test Script Types**. In the Console Adapter Type tab, select **Use the Command-Line Console Adapter**. You can enter the names of the executable commands for creating and editing a test script. Selecting test scripts is done through the standard File Open dialog box.

- Using a customer-created adapter. You use this kind of adapter for file-based test scripts when you want to take advantage of Rational's Test Script Services or Verification Point services, or both. Also, for test script types that are not file based—for example, Rational Visual Test scripts, you must implement a DLL that provides a minimal user interface and enables TestManager to connect to and disconnect from the test input source, in addition to creating, editing, and selecting test scripts of the custom type. Rational provides an applications programming interface (API) for implementing this DLL, which is called a Test Script Console Adapter (TSCA). You can learn more about custom adapters in the *Rational TestManager Extensibility Reference* manual.

## Virtual Testers and Types of Tests

---

### About Virtual Testers

A *virtual tester* is a single instance of a test script running on a computer. For functional test, only one virtual tester at a time can run on a computer. For performance tests, many virtual testers can run on a computer simultaneously.

A virtual tester running a performance test emulates client/server requests sent to a server. For example, when you record a session in Robot, Robot records a client's requests—such as Oracle, Microsoft SQL Server, and HTTP requests—to the server.

Robot also records the server's responses. This network traffic is the only activity that Robot records. Robot ignores GUI actions such as keystrokes and mouse clicks. After recording the session, Robot generates an appropriate test script.

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

## Functional and Performance Testing

When you plan tests, you might need to think about whether you are interested in performance testing, functional testing, or both types of testing.

### About Functional Testing

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

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

For information about functional testing with TestManager, see Part 2.

### About Performance Testing

In performance testing, you can measure the following things:

- The client response time. This is the total end-to-end response time as seen by a user. It is the time it takes for a virtual tester to enter a request, the server to respond to the request, and the virtual tester to see the results.
- The server response time. This is the time it takes for the server to process a request.

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

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

For information about performance testing with TestManager, see Part 3.

## Differences Between Functional and Performance Tests

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

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

## Local and Agent Computers

You coordinate the activities of all your test scripts from a single Windows computer where TestManager is running. This is known as the *Local computer*. From the Local computer, you create, run, and monitor suites.

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

- Adding workload to the server. If you are running a test with a large number of virtual testers, you can use Agent computers to add load to the server.
- Running test scripts on more than one computer. If you are running a functional test, you can save time by running the test scripts on the next available Agent computer instead of having the Local computer run all the test scripts. For this situation, the test scripts must be modular and independent.

- Running functional tests with many virtual testers. If you are running a functional suite with more than one virtual tester, you need an Agent computer, because only one virtual tester can run on each computer.
- Testing hardware configurations. If you are testing different hardware configurations, you can run test scripts on different Agent computers that are set up with these hardware configurations.

## Suites

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

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

## Test Script Services

Rational Test Script Services (TSS) are testing services that you can add to your test scripts using the calls in the Test Scripts Services API. For example, you can add logging, synchronization, timing, and datapool calls to test scripts. You can also add verification calls to validate the state of UI, COM/DCOM, and EJB objects.

The following table lists the categories of services that the Test Script Services provides:

Category	Description
Datapool	Provide variable data to test scripts during playback, allowing virtual testers to send different data to the server with each transaction.
Logging	Log messages for reporting and analysis.
Measurement	Provide the means of fine-tuning and controlling your tests through operations such as timing actions, setting think time delays, and setting environment variables.
Utility	Perform common test script operations such as retrieving error information, controlling the generation of random numbers, and printing messages.
Monitor	Monitor test script playback progress.
Synchronization	Synchronize multiple virtual testers running on a single computer or across multiple computers.

Category	Description
Session	Manage test script session execution and playback.
Advanced	Advanced features, such as setting values for internal variables.
Verification Point	Validate the state of UI, COM/DCOM, and EJB objects.

## Test Script Services and Test Script Types

All test script types—including custom test script types that you might add to TestManager—support Test Script Services.

With all automated test script types, you can add Test Script Services commands to test scripts manually during test script editing. Further, with some test script types, some or all Test Script Services commands can be added to test scripts automatically during the following operations:

- Recording with Rational Robot. Robot lets you record GUI, VU, and Visual Basic test scripts.
- Recording with the Rational QualityArchitect Scenario Recorder. The Scenario Recorder lets you record Java and Visual Basic test scripts.
- Rational QualityArchitect test script generation from a Rational Rose sequence diagram. The test script generator automatically generates Java and Visual Basic test scripts.

Use the following table as a guideline for including Test Script Services in different kinds of test scripts. For details, see the documentation listed for each test type:

Test Script Type	Method of Adding Test Script Services Commands	Documentation
GUI	Recording with Rational Robot or manually editing.	<i>SQABasic Language Reference</i>
VU	Recording a session and generating the test script with Rational Robot or manually editing.	<i>VU Language Reference</i>
Visual Basic	Recording a session and generating the test script with Rational Robot or manually editing.	<i>Rational Test Script Services for Visual Basic</i>
Java	Manual editing.	<i>Rational Test Script Services for Java</i>
Command Line	Manual editing.	<i>Rational Test Script Services for Shell Scripts</i>



Custom	Manual editing.	<i>Rational TestManager Extensibility Reference</i>
--------	-----------------	---

## Test Script Services and TestManager

Test Script Services are designed for use with TestManager. As a result, Test Script Services features that are included in any type of test script—including test scripts of custom test type(s)—can be tracked and managed by TestManager. For example:

- TestManager will adhere to any synchronization and delay functionality in your test script when it plays back (executes) the test script within a suite of test scripts.
- During test script playback, a tester can monitor various status information about your test script through the test script monitoring commands.
- During script playback, TestManager can provide realistic and verifiable data to the test scripts through use of datapools.
- The results of timed actions are displayed in TestManager reports.
- TestManager test cases can be associated with test scripts that contain verification commands for validating UI, COM/DCOM, or EJB objects.
- TestManager can run test scripts of multiple types within a single suite. For example, SQABasic, VU, Visual Basic, Java, Shell Command, and test scripts of custom types can all be run within the same suite.

## Starting TestManager

---

Before you start using TestManager, you need to have:

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

## Logging into TestManager

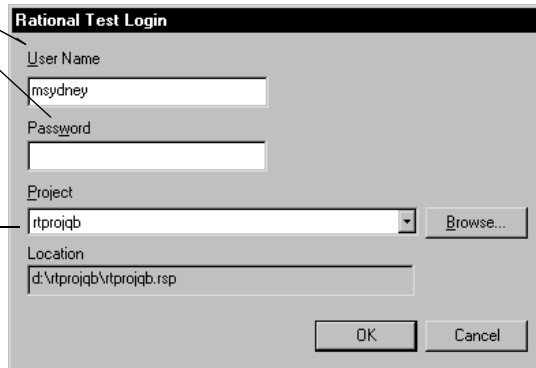
When you log into TestManager, you provide your user ID and password, which are assigned by your administrator. You also specify the project to log into.

To log in:

- Click **Start > Programs > Rational product name > Rational TestManager** to open the Rational Test Login dialog box.

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

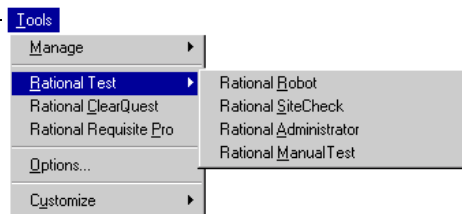
Select a project. To change projects after you log in, exit TestManager and log in again. (Projects are created in the Rational Administrator.)



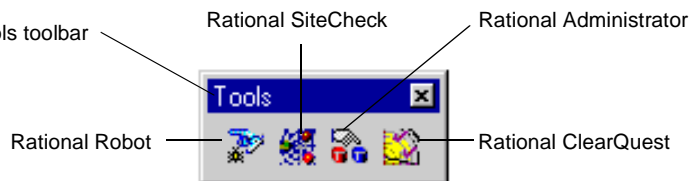
## Starting Other Rational Products and Components from TestManager

Once you are logged into TestManager, you can start other Rational products and components from either the Tools menu or the Tools toolbar.

The Tools menu

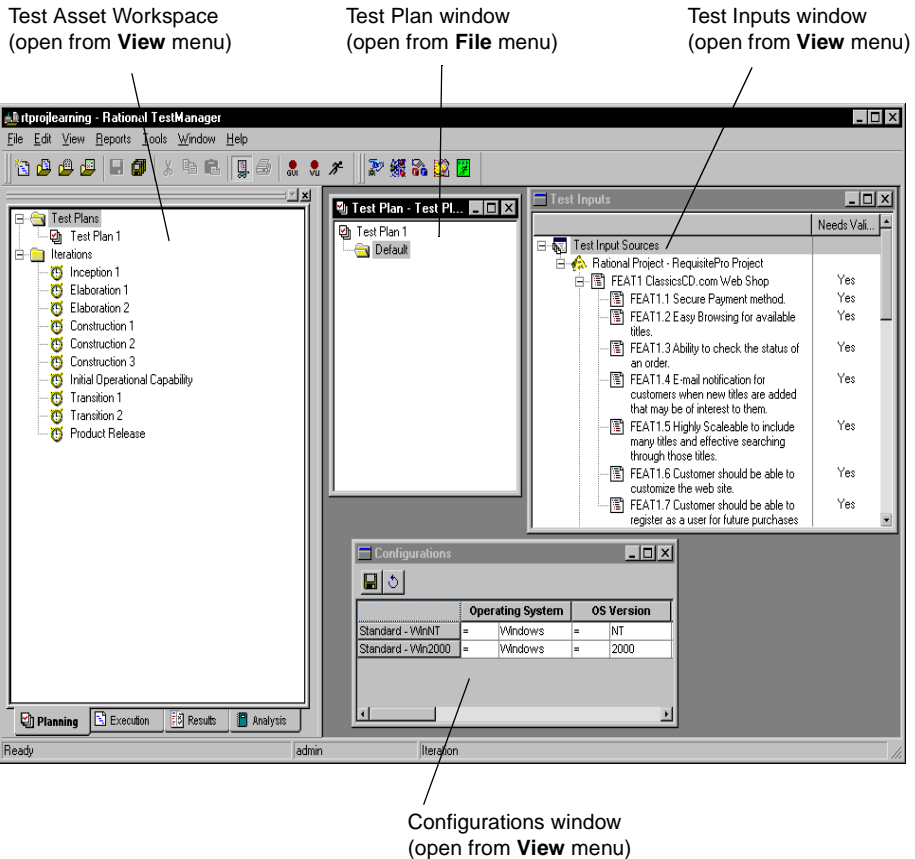


The Tools toolbar



# The TestManager Main Window

The following figure shows the TestManager main window and some of its child windows.



## Test Asset Workspace

The Test Asset Workspace gives you different views of the test assets in your project. It has four tabs:

- Planning
- Execution
- Results
- Analysis

To show or hide the Test Asset Workspace:

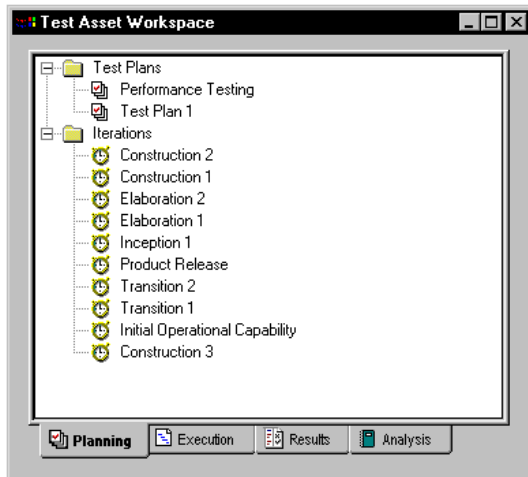
- Click **View > Test Asset Workspace**.

Right-click any test asset in the Workspace to display a shortcut menu.

Right-click near the bottom of the window (in an empty area) to allow docking of the Workspace or to float it in the main window.

## Planning Tab

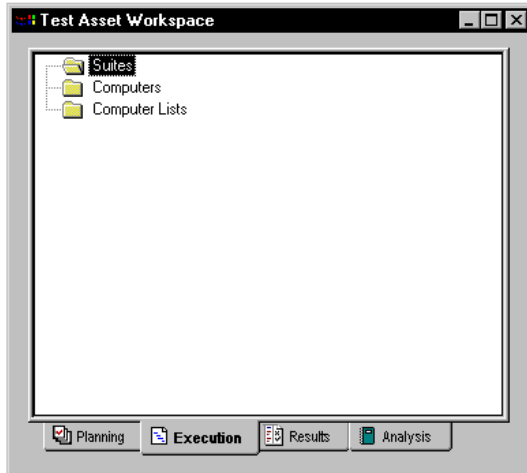
The **Planning** tab lists the test plans and iterations in the project.



For information about test plans and iterations, see *Planning Tests* on page 25.

## Execution Tab

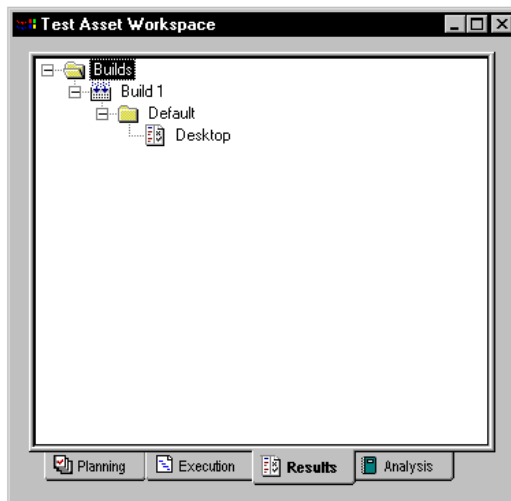
The **Execution** tab lists the suites, computers, and computer lists in the project.



For information about suites, computers, and computer lists, see *Implementing Tests* on page 53 and *Evaluating Tests* on page 137.

## Results Tab

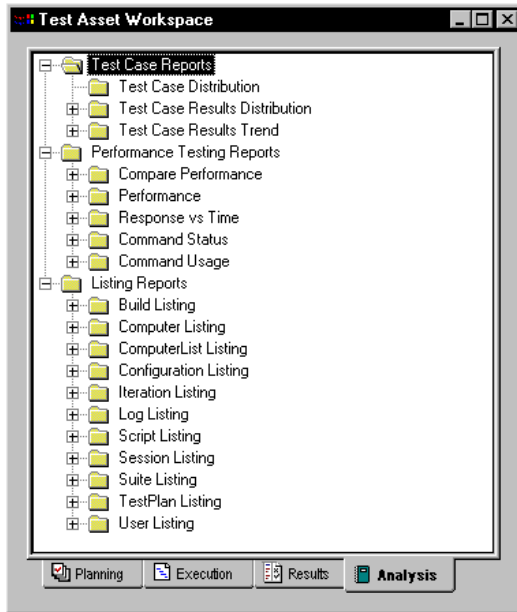
The **Results** tab lists the builds, test log folders, and test logs in the project.



For information about builds, test log folders, and test logs, see *Executing Tests* on page 89 and *Evaluating Tests* on page 137.

## Analysis Tab

The **Analysis** tab lists the reports in the project.



For information about reports, see *Evaluating Tests* on page 137 and *Reporting Performance Testing Results* on page 313.

## Other TestManager Windows

The following table lists other TestManager windows and where to find more information about them.

Window	Description	See
Test Input	Shows the test inputs associated with the project.	<i>Defining What to Test by Using Test Inputs</i> on page 26
Test Plan	Shows a test plan and all of its test case folders and test cases.	<i>Creating a Test Plan</i> on page 29

<b>Window</b>	<b>Description</b>	<b>See</b>
Configuration	Shows all of the configurations and configuration attributes in the project.	<i>Defining the Configurations to Test</i> on page 35

Each window opens within the TestManager main window as an MDI (Multiple Document Interface) child, which can be moved around inside of the main window. However, you can choose to make each window a floating (undocked) or anchored window (docked at the top, left, bottom, or right in the TestManager window). Right-click the title bar of the window and select a command on the shortcut menu.





This chapter describes how to plan tests. It includes the following topics:

- About test planning
- Defining what to test by using test inputs
- Creating a test plan
- Organizing test cases with folders
- Creating test cases

**Note:** For detailed procedures, see the TestManager Help.

## About Test Planning

---

The activity of test planning answers the question, “What do I have to test to meet the agreed-upon quality objectives?” When you complete your test planning, you have a test plan that defines what you are going to test.

The test planning happens over time. You continually add things to the test plan. You’ll come up with new test cases that you have to define, new situations that you need to test, and new features that you are just learning about. In other words, a test plan is not something that you just create at the beginning of the process and view as a stagnant object. A test plan is an evolving artifact that is defined iteratively.

In TestManager, a test plan can contain lists of test cases. The test cases can be organized hierarchically in test case folders.

In TestManager, test planning consists of the following major tasks:

- Gathering and identifying the test inputs.
- Creating the test plan.
- Creating the test case folders.
- Creating the test cases.
- Defining the configurations you need to test against.
- Defining the iterations—when you need to run the tests.

## Defining What to Test by Using Test Inputs

---

When you first start your test planning, your goal is to build a checklist of all of the things that need to be tested.

One way to start planning is to look at any available source materials that will help you determine what you need to test. For example, you can look at:

- Builds of the software
- Functional specifications
- Requirements
- Models
- Source code
- Change requests

All of these materials are something that you as a tester might look at to try to help you decide, “What do I have to test?” These materials are your *test inputs*. They are inputs to the planning phase. They help you build the checklist of the things you need to test.

Once you build this checklist, you can create test cases. You can then associate the test cases with the test inputs for tracking purposes. By setting up these associations, you can more easily track changes to the test inputs that might result in changes to the test cases or their implementations. For information, see *Setting up Traceability Using Test Inputs* on page 42.

You can also run reports to identify the test inputs that have test cases and implementations associated with them, and which of those test cases have been run. For example, analysts might be interested in the test coverage based on requirements. Architects might be interested in the test coverage based on model elements. For information about reports, see *Reporting Results* on page 154.

Almost anything can be a test input. TestManager provides built-in test input types, and you can also define custom test input types as your testing environment requires.

To view the available test inputs:

- Click **View > Test Inputs** to open the Test Input window.

### Built-in Test Input Types

TestManager comes with two built-in test input types:

- Requirements created in a Rational RequisitePro project

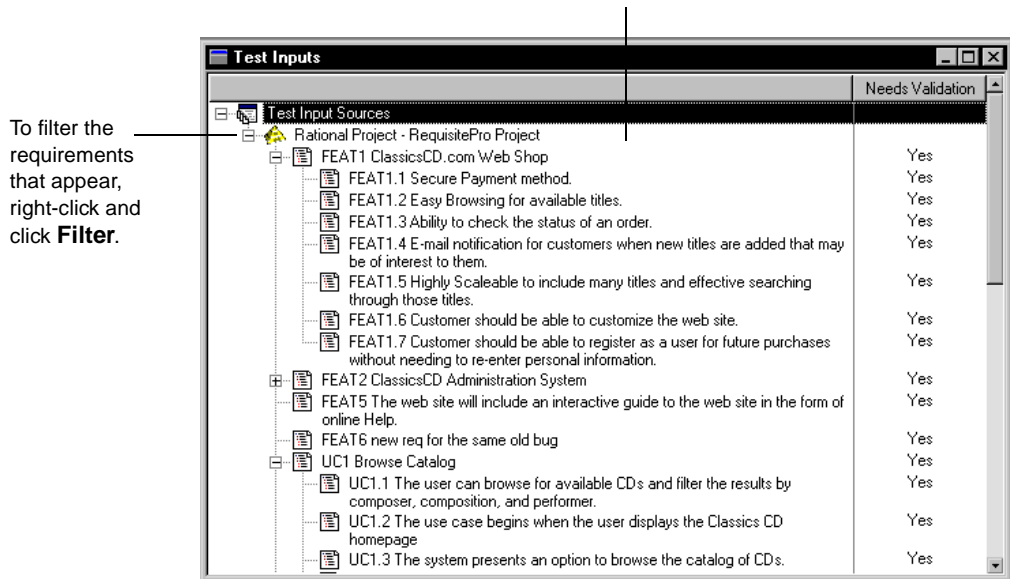
- Elements in a Rational Rose visual model

## Requirements from Rational RequisitePro

You can easily use RequisitePro requirements as test inputs. You or an administrator can use the Rational Administrator to associate a RequisitePro project with a Rational project. Once this is done, the requirements appear in the Test Input window after you log on to that project. You can then create an association between a requirement and a test case. The requirements themselves are created and managed in RequisitePro, but you can modify the properties of the requirements in TestManager.

In the following figure, the project has a RequisitePro datastore associated with it. When you open the Test Inputs window, you can see the requirements.

Rational RequisitePro requirements associated with the project.



## Model Elements from Rational Rose

If you use Rational Rose, then you can use Rose model elements as test inputs. You use the Rational Administrator to associate a Rose model with a project. You can then look at each individual model element in the Test Inputs window, and create an association between a model element and a test case.

## Custom Test Input Types

TestManager supports using test input types other than RequisitePro requirements or Rational Rose model elements. For example, you might want to use C++ language project files that contain project-specific language restrictions and metrics, such as permitted depth of inheritance. The test designer can create test cases that determine whether specific source modules adhere to these restrictions.

For TestManager to support an extensible test input type, you must implement a DLL with certain required functions for TestManager to call when necessary—for example, when connecting to or disconnecting from the test input source. This DLL is called a Test Input Adapter (TIA). The TIA functionality enables you to associate project files for the custom test type with test cases. For information about writing test input adapters, see the *Rational TestManager Extensibility Reference* manual.

To register a new test input type in TestManager:

- Click **Tools > Manage > Test Input Types**. Click **New**.

**Note:** If the **New** button is disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.

Click to register the source of the test input type.

Type the path to the DLL file.

The screenshot shows a dialog box titled "New Test Input Type" with three tabs: "General", "Sources", and "Statistics". The "General" tab is selected. The dialog contains the following fields and controls:

- Name:** A text input field.
- Description:** A large text area with a vertical scrollbar.
- Owner:** A dropdown menu with "admin" selected.
- Adapter:** A text input field for the DLL path.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

Annotations in the image include an arrow pointing from the text "Click to register the source of the test input type." to the "Sources" tab, and another arrow pointing from "Type the path to the DLL file." to the "Adapter:" text box.

After you register a new test input source, that source appears in the Test Inputs window.

## Creating a Test Plan

---

In TestManager, a test plan is an asset of a Rational project. You can have one or more test plans in a project. They can be organized in any way that makes sense for your organization. For example, you could have one test plan for the entire testing project, or you could have one test plan for each major component of the project.

You work with test plans in the Test Plan window. To open the Test Plan window, do one of the following:

- Click **File > Open Test Plan**.
- In the **Planning** tab of the Test Asset Workspace, expand **Test Plans**. Right-click the test plan and click **Open**.

## Editing and Creating Test Plans

TestManager provides you with an empty test plan named Test Plan 1 that you can use to start your planning. You can also create your own test plans.

To edit a test plan:

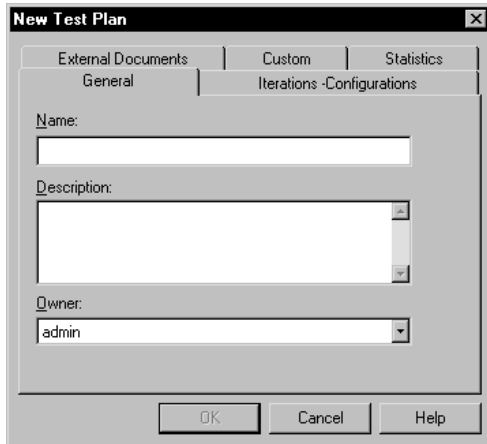
- From the **Planning** tab of the Test Asset Workspace, expand **Test Plans**. Right-click the test plan and click **Open**.

To create a new test plan, do one of the following:

- Click **File > New Test Plan**.

- From the **Planning** tab of the Test Asset Workspace, right-click **Test Plans** and click **New Test Plan**.

**Note:** If the **New Test Plan** menu command is disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.



## Properties of a Test Plan

A test plan has many properties. For example:

- The name of the test plan (required).
- A description of the test plan.
- The owner of the test plan. For information, see *Specifying the Owner* on page 34.
- The configurations associated with the test plan. For information, see *Defining the Configurations to Test* on page 35.
- The iterations associated with the test plan. For information, see *Specifying When to Run Tests* on page 40.
- Any external documents associated with the test plan. For example, you could associate a Microsoft Project to track effort, tasks, and progress.

The name of the test plan is required. For all other properties, you can add them when you first create the test plan, or add or change them later.

Any iterations and configurations associated with a test plan are automatically associated with any new test case folders that are direct children of the test plan. In other words, if you create a test case folder directly below a test plan in the Test Plan

window, that new folder inherits all iterations and configurations that are associated with the test plan. You can easily change the folder's associations if they aren't appropriate.

When you assign properties to a test plan, you can run reports based on those properties. For example, you can run Listing reports to determine which test plans are owned by each tester, or which iterations are associated with a test plan. The reports can give you valuable information about the state of your testing project. For information about reports, see *Reporting Results* on page 154.

To change the properties of a test plan:

- Right-click the plan in the **Planning** tab of the Test Asset Workspace or in the Test Plan window, and click **Properties**.

You can also copy an existing test plan, which copies all of its properties.

## Customizing the Properties of a Test Plan

When you create test plans, you can add your own properties and values.

Using the Customize Test Assets dialog box, you can define both the property itself (the label) and the values that can be used with that property. You can then access those properties in the **Custom** tab of the New Test Case or Test Case Properties dialog box.

To add customized properties and values to a test plan:

- Click **Tools > Customize > Test Plan**.

**Note:** You can also customize the properties and values of test cases, suites, builds, test scripts, and sessions from the **Tools > Customize** menu.

## Organizing Test Cases with Folders

---

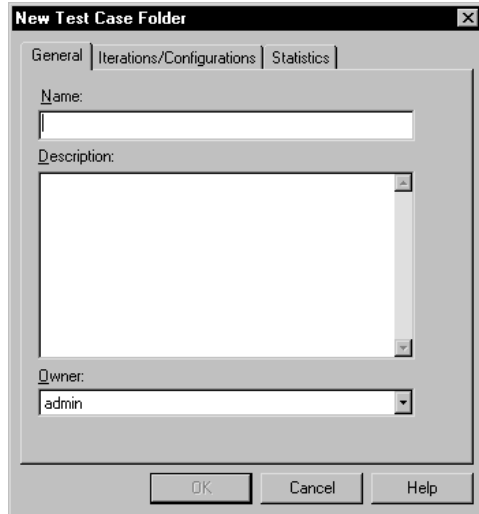
Within a test plan, you can create *test case folders* to organize your test cases hierarchically. You can organize your test cases in any way that makes sense for you. For example, you might have a test case folder:

- For each tester in your department.
- For each major use case of the system.
- For each major component in the application.
- For each phase of testing.

You can nest test case folders within test case folders. For example, you could have a folder for a tester, and then that tester could create folders for each piece of functionality she plans to test.

To create a test case folder:

- In the Test Plan window, right-click a test plan or a test case folder and click **Insert Test Case Folder**.



Just as with test plans, a test case folder has certain properties. For example:

- The name of the folder (required).
- A description of the folder.
- The owner of the folder. For information, see *Specifying the Owner* on page 34.
- The iterations associated with the folder. For information, see *Specifying When to Run Tests* on page 40.

The name of the folder is required. For all other properties, you can add them when you first create the folder, or add or change them later.

Any iterations and configurations associated with a test case folder are automatically associated with any new folders and test cases that are direct children of the test case folder. In other words, if you create a test case directly below a test case folder in the Test Plan window, that new test case inherits all iterations and configurations that are associated with the folder. You can easily change the test case's associations if they aren't appropriate.



Properties are useful when you run reports. For example, you could run a Listing report to see the iterations associated with each test case folder. For information about reports, see *Reporting Results* on page 154.

## Creating Test Cases

---

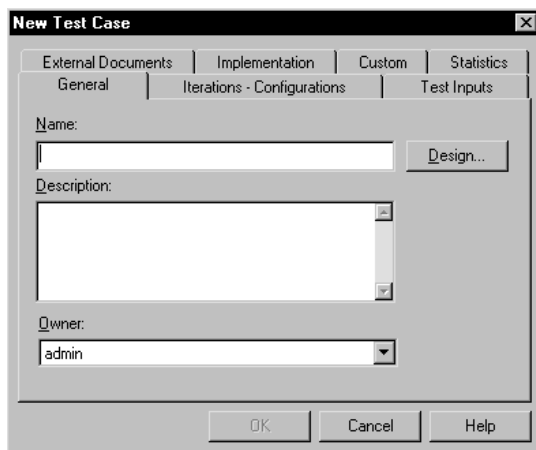
The test plan is centered on test cases. Once you've identified your test inputs and decided what you plan to test, you can create your test cases.

Use test cases to validate that the system is working the way that it's supposed to work and is built with the quality you need before you can ship it. The test case is the artifact in TestManager that answers the question, "What do I need to test?" You develop test cases to validate particular behaviors.

A test case always resides in a test case folder in a test plan.

To create a test case:

- In the Test Plan window, right-click a test case folder and click **Insert Test Case**.



### Properties of a Test Case

A test case has many properties. For example:

- The name of the test case (required).
- A description of the test case.
- The owner of the test case. For information, see *Specifying the Owner* on page 34.
- The configurations associated with the test case. For information, see *Defining the Configurations to Test* on page 35.

- The iterations associated with the test case. For information, see *Specifying When to Run Tests* on page 40.
- Any test inputs associated with the test case. For information, see *Setting up Traceability Using Test Inputs* on page 42.
- Any external documents associated with the test case.
- The manual and automated implementations of the test case. These are the actual test scripts that will be run. For information, see *Implementing Tests* on page 53.
- The design of the test case (in other words, the steps to be performed when the test case is implemented). For information, see *Specifying the Testing Steps and Verification Points* on page 46.
- Pre-conditions, post-conditions, and the acceptance criteria of the test case. For information, see *Specifying Conditions and Acceptance Criteria of Test Cases* on page 48.

The name of the test case is required. For all other properties, you can add them when you first create the test case, or add or change them later.

You can also add your own properties and values to a test case. For information, see *Customizing the Properties of a Test Plan* on page 31.

Properties are useful when you run reports. For example, you could run a Test Case Distribution report to see the test cases distributed over the owners. For information about reports, see *Reporting Results* on page 154.

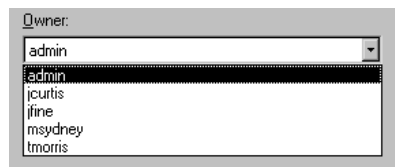
To change the properties of a test case:

- Right-click the test case in the Test Plan window, and click **Properties**.

## Specifying the Owner

You can define the owner of the test case from the **Owner** list in the **General** tab of the New Test Case dialog box.

The **Owner** list contains the User IDs of the test users that were added to the project through the Rational Administrator.



The owner is important for tracking purposes. For example, you could run a Test Case Distribution report to see the test cases distributed over the owners. For information about reports, see *Reporting Results* on page 154.

## Defining the Configurations to Test

You use configurations to test against different sets of attributes. For example, you might need to make sure that a test case works on certain operating systems and certain browsers. You could have configurations that test each operating system and browser separately. Or you might need to test that certain combinations of operating systems and browsers work together. (In many organizations, the configurations that you need to test against are determined by an analyst.)

For example, a test case might need to work on these operating systems:

- Windows 2000.
- Windows NT 4.
- Windows 98.
- Windows 95.

Each of these is a configuration consisting of an operating system.

In another example, a test case might need to work on a combination of an operating system and a Web browser:

- Windows 2000 and Internet Explorer 4.
- Windows 2000 and Netscape 4.
- Windows NT 4 and Internet Explorer 4.
- Windows NT 4 and Netscape 4.

Each of these is a configuration consisting of a combination of an operating system and a specific browser.

There are three main steps to working with configurations:

- 1 Define the configuration attributes and their possible values.

For example, *Operating System* is a configuration attribute, and *Windows 2000* and *Windows NT* are its values.

- 2 Define the specific configurations that you need to test.

For example, *Windows 2000* is one configuration, and *Windows NT* is another configuration.

- 3 Create configured test cases.

You create a configured test case by associating a configuration (created in step 2) with a test case.

Once you've defined the attributes, you can combine them in different ways to define the configurations. Defining attributes and configurations is an iterative process, and you will most likely continue to add and refine both throughout the testing project.

## **Defining Configuration Attributes and Their Values**

When you start planning, think about how you will combine the pertinent configuration attributes to define the configurations to test against. Keep in mind that you can run reports against these attributes after you run your test cases. For example, you can run a Test Case Results Distribution report, and distribute the results over the configuration variables. For these reports to be useful, you must define your attributes appropriately.

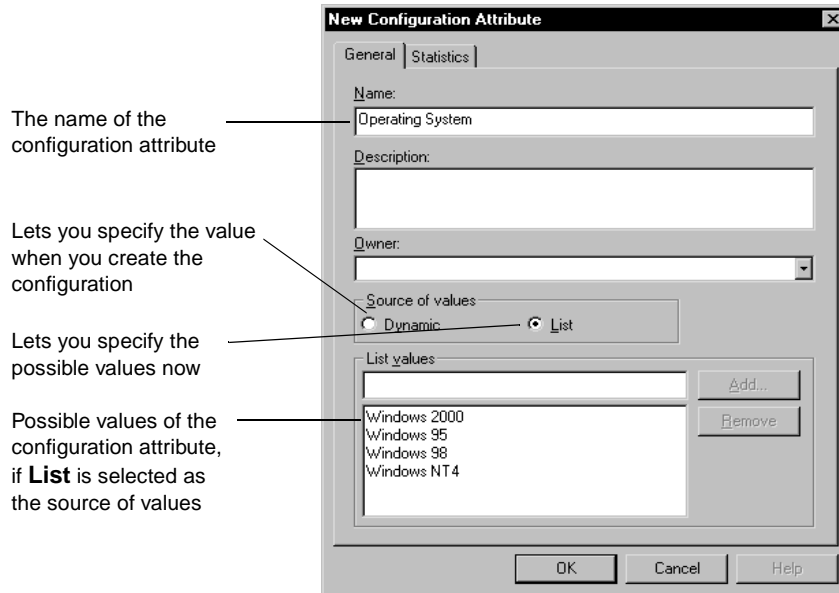
This process is iterative. Throughout the testing project, you'll probably continue to expand and refine this list.

For example, when you start the project, you may need to test only Windows 2000 and Window NT. Later in the project, the analyst may decide that you also need to test Windows 98. You can open the configuration attribute named *Operating System* and add *Windows 98* as a new value.

To start defining a configuration attribute:

- Click **Tools > Manage > Configuration Attributes**. Click the **New** button.

**Note:** If the **New** button is disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.



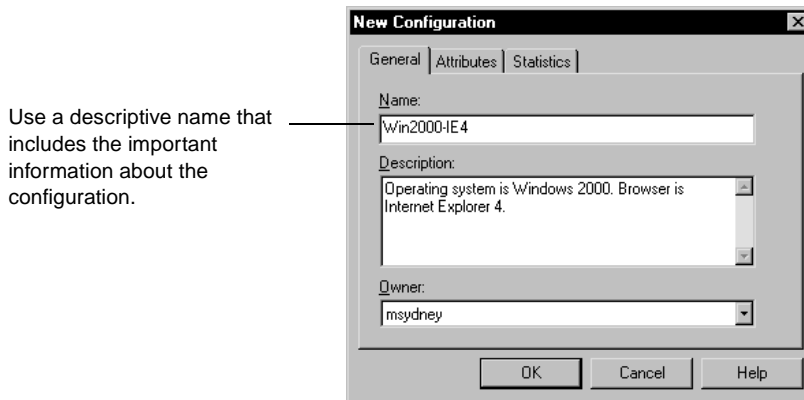
## Defining the Configurations You Need to Test

Now that you have defined the configuration attributes and their values, you define the configurations you need to test. This process is iterative. Throughout the testing project, you'll probably continue to expand and refine this list.

To start defining a configuration:

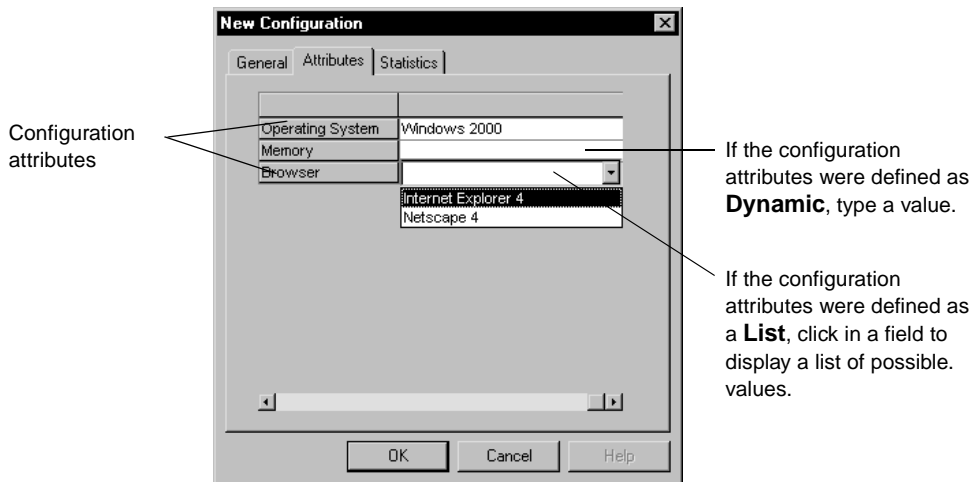
1 Click **Tools > Manage > Configurations**. Click the **New** button.

**Note:** If the **New** button is disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.



In this figure, the name of the configuration is Win2000-IE4. This name easily identifies that the configuration will be used for testing on a combination of Windows 2000 and Internet Explorer 4.

2 Click the **Attribute** tab.



In this tab, you can select a value from the list of values you defined when you created the configuration attributes. In this example:

- The Operating System is Windows 2000

- The Browser is Internet Explorer 4

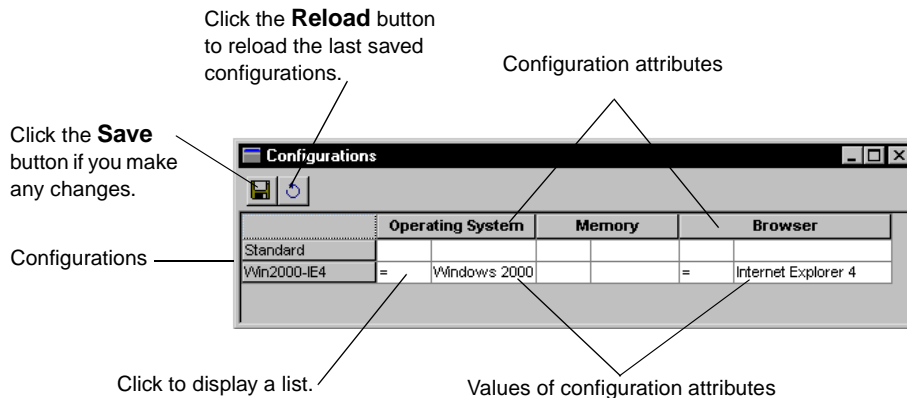
Since **Memory** is left blank, this configuration includes any memory value.

## Using the Configurations Window

You can easily view and edit any of your configurations in the Configurations window.

To open the Configurations window:

- Click **View > Configurations**.



When the Configurations window is open, you can insert configuration attributes and configurations from the **Edit** menu, or by right-clicking a row in the window.

## Creating a Configured Test Case

Once you've created configuration attributes and configurations, you can create configured test cases.

Configured test cases are useful when you need to validate that a piece of functionality works under various configurations. For example, suppose you have a test case that says, "Close the application." You need to validate that the test case passes on each of the following operating systems: Windows 2000, Windows NT 4, Windows 98, and Windows 95. You could create four configured test cases associated with the main test case. In order for the main test case to pass, all of its configured test cases need to pass.

After you run the configured test cases, you can modify the standard Test Case Results Distribution report so that it filters based on the configurations you are interested in (for example, operating system). When you run that report, you can see the test case results based on operating system. For information about reports, see *Reporting Results* on page 154.

You can associate a configuration with a test case in several ways:

- When creating a new test case, click the **Iterations - Configurations** tab in the New Test Case dialog box. Select the configurations to associate.
- When editing the properties of an existing test case, click the **Iterations - Configurations** tab in the Test Case Properties dialog box. Select the configurations to associate.
- In the Test Plan window, right-click a test case and click **Associate Configuration**. Select the configurations to associate. A message appears giving you the option of associating the configurations with all of the object's existing children.

You can follow similar steps to associate configurations with a test plan or a test case folder. When you associate configurations with an object, all new objects that are direct children of that object inherit that configuration. For example, if you associate a configuration named Windows 2000 with a test plan, all of the new test case folders created directly under that test plan will be associated with Windows 2000. You can always change an object's configurations when you create that object or at a later time.

If a configuration is associated with a test case, the configured test case appears under the test case in the Test Plan window.

## Specifying When to Run Tests

Many test organizations plan more test cases than can actually execute at any given time. You can create all of the test cases in TestManager, and then use iterations to identify which test cases actually need to run and when they need to pass.

An *iteration* is a defined span of time during a project. The end of an iteration is a milestone. An iteration says that at some point in time, the product has to meet a certain quality standard to reach a milestone. The quality standard is defined by the test cases that must pass.

Multiple iterations can be associated with a test case. The iterations indicate when the test case must pass. In many organizations, the tester works with an analyst or developers to determine at which iterations the test case needs to pass.

For example, at the beginning of a project you start to create all of the test cases that you can think of for the system. The analyst reviews your test plan and says that test cases 1, 2, 3, and 8 are important for the Construction 2 iteration. You go back into



TestManager and associate the Construction 2 iteration with these four test cases. During your testing, you come up with another test case. The analyst decides that this is an important test case for Construction 2, so you add that iteration to the test case.

TestManager provides you with an initial set of iterations based on the Rational Unified Process. (For a description of these iterations, see the Rational Unified Process documentation.) You can use these iterations, or add your own based on what makes sense for your organization.

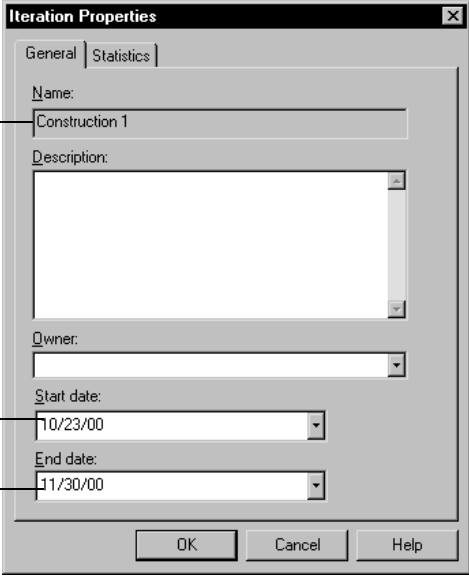
## Creating and Editing Iterations

To create or edit iterations:

- 1 Select **Tools > Manage > Iterations**.
- 2 Click **New** to create a new iteration, or select an existing iteration and click **Edit**.

**Note:** If the **New** and **Edit** buttons are disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.

You can also right-click **Iterations** in the **Planning** tab of the Test Asset Workspace.



The screenshot shows the 'Iteration Properties' dialog box with the 'General' tab selected. The dialog has a title bar with a close button. Below the title bar are two tabs: 'General' and 'Statistics'. The 'General' tab contains the following fields:

- Name:** A text box containing 'Construction 1'. A line points from the label 'Name of iteration' to this field.
- Description:** A large text area with a vertical scrollbar, currently empty.
- Owner:** A dropdown menu.
- Start date:** A date picker showing '10/23/00'. A line points from the label 'Start date of iteration' to this field.
- End date:** A date picker showing '11/30/00'. A line points from the label 'End date of the iteration' to this field.

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

## Associating Iterations with a Test Case

You can associate an iteration with a test case in several ways:

- When creating a new test case, click the **Iterations - Configurations** tab in the New Test Case dialog box. Select the iterations to associate.

- When editing the properties of an existing test case, click the **Iterations - Configurations** tab in the Test Case Properties dialog box. Select the iterations to associate.
- In the Test Plan window, right-click a test case and click **Associate Iteration**. Select the iterations to associate. A message appears giving you the option of associating the iterations with all of the object's existing children.

You can follow similar steps to associate iterations with a test plan or a test case folder. When you associate iterations with an object, all new objects that are direct children of that object inherit that iteration. For example, if you associate the iteration named Transition 1 with a test plan, all of the new test case folders created under that test plan will be associated with Transition 1. You can always change an object's iteration when you create that object or at a later time.

You can define a test case report so that it reports on specific information. For example, you could start with one of the standard test case reports and build a report that adds filtering for iteration. When you run that report, you can see the test case coverage based on a specific iteration. When all of the test cases that define your quality acceptance criteria for a given iteration pass, you have met your quality milestone.

## Setting up Traceability Using Test Inputs

As described in *Defining What to Test by Using Test Inputs* on page 26, test inputs help you decide what to test. When you create your test cases, you can associate test inputs with them. By doing this, you can determine if a test case needs to change because its associated test input changes.

The association also lets you determine if the test input is covered by a test case. For example, suppose you're using requirements as test inputs. When every test input is associated with a test case, you know that all of your requirements are covered. When every test case passes, you know that all of the test inputs have been validated.

To associate a test input type with a test case:

- In the Test Plan window, right-click a test case and click **Associate Test Input**.

The test inputs appear. When you select the test input, it becomes associated with that test case.

You can also associate a test input when you first create the test case, or in the test case's Properties dialog box.

When you run a test case report, you can modify that report so that it reports on specific information. For example, you could start with one of the standard reports and build a report that adds filtering for test input. When you run that report, you can see the test case coverage based on the test inputs.



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

- About designing tests
- Specifying the testing steps and verification points
- Specifying pre-conditions, post-conditions, and acceptance criteria of test cases
- Example of a test design

**Note:** For detailed procedures, see the TestManager Help.

## About Designing Tests

---

Once you've defined the features that you need to test, you need to decide how to do the testing. The activity of test design is primarily answering the question "How can I test this? How can I perform this test case?"

As part of the design of the test case, you need to identify:

- The steps required to interact with the application and system in order to perform the test.
- How to validate that the items or features you are testing are working properly.
- The pre-conditions of the test case—how to set up the application and system so that the test case can run.
- The post-conditions of the test case—how to clean up after the test case runs.
- The acceptance criteria—how to decide if the test case passed.

You should be able to design your tests based on test inputs such as feature descriptions and software specifications (for example, requirements) before or during the implementation of the actual system. This is a key aspect of making testing a parallel development with system implementation.

Someone should then be able to take the test design and an implementation of the system (with documentation) and know how to implement the test.

For example, if you are using an automated testing tool like Rational Robot, you should be able to start your tool and follow the steps documented in the test case's design to create a test script. The test script becomes an implementation of the designed test case, and therefore of the test case itself.

As another example, you could also look at all of the test designs (one for each test case) before you implement the test cases. You might find patterns in the test designs that indicate a more efficient way to implement the test cases. For example, you might see that every test design begins with a step that says "From the Start menu, start the application." You might decide that it doesn't make sense to record this step in every test script, because if the name of the application changes, all of the scripts would need to be changed. Instead, you might build a subroutine to start the application, and have the test scripts call that subroutine. This would become obvious by looking at the test designs.

You can also use the test design to assist you in creating a manual test. For information about manual tests, *Creating Manual Test Scripts* on page 56.

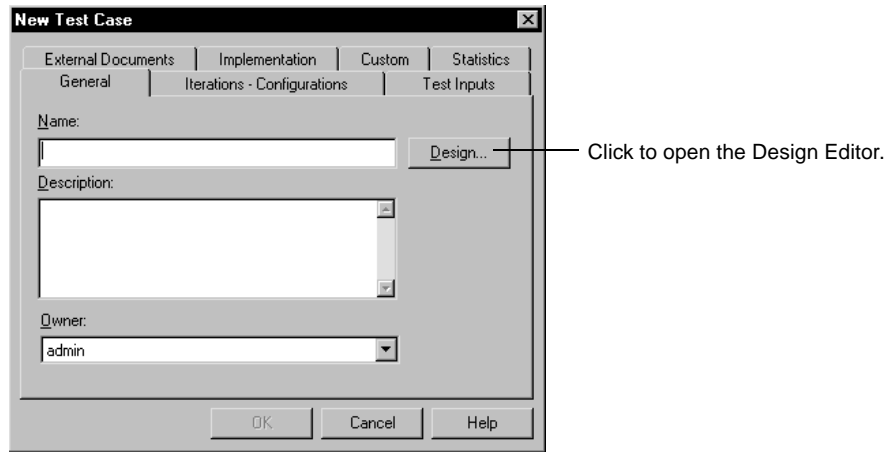
## **Specifying the Testing Steps and Verification Points**

---

You can design a test case when you first create the test case or at a later time.

To design a new test case:

- In the Test Plan window, right-click a test case folder. Click **Insert Test Case**.



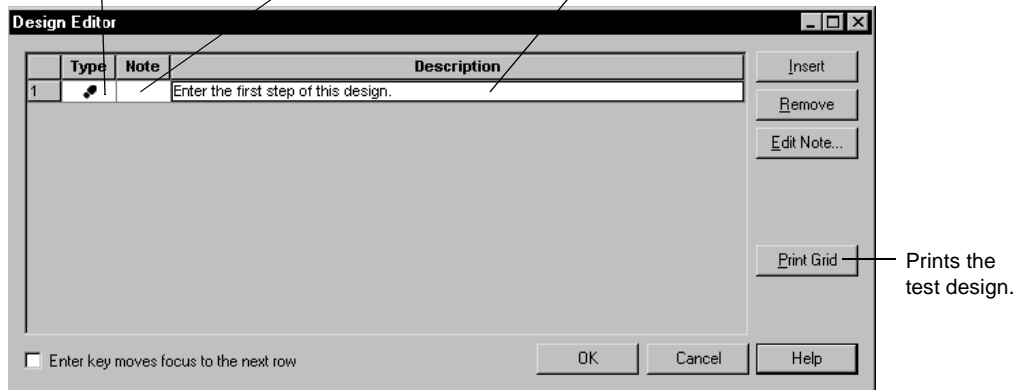
To design an existing test case:

- Right-click the test case in the Test Plan window and click **Design**.

Indicates whether a row is a step (footprint) or a verification point (check mark). Click to change.

Use to include a note.

Contains the step or verification point.



Use the Design Editor to include all of the steps and verification points that should be included in the test script:

**Step** – An action to be taken in the application or system.

**Verification Point** – A point in a test script that confirms the state of one or more objects.

When you click **OK** in the Design Editor, that design becomes a property of the test case.

The test design will evolve over iterations of the development process. As you learn more of the details of how the system will be implemented, you can add more steps and verification points to the design.

## **Specifying Conditions and Acceptance Criteria of Test Cases**

---

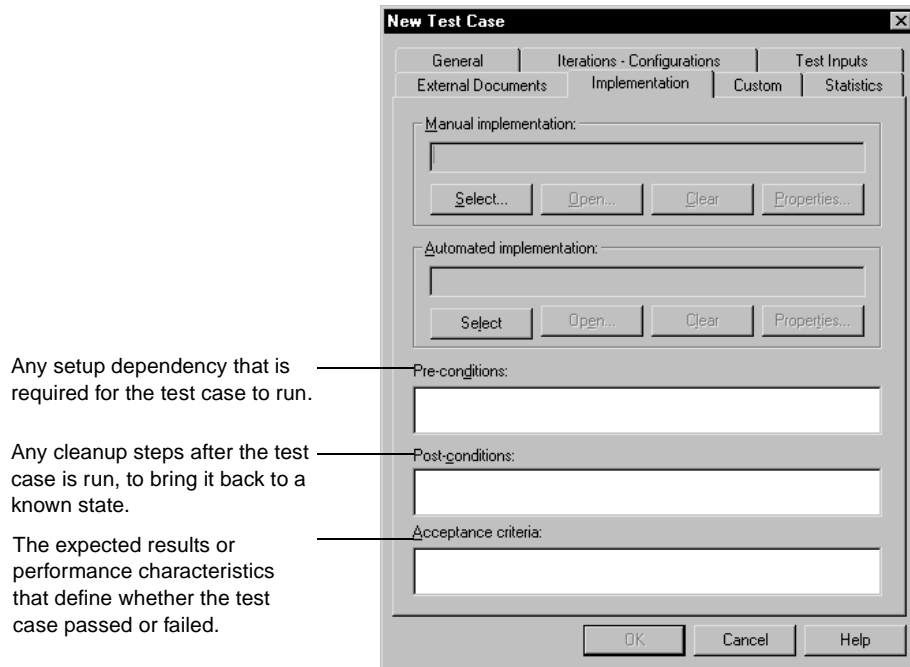
*Pre-conditions* and *post-conditions* provide information for the person executing the test. They describe the constraints on the system that must be true when an operation starts or end, therefore ensuring that the test case can run properly and that it leaves the system in an appropriate state. Failure of a pre-condition or post-condition does not mean that the behavior or function being tested did not work. It means that the constraint wasn't met.

The *acceptance criteria* indicates what needs to be true in order for a particular test case to pass.

To specify the conditions and acceptance criteria:



- 1 In the Test Plan window, right-click a test case. Click **Properties**.
- 2 Click the **Implementation** tab.



For example, if the test case needs to verify whether the response time for logging into a system is acceptable, then you might include the following information with the test case:

**Pre-condition** – You must have the proper user ID login available in the system and the system must be in a logged out state.

**Post-condition** – After you log in and successfully verify the test case, you need to log out (or bring the system back into a known state for the following tests).

**Acceptance criteria** – The response time range should be between .5 and 2.0 seconds for this test case to pass.

In another example, you could have five verifications in your test. However, at a certain point in time, only three of them might need to pass for the test case to pass. In this case, the acceptance criteria might change based on the iteration.

## Example of a Test Design

---

This section gives an example of a design for a test case. Because the test design is based on test inputs, it can be developed before any code is written.

In this example, you are testing an automated teller system (ATM). Your requirements include a use case for withdrawing money from a specified account type. Another requirement specifies that to perform any transactions with the ATM, the user must be identified and validated. From these requirements, you have defined a test case to ensure that you can withdraw a sum of money from a checking account when the account contains more money at the start of the transaction than the amount withdrawn.

The first iteration of the test design might be as follows:

### **Pre-conditions**

- Ensure that we have a valid account set up and know the user ID and validation information (password or PIN).
- Ensure that there is a checking account for the user and that we know the current balance.
- The current balance must be greater than zero.

### **Design**

- Step – Identify the user to the ATM and validate.
- Verification Point – Make sure that we're logged in.
- Step – Select "Checking" as the account type and "Withdraw" as the transaction.
- Step – Specify the amount to withdraw where the amount is less than the current balance.
- Verification Point – Ensure that the amount dispensed matches the amount specified.
- Verification Point – Run the account balance transaction to ensure that the new balance equals the old balance minus the amount withdrawn.

### **Post -Conditions**

- Make sure that the user is logged off of the ATM.

### **Acceptance Criteria**

- All verifications must succeed.

As more details of the system become available—as you move through iterations of artifacts like visual models, software specifications, prototypes, and so on—you can add more detail to the test design. For example, you might learn later that users will identify themselves via a card and PIN. You could update the design to have steps to insert the card, enter the PIN, and retrieve the card at the end.



This chapter describes how to implement tests. It includes the following topics:

- About implementing tests
- Implementing built-in test script types and suites
- Implementing extensible test script types
- Creating manual test scripts
- Associating an implementation with a test case
- Implementing tests as suites

**Note:** For detailed procedures, see the TestManager Help.

## About Implementing Tests

---

After you've created the test design for each test case, you're ready to implement the test case. You *implement* a test case by building a test script and then associating that test script with the test case.

Implementation is different in every organization. You can use your preferred tools to build any kind of test script appropriate for your testing environment.

For example, one testing organization might decide to implement all of the test cases by recording the test script using Rational Robot.

Another organization might decide to write modular pieces of software using a combination of Visual Test scripts, batch files, and Perl scripts, and then programmatically tie them together in a higher level script.

After you implement a test script, you can associate it with a test case in TestManager. For information, see *Associating an Implementation with a Test Case* on page 60.

You can then run the test case or the test script in TestManager. You can also insert the test script into a suite and run the suite. For information about running implementations, see *Executing Tests* on page 89.

## Implementing Built-in Test Scripts Types and Suites

---

TestManager is tightly integrated with Rational's test implementation tools. Starting from TestManager, you can easily implement:

- Automated test scripts recorded in Rational Robot
- Manual test scripts created in Rational ManualTest
- Suites created in Rational TestManager

### Automated Test Scripts Recorded in Rational Robot

TestManager is tightly integrated with Rational Robot, and therefore comes with built-in support for implementing the following test script types:

- **GUI** – A test script written in SQABasic, a Rational proprietary Basic-like scripting language. GUI scripts are used primarily for functional testing.
- **VU** – A test script written in VU, a Rational proprietary C-like scripting language. VU test scripts are used primarily for performance testing.

To start recording a test script from TestManager:

- Click **File > Record Test Script > GUI** or **VU**.

This starts Robot and opens the Record dialog box.

**Note:** When you choose to record a VU test script, you actually begin recording a session. You can choose to generate VU or VB test scripts from the recorded session.

For detailed information about recording test scripts, see the *Using Rational Robot* manual and the Robot Help.

### Manual Test Scripts Created in Rational ManualTest

TestManager is also tightly integrated with Rational ManualTest, and therefore comes with built-in support for implementing the following test script type:

- **Manual** – A set of testing instructions to be run by a human tester.

For information, see *Creating Manual Test Scripts* on page 56.

### Suites Created in TestManager

TestManager allows you to build test *suites* from test scripts, test cases, and other items. Suites provide great flexibility and power for creating multi-faceted functional and performance tests.

Suite basics are covered in *Implementing Tests as Suites* on page 62. For details about planning, creating, and interpreting functional testing suites, see *Creating Functional Testing Suites* on page 173. For details about planning, creating, and interpreting performance testing suites, see *Creating Performance Testing Suites* on page 243.

## Implementing Extensible Test Script Types

---

TestManager's extensible test script type functionality enables you to implement tests using any tool that is appropriate for your testing environment. There are two ways to extend TestManager to support a new test script type.

You can create a new test script type that is based on the Command Line test script type and that uses the Command Line Test Script Console Adapter (TSCA) provided with TestManager. The advantage of this method is simplicity: it requires no custom programming. The only requirement is that the test scripts you want to run from TestManager can be executed from the command line.

The drawback of these scripts is that, while TestManager can execute them individually and also in suites that include test scripts of other types, they are not fully integrated into TestManager. For example, any procedures required to make these test scripts executable must be performed outside of TestManager.

The other way to extend TestManager is to create a new custom test script type. This requires that you develop programs implementing the C-language APIs described in chapters 2-4 of the *Rational TestManager Extensibility Reference* manual. Custom test script types are fully integrated into the TestManager framework, but they require considerably more effort to provide.

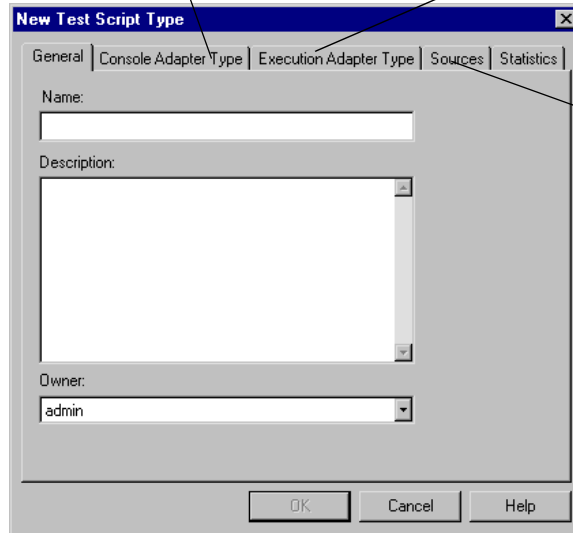
Which ever way you choose to incorporate a new test script type into TestManager, you must register the type. To register a new test script type in TestManager:

- Click **Tools > Manage > Test Script Types**. Click **New**.

**Note:** If the **New** button is disabled, you do not have Administrator privileges. See the *Using the Rational Administrator* manual or Help for information.

Click this tab to specify the console adapter.

Click this tab to specify the execution adapter.



Click this tab to specify the test script type source.

After you register the test script type, you should be able to open and run a test script of that type from TestManager. Depending on how the adapter is implemented, you might also be able to create and edit test scripts of that type.

## Creating Manual Test Scripts

---

Rational TestManager is tightly integrated with Rational ManualTest.

Rational ManualTest lets you create and run test scripts for tests that you cannot automate. A *manual test script* is a set of testing instructions to be run by a human tester. The script can consist of steps and verification points that you type into an editor.

A *step* is an instruction to be carried out by the tester when a manual test script is run. This could be as simple as a single sentence (such as “Reboot the computer”) or as complex as a whole document. In general, a step consists of one or two sentences.



Within a manual test script, a *verification point* is a question about the state of the application (for example, “Did the application start?”). A verification point can consist of any amount of text but is likely to be one or two sentences, usually ending with a question mark.

After you create a manual test script, you can associate it with a test case. When you run the test case or manual test script, the script opens in ManualTest.

When you run a manual test script, you perform each step and indicate whether each verification point passed or failed. You can then open the Test Log window of TestManager and see the results. If all of the verification points passed, then the script passes. If any verification points failed, then the script fails.

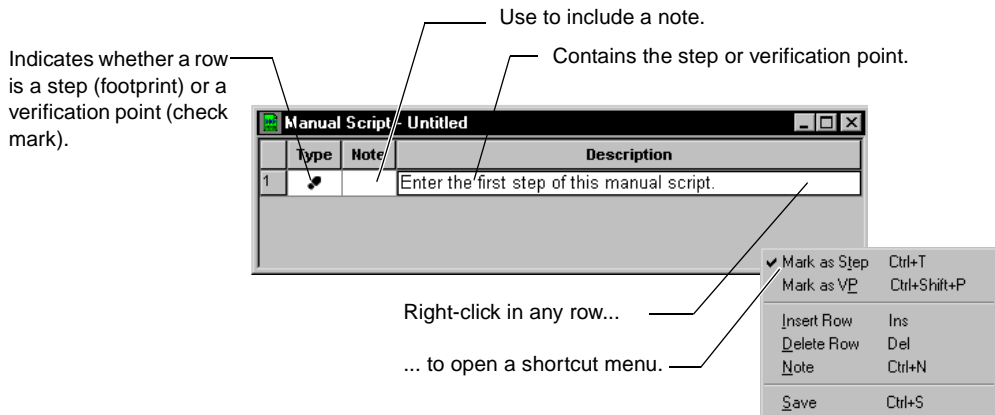
As with other types of test scripts, you can include your manual test scripts in TestManager reports.

**Note:** For detailed procedures about manual test scripts, see the Rational ManualTest Help.

## Starting Rational ManualTest

To start ManualTest and create a new manual test script:

- In TestManager, click **File > New Test Script > Manual**.



## Example of a Manual Test Script

The following manual test script contains five steps and four verification points.

- The steps are actions for you to take when you run the script.
- The verification points are questions for you to answer.

The footprint indicates a step to be performed.

The check mark indicates a verification point that can pass or fail.

The Note icon indicates that a note exists. Click the icon to open the note.

	Type	Note	Description
1	⚙️	📝	Loosen the 4 thumb screws on the rear of the case. Slide open the two side panels to reveal the drive cage and motherboard housing.
2	⚙️		Detach the power supply, the power LED, keylock switch, hardware reset switch, power on, HDD LEF, hardware suspend, and speaker connectors from the motherboard. Slide out the motherboard cage.
3	⚙️	📝	Record the serial number on the motherboard.
4	✓		Is the motherboard a BP6e model with a revision number greater than 1.0a?
5	⚙️		Detach the processor fan power connector from the motherboard.
6	⚙️		Remove the processor by pushing the lever on the socket outward and then pulling upward. The processor should shift slightly.
7	✓		Inspect the CPU. Are any of the pins on the bottom side bent or missing?
8		📝	Inspect the capacitors near the CPU socket. There should be 12 of them grouped together. Are any of them cracked?
9	✓		Are any of the traces running from the CPU socket damaged (burned or severed)?

## Setting the Default Editor for Manual Test Scripts

You can use either the grid editor or the text editor when you create a manual test script. The grid editor is a structured editor that makes it easy to enter your steps and verifications points. The text editor is a free-form editor that makes it easy to manipulate text.

Grid editor →

	Type	Note	Description
1	⚙️		Enter the first step of this manual script.

Text editor →

Enter the first step of this manual script.

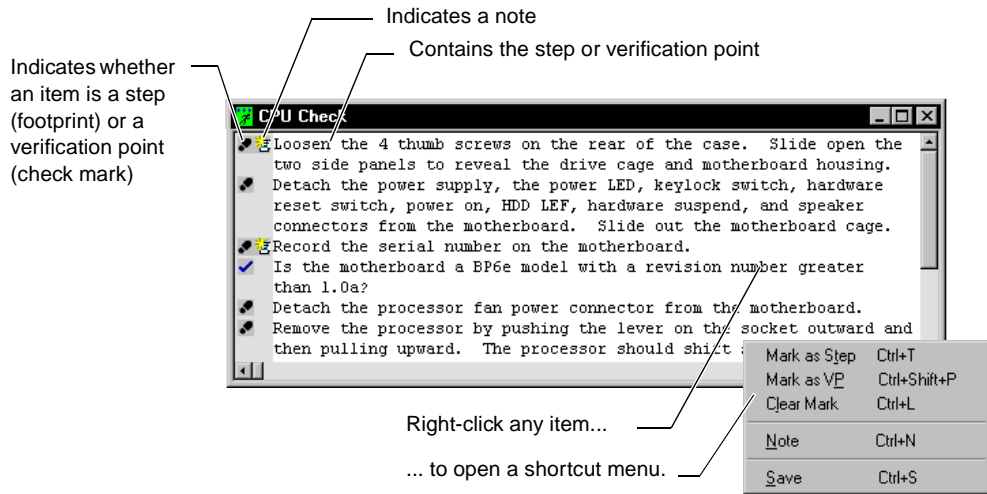
To set the default editor in ManualTest:

- Click **Tools > Options**.

This setting takes effect the next time you create or open a manual test script.

**Note:** Instead of using either editor, you can include an external file in the Test Script Properties dialog box.

In the text editor, you use a shortcut menu to mark items as steps and verification points, and to create and view notes.



The start of an item (step or verification point) is indicated by the footprint or check mark icon. All lines that do not begin with either of these icons are part of the previous item.

## Including an External File in a Manual Test Script

Instead of typing the steps and verification points into the grid or text editor, you can include an external file. This file can contain all of the instructions to be used in the manual test script. You can create this file with Notepad, Microsoft Word, or any other program for which there is a file association.

To include an external file in a manual test script.

- 1 Open an existing manual test script.
- 2 Click **File > Properties**.
- 3 Click the **Specifications** tab and include the file.

When you run the script, the external file opens so that you can follow the instructions.

**Note:** When you run a manual test script that includes an external file, you can indicate the results of the entire script (Pass or Fail), but you cannot indicate the results of the individual verification points.

## Creating Script Queries

Rational ManualTest provides *queries* that help you select test scripts in your Rational project. Queries let you specify the fields that appear in selection dialog boxes, how the test scripts are sorted, and which test scripts appear.

Use *filters* in your queries to specify the information that is retrieved from a project. Filters help you make queries more specific by narrowing down the information that you are searching for. You can build simple filters or combine simple filters into more complex ones. You use filters when you create or edit a query.

To create a new query:

- Click **Tools > Manage Script Queries**.

## Customizing Test Assets

When you create a manual test script, you can add custom properties to tailor the terminology associated with the scripts to the standards and practices used within your organization.

You can do the following to the properties of a script:

- Add up to three custom properties and values. (These appear in the **Custom** tab of the Test Script Properties dialog box.)
- Add new operating environments and modify or delete existing ones.
- Add a new purpose or modify or delete existing ones. You assign a purpose to indicate why you would use a script.

To see the standard properties of a manual test script:

- Click **File > Properties**.

To customize a manual test script:

- Click **Tools > Customize Test Script**.

You can define both the property itself (the label) and the values that can be used with that property.

For more information about customizing a script, see the ManualTest Help.

## Associating an Implementation with a Test Case

---

Once you've created an implementation, you can associate it with a test case. You can then run the test case, which runs its implementation. By associating test scripts with test cases, you can run reports that provide test coverage information.

TestManager comes with built-in support for associating the following types of implementations:

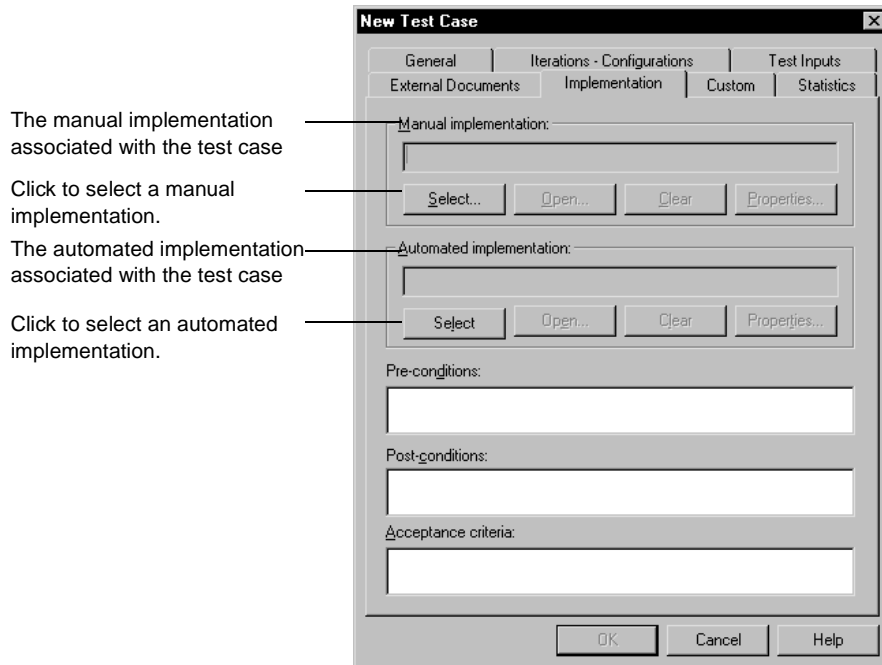
- GUI test scripts

- VU test scripts
- VB test scripts
- Manual test scripts
- Command-line executable programs
- Suites

TestManager also supports associating other test script types that you have registered. For information, see *Defining Extensible Test Script Types* on page 12.

To associate an implementation with a test case:

- 1 In the Test Plan window, right-click a test case. Click **Properties**.
- 2 Click the **Implementation** tab.



You can have at most two implementations associated with a test script: one manual and one automated. If both are associated with a test case, TestManager will run the automated implementation when you run the test case. For information about running test cases, see *Running Test Cases* on page 92.

## Implementing Tests as Suites

---

Suites are another way to implement tests in TestManager. Suites are categorized as either functionally-based suites or performance-based suites.

A suite shows a hierarchical representation of the tasks that you want to test, or workload that you want to add to a system. It shows such items as the user or computer groups, the resources assigned to each group, which test scripts the groups run, and how many times each test script runs.

When structuring a test using a suite, you can:

- Define user or computer groups, and apply resources to them specifying where they run

Groups are collections of virtual testers (emulating actual users or computers) that perform similar tasks in the application.

- Add test scripts

Test scripts are sets of instructions. Test scripts can be used to navigate the user interface of an application to make sure all features work, or to test the activities that the application performs behind the interface.

- Add subordinate suites

Subordinate suites are structures that specify how test scripts are played back and how test are run.

- Add test cases

A test case is a testable and verifiable behavior in a target test system.

- Create and add scenarios.

Scenarios are modular groups of test scripts and other items in a suite that can be used by more than one user or computer group.

- Specify a selector

Selectors allow you to indicate how often and in what order to run test scripts in suites.

- Add a delay

Delays tell TestManager to pause for a given length of time before executing the next item in a suite.

- Define synchronization points

Synchronization points are definable places in suites where all virtual testers stop and wait for other virtual testers.

- Add a transactor

Transactors allow you to specify the number of user-defined transactions that a virtual tester performs in a given time period during the suite run.

While performance tests make use of all these feature, not all apply to functional tests. If you are creating a functional test, the following suite elements are most beneficial to you:

- Computer groups
- Test scripts
- Suites
- Test cases
- Certain types of selectors

Additionally, more complex functional tests might include:

- Scenarios
- Delays
- Synchronization points

For more information on functional testing suites, see Chapter 8, *Creating Functional Testing Suites*.

For more information on performance testing suites, see Chapter 11, *Creating Performance Testing Suites*.

Certain activities associated with creating scripts are independent of the type of suite. These topics are discussed in the remainder of this section. They include:

- Defining computers and computer lists
- Opening suites
- Editing test scripts
- Editing suites, including:
  - Editing suite properties
  - Editing user group information and user settings
  - Editing Agent computer settings
  - Setting system environment variables

- Setting TSS environment variables
- Changing the number of start test scripts
- Limiting the number of test scripts
- Changing the way random numbers are generated
- Setting shared variables
- Printing and exporting a suite
- Saving a suite

## Defining Computers and Computer Lists

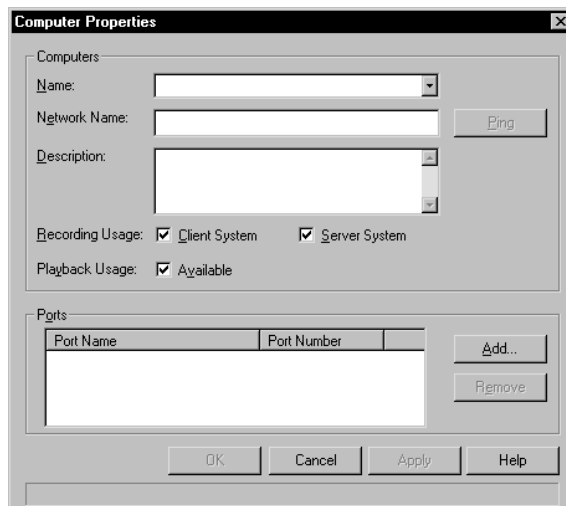
For both functional and performance tests, you can specify whether to run tests on the Local computer, any defined Agent computer, or groups of computers.

The Local computer is always the default. Unless you specify other computers, tests can only run on the Local computer.

### Adding a New Computer

To add a new computer definition to TestManager:

- Click **Tools > Manage > Computers**. Click the **New** button.



When you add a new computer definition to TestManager, you can include the following properties:

- **Name** - A name for the computer as appropriate to the testing environment.



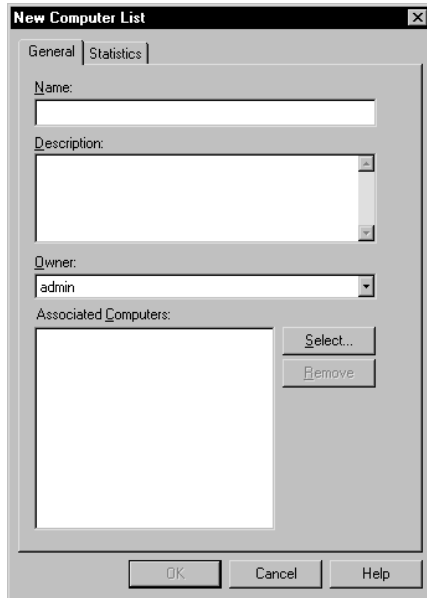
- **Network Name** - The name of the computer as recognized on the computer network. This may or may not be the name of the computer as defined above. To make sure you have the correct network name for the computer, click **Ping** to have your system search for the computer and respond.
- **Description** - (Optional) A description of the computer, perhaps noting its role in the testing process.
- **Recording Usage** - Specify whether the computer will be seen as a client or server system during recording.
- **Playback Usage** - Specify whether the system will be available for test script playback.
- **Port Information** -TCP/IP port information associated with the system. For information about port settings, see Appendix A, *Configuring Local and Agent Computers*.

## Creating a Computer List

After you have defined computers within TestManager, you can define lists of computers. This is useful if you want to run tests on several computers and only want to reference that group once, or if a series of computers has as similar configuration or organizational task.

To create a computer list:

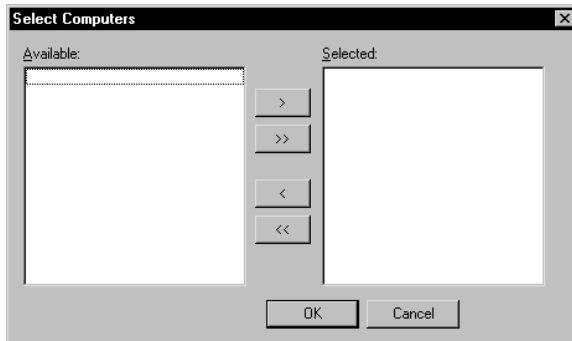
- Click **Tools > Manage > Computer Lists**. Click the **New** button.



After you have defined the computer list with a name and description, add computers to the list. Computers must have been added to TestManager individually before they can be included in a computer list

To add computers to a computer list:

- From the Computer List Properties dialog box, click **Select**.



Once you have defined computers and computer lists, they are resources available for running suites.

## Opening a Suite

To work with a suite, you must open it. You can open a suite from a menu or from the Test Asset Workspace.

To open a suite from a menu:

- Click **File > Open Suite**.

To open a suite from the Test Asset Workspace:

- From the **Execution** tab, double-click the suite in the tree.

## Editing a Test Script

While you are working with a suite, you may want to edit a test script. Through TestManager, you can:

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

### Editing the Properties of a Test Script

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

To edit the properties of a test script:

- Choose the test script, then click **Edit > Properties**.

The image shows a 'Test Script Properties' dialog box with the following fields and controls:

- Name:** Text box containing 'Calcualte Hours'.
- Description:** Text area.
- Owner:** Dropdown menu.
- Purpose:** Dropdown menu.
- Environment:** Dropdown menu.
- Referenced session:** Text box with a 'Clear' button.
- Type:** Check box labeled 'Developed'.
- Buttons:** 'OK', 'Cancel', and 'Help' at the bottom.

## Editing the Text of a Test Script

To edit the text of a test script:

- Select the test script to edit, and click **Edit > Open Test Script**.

TestManager starts the application appropriate to editing that script type. For example, for VU test scripts, TestManager starts Rational Robot. Robot opens with the selected test script ready to edit in a window. For Visual Basic or Java test scripts, TestManager starts the appropriate registered application, such as VB6 Integrated Development Environment (IDE), or Notepad if no IDE is associated with the selected script type.

For information about editing other script types, see the appropriate Rational Test Script Services API documentation.

## Editing a Suite

While you are working with a suite, you might want to want to edit its contents. You might want to:

- Edit the properties of the suite.

For example, the suite name, its description, or specifications.

- Edit the items in the suite.

For example, you might want to change the properties of a selector from sequential to random with replacement, or the properties of a transactor from coordinated to independent.

- Replace items in a suite.

For example, you might want to replace a script or a scenario with a new one.

- Edit user group information.

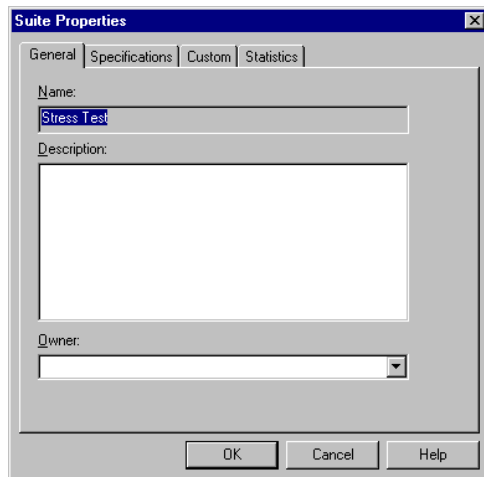
For example, you might want to change a user group from fixed to scalable, and/or change the computers on which the user group runs.

## Editing the Properties of a Suite

A suite has properties associated with it that can make it unique and help you differentiate it from similar suites. Examples of suite properties include a description of the suite and the owner of the suite.

To edit the properties of a suite:

- Open the suite to edit, then click **File > Properties**.



## Replacing Items in a Suite

Use inline editing to replace any item in a suite except delays and selectors. Replacing an item—especially an item high in the suite structure—is often easier than deleting the item and adding another one. For example, your suite may contain a complex structure of user groups, test scripts, and scenarios. Rather than deleting an item and recreating the suite structure underneath, you can replace the item.

**Note:** To be able to replace an item in a suite, you must first verify the value of one of the TestManager suite creation options. Click **Tools > Options > Create Suite**, and clear the **Show numeric values** check box. This option allows you to perform inline editing and thus rename suite items in the tree.

To replace an item:

- Select the item, then type over the new item name.

## Editing Items in a Suite

As your testing process evolves over time, you may want to edit the properties associated with suite items and how they are used within the suite. You can edit the run properties of:

- Test cases
- Test scripts
- Suites
- Delays
- Scenarios
- Selectors
- Synchronization points
- Transactors

To edit the properties of an item:

- Select that item, right-click and select **Run Properties**.

TestManager displays the same dialog box that appeared when you created the item. You can edit the values in each box.

**Note:** When you edit the run properties of a suite item, the changes affect only that instance of the item.

For example, to edit the properties of a transactor, right-click a transactor, then select **Run Properties**.

## Editing Information for All User Groups

At times, you may want to edit information for more than one user group. For example, you might want to change the scaling proportion of the user groups. Although you can edit each user group individually, it is much easier to edit the information for all of the user groups at the same time.

To edit information for all user groups:

- Click **Suite > Edit Groups**.

Fixed user groups:			Total: 0
User Group	Computers	Number of Users	

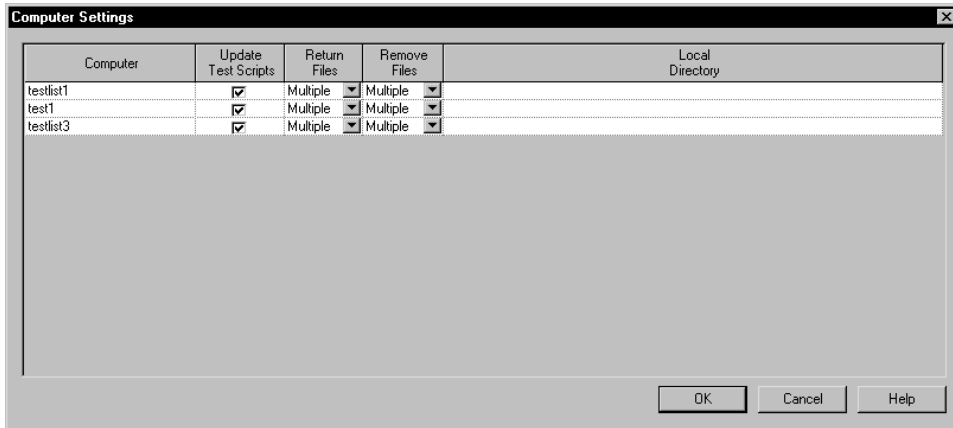
Scalable user groups:			Total: 100
User Group	Computers	%	
Accounting	Local computer	20.000	
Data Entry	Local computer	30.000	
Sales	Local computer	50.000	

## Editing the Settings of an Agent Computer

You may want to change the default settings associated with an Agent computer. The configuration of that Agent system may have changed and the settings in TestManager need to reflect this.

To change the computer settings:

- Click **Suite > Edit Computers**.

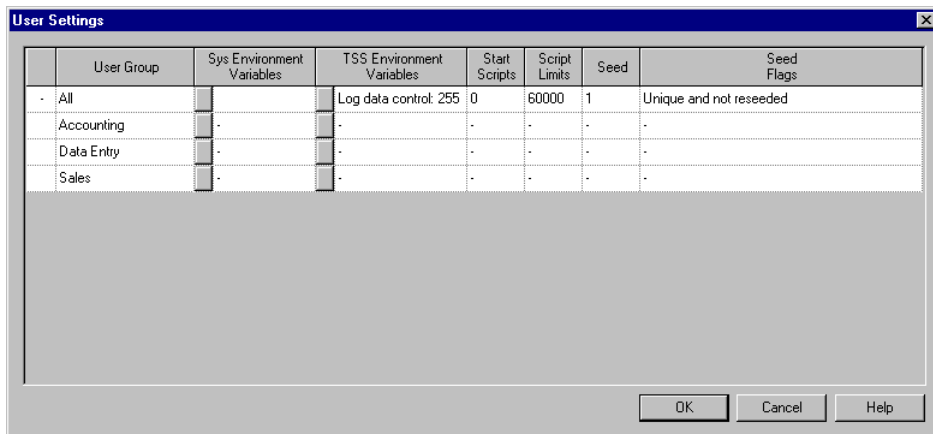


## Editing the User Settings

You may want to change the default settings associated with virtual testers. In particular, it is often useful to change which information is logged when you run a suite by setting TSS environment variables. For example, if you are having problems running a suite, set one virtual tester to log all environment variables and the other virtual testers to log failed environment variables so that you can investigate the problem more thoroughly.

To edit virtual tester settings:

- Click **Suite > Edit Settings**.



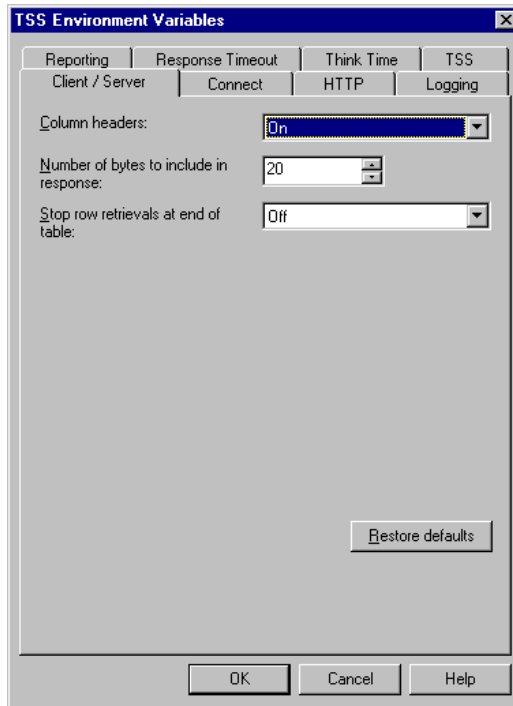


## Initializing TSS Environment Variables

You can initialize the value of most TSS environment variables within TestManager. When an environment variable is initialized through the TestManager interface, the value is in effect for an entire suite run unless the test script specifically changes the value.

To initialize the value of a TSS environment variable:

- Click **Suite > Edit Settings**, then click the button in the **TSS Environment Variables** column of the User Settings dialog box.

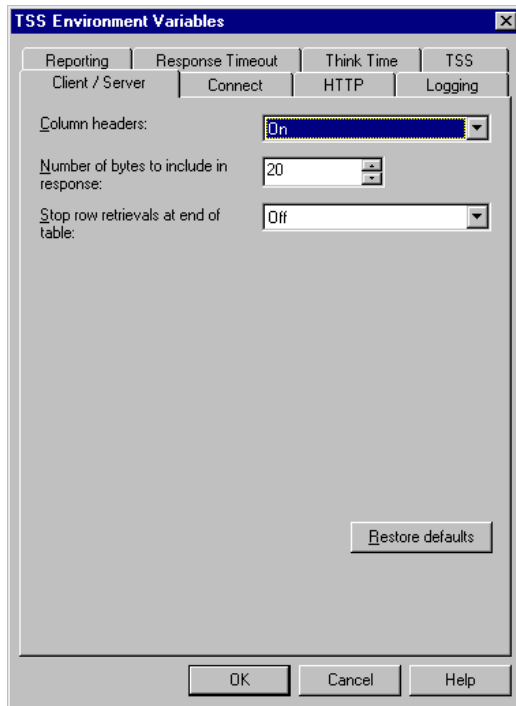


**Note:** For more information on TSS environment variables and their meanings, see the *Rational TestManager Extensibility Reference* manual and the appropriate API documentation.

## Initializing Client/Server Environment Variables

To initialize client/server environment variables

- Choose the **Client/Server** tab in the TSS Environment Variables dialog box.



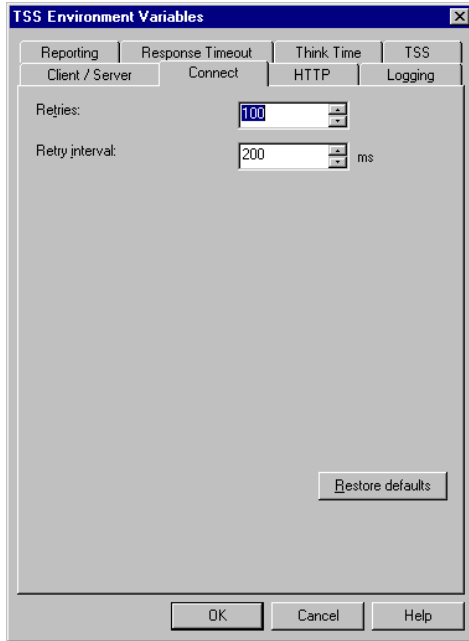
Values for client/server environment variables can be initialized as follows:

Variable	Description	Potential values	Default value
COLUMN_HEADERS	Column headers	ON OFF	ON
SQLNRECV_LONG	Number of bytes to include in response	0-2000000000	20
TABLE_BOUNDARIES	Stop row retrievals at end of table	ON OFF	OFF

## Initializing Connect Environment Variables

To initialize connect environment variables:

- Choose the **Connect** tab in the TSS Environment Variables dialog box.



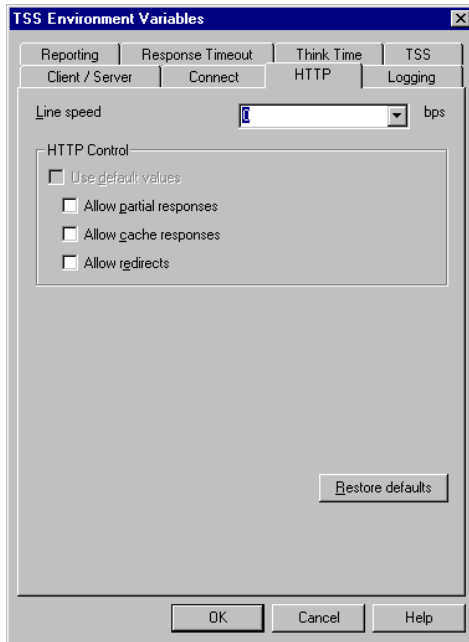
Connect environment variables apply to scripts that use HTTP and socket protocols only. Values for connect environment variables can be initialized as follows:

Variable	Description	Potential values	Default value
CONNECT_RETRIES	Retries	0-2000000000	100
CONNECT_RETRY_INTERVAL	Retry interval	0-2000000000 ms	200

## Initializing HTTP Environment Variables

To initialize HTTP environment variables:

- Choose the **HTTP** tab in the TSS Environment Variables dialog box.



Values for HTTP environment variables can be initialized as follows:

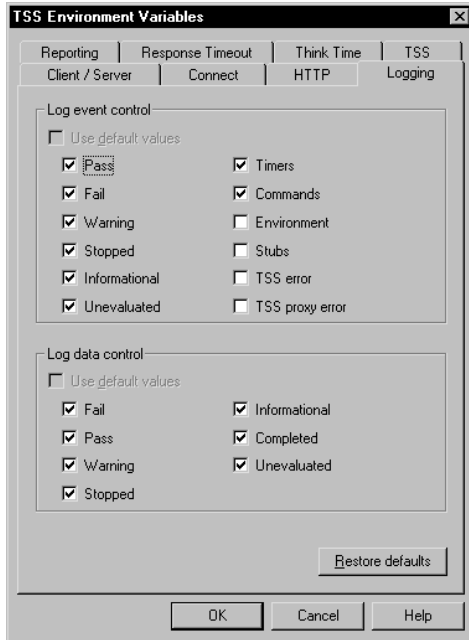
Variable	Description	Potential values	Default value
LINE_SPEED	Line speed	0-2000000000 Standard selectable values: 9600 14400 28800 33600 56600 64000 128000	0
HTTP_CONTROL	Http Control	integer indicating 0 or more of: 0 (exact match) Allow partial responses (HTTP_PARTIAL_OK) Allow redirects (HTTP_PERM_REDIRECT_OK and HTTP_TEMP_REDIRECT_OK) Allow cache response (HTTP_CACHE_OK)	0

## Initializing Logging Environment Variables

Logging environment variables specifically refer to events that appear in the log files associated with a suite run.

To initialize logging environment variables:

- Choose the **Logging** tab in the TSS Environment Variables dialog box.



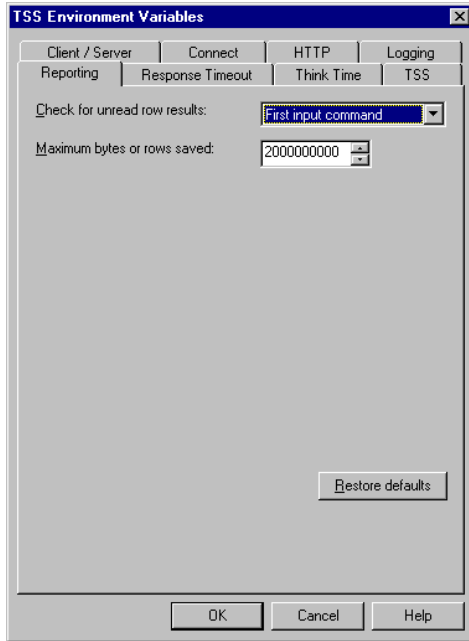
Values for the Logging environment variables can be initialized as follows:

Variable	Description	Potential values	Default values checked
TSS_LOG_EVENT_CONTROL	Log event control	Pass Fail Warning Stopped Informational Unevaluated Timers Commands Environment Stubs TSS Error TSS Proxy	Pass Fail Warning Stopped Informational Unevaluated Timers Commands
TSS_LOG_DATA_CONTROL	Log data control	Pass Fail Warning Stopped Informational Completed Unevaluated	Pass Fail Warning Stopped Informational Completed Unevaluated

## Initializing Reporting Environment Variables

To initialize reporting environment variables:

- Choose the **Reporting** tab in the TSS Environment Variables dialog box.



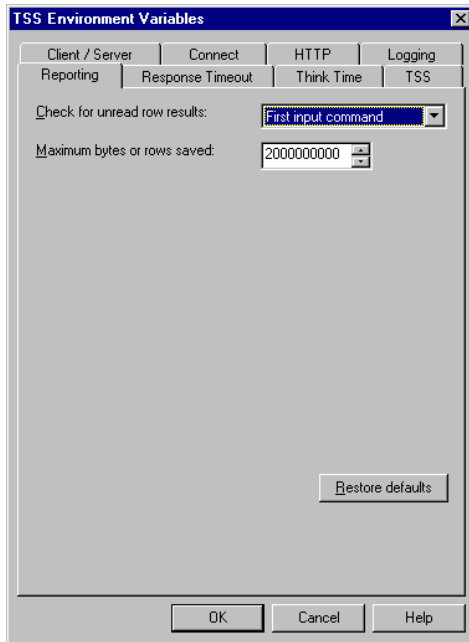
Values for the HTTP reporting environment variables can be initialized as follows:

Variable	Description	Potential values	Default value
CHECK_UNREAD	Check for unread row results	First input command (FIRST_INPUT_CMD) Off (OFF) Every input command (EVERY_INPUT_CMD)	First input command (FIRST_INPUT_CMD)
MAX_NRECV_SAVED	Maximum bytes or rows saved	0-2000000000	2000000000

## Initializing Response Timeout Environment Variables

To initialize response timeout environment variables:

- Choose the **Response Timeout** tab in the TSS Environment Variables dialog box.



Values for the HTTP response timeout environment variables can be initialized as follows:

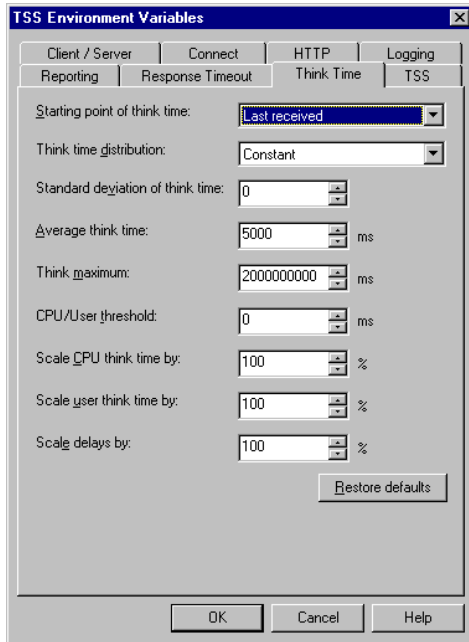
Variable	Description	Potential values	Default value
TIMEOUT_VAL	Timeout	0-2000000000 ms	120000
TIMEOUT_SCALE	Scale timeout by	0-2000000000 ms	100
TIMEOUT_ACT	Timeout action	Ignore (IGNORE) Fatal (FATAL)	Ignore (IGNORE)



## Initializing Think Time Environment Variables

To initialize think time environment variables:

- Choose the **Think Time** tab in the TSS Environment Variables dialog box.



Values for the HTTP think time environment variables can be initialized as follows:

Variable	Description	Potential values	Default value
THINK_DEF	Starting point of think time	First sent (FS) Last sent (LS) First received (FR) Last received (LR) First connection (FC) Last connection (LC)	Last received (LR)
THINK_DIST	Think time distribution	Constant (CONSTANT) Uniform (UNIFORM) Negative exponential (NEGEXP)	Constant (CONSTANT)
THINK_SD	Standard deviation of think time	0-2000000000 ms	0
THINK_AVG	Average think time	0-2000000000 ms	5000
THINK_MAX	Maximum think time	0-2000000000 ms	2000000000

Variable	Description	Potential values	Default value
THINK_CPU_THRESHOLD	CPU/tester threshold	0-2000000000 ms	0
THINK_CPU_DLY_SCALE	Scale CPU think time by	0-2000000000 percent	100
THINK_DLY_SCALE	Scale tester think time by	0-2000000000 percent	100
DELAY_DLY_SCALE	Scale delays by	0-2000000000 percent	100

## Disabling Test Script Services

In some situations you may want to disable use of environment variables completely during a test run. Disabling some test script services allows you to eliminate some runtime overhead during a suite run. You also could create a test script that uses all types of test script services and then, based on what is enabled or disabled here, test only functional or performance related services.

To disable test script services and use of environment variables in certain testing situations:

- Choose the **TSS** tab in the TSS Environment Variables dialog box.



Enable or disable the following services:

- Datapools
- Timers
- Commands
- Think time
- Delays
- Monitoring
- Logging
- Verification points
- Synchronization points
- Shared variables

By default, no test script services are selected as disabled.

### **Changing the Number of Start Test Scripts**

If you are starting virtual testers in groups, TestManager lets you specify the number of test scripts that the group of virtual testers must complete before the next group starts. You may need to do this to control the maximum number of virtual testers that log on to a database server at a given time.

For example, assume the Data Entry user group contains 100 virtual testers. Each virtual tester runs a Login test script and then selects three test scripts in a random order: Add New Record, Modify Record, and Delete Record. You have changed the runtime settings so that the 100 virtual testers are not starting all at once; instead, they are starting in groups of 25.

If you set the number of start scripts to one, the second group of 25 starts when each virtual tester in the first group of 25 completes the Login test script. The third group of 25 starts when each virtual tester in the second group has completed the Login test script, and so on.

To change the number of start test scripts:

- Click **Suite > Edit Settings**.

### **Limiting the Number of Test Scripts**

TestManager allows you to limit the number of test scripts that virtual testers can run without having to remove any test scripts from the user group.

For example, limit the number of test scripts to:

- Test if the virtual tester can complete an initial logon test script. By limiting the number of test scripts, you don't have to delete the remainder of the test scripts assigned to the group and add them later, or create a new suite just to run a simple test.
- Temporarily disable a user or computer group without deleting it from the suite. By setting **Script Limits** to 0 for the group, you disable it. (You can also disable a fixed group by setting the number of virtual testers to 0.)
- Vary the length of a suite run. If your suite contains nested scenarios with varying numbers of test scripts, and you specified random selection of those scenarios, it is very complicated to use repetition counts to vary the length of the suite run. A simpler way is to limit the number of test scripts.

To limit the number of test scripts:

- Click **Suite > Edit Settings**.

### Changing the Way Random Numbers Are Generated

Each virtual tester in a user group has a *user seed*, which generates random numbers in a test script. These random numbers affect a virtual tester's think time, random number library routines, and random access in datapools. Seeds are, primarily, either unique or the same:

- With a *unique* seed, each virtual tester who runs the same test script has a slightly different behavior.

For example, if one virtual tester thinks for 1.3 seconds before executing the first command, the second virtual tester might think for 2.4 seconds. Although the individual think times vary, they have the same distribution around a mean value.

The seeds also affect the library routines involving random numbers. For example, if the first virtual tester calls the a *uniform* routine twice (*uniform* in VU test scripts, *TSSUtility.Uniform* in VB test scripts, or *TSSUtility.uniform* in Java test scripts) and receives the numbers 5 and 3, other virtual testers in that group probably receive different numbers, bounded only by the minimum and maximum values that are set in the test script.

- With the *same* seed, each virtual tester who runs the same test script has exactly the same behavior.

For example:

- If the first virtual tester thinks for 1.3 seconds before executing the first command, the second virtual tester (and all subsequent virtual testers) also thinks for 1.3 seconds before executing that command.
- If the first virtual tester calls the `uniform` routine twice and receives the numbers 5 and 3, all other virtual testers in that group also receives 5 and 3.

You can also set whether or not the random number generator is reseeded at the beginning of each test script. In general, it is better not to reseed, because one long pseudorandom sequence is more realistic than many short ones.

Therefore seeds are defined in one of the following ways:

- *Unique and not reseeded* – Generates a unique seed for each virtual tester and does not reseed the random number generator at the beginning of each test script. Each virtual tester in a user group behaves slightly differently. This is the most commonly used option in performance testing.
- *Unique and reseeded* – Generates a unique seed for each virtual tester and reseeds the random number generator at the beginning of each test script. Each virtual tester in a user group behaves slightly differently, but the numbers are reseeded at the beginning of each test script. This option is very effective for government LTD (Live Test Demonstration) testing.
- *Same and not reseeded* – Generates the same seed for each virtual tester and does not reseed the random number generator at the beginning of each test script. This is generally not a desirable option to select when modeling a realistic workload, because each virtual tester who runs the same test script behaves in the same way. But this option may be useful for certain types of stress testing.
- *Same and reseeded* – Generates the same seed for each virtual tester and reseeds the random number generator at the beginning of each test script. This generally is not a desirable option to select when modeling a realistic workload, because each virtual tester who runs the same test script behaves in the same way, and short pseudorandom sequences are not realistic.

However, this option may be useful for certain types of stress testing. For example, if you have a suite with a shared datapool with the seed set as unique and not reseeded, each virtual tester and iteration has a different seed that gives random data across all virtual testers and all iterations. To see what happens when all virtual testers access the same data pattern over and over again, set the seed as same and reseeded for all virtual testers.

To change the behavior of the default random number generator:

- Click **Suite > Edit Settings**.

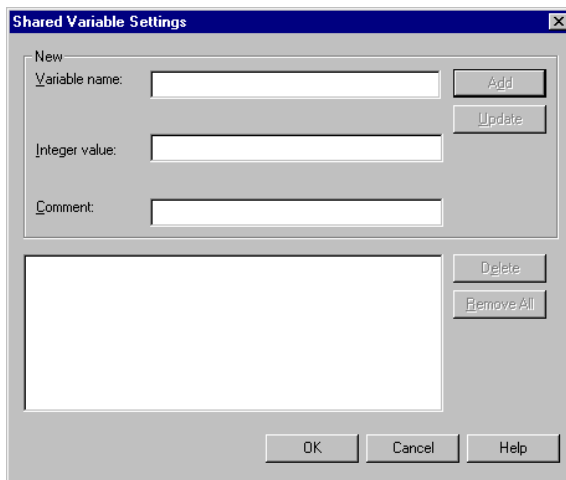
**Note:** The library routines that generate random numbers are `negexp`, `rand`, and `uniform` in VU test scripts, `TSSUtility.NegExp`, `TSSUtility.Rand`, and `TSSUtility.Uniform` in VB test scripts, and `TSSUtility.negExp`, `TSSUtility.rand`, and `TSSUtility.uniform` in Java test scripts. For information about these routines in VU test scripts, see the *VU Language Reference*. For information about these routines in other script types, see the appropriate API documentation.

## Setting Shared Variables

TestManager allows you to initialize shared variables in a suite. A shared variable maintains its value across all test scripts in a suite. Each virtual tester can access and change the shared variable.

To initialize a shared variable:

- Click the **Suite > Edit Shared Variables**.



Shared variables are useful in the following situations:

- For synchronizing virtual testers when you need more specific coordination than a synchronization point provides. For example, you can limit a transaction so that only five virtual testers perform it at once. In that case, use a shared variable with a wait routine (`wait` in VU test scripts, `TSSSync.SharedVarWait` in VB test scripts, and `TSSSync.sharedVarWait` in Java test scripts).

- For blocking a virtual tester from executing until a global event occurs. It is easier to set an event and a dependency than to set a shared variable. However, if the event depends on some logic *within* a test script, you must use a shared variable.
- For counting loops within a test script. If you want to set a loop for an entire test script, it is easier to set a selector or an iteration count within the suite. However, if only a portion of the test script loops, set a shared variable to control the number of iterations of that loop.
- For monitoring specific transaction counts and conditions. You can view shared variables when you monitor a suite, and they provide detailed information about the progress or state of a suite run.

You *declare* a shared variable within a test script or resource file with a `shared` keyword (`shared` in VU test scripts, `TSSUtility.SharedVarAssign` for VB resource files, and `TSSUtility.sharedVarAssign` for Java resource files). For more information about declaring shared variables, see the *VU Language Reference*, or the appropriate Rational Test Script Services API manual.

You *initialize* a shared variable within a suite. This is optional—the default value is 0.

You *manipulate* the value of a shared variable through the logic in a test script or when you monitor the suite.

## Printing and Exporting a Suite

Designing a suite can involve many iterations and changes. You may find it helpful to examine a printed view of a suite. You can print a suite or export it as a .txt file.

To print a suite, from the open suite:

- Click **File > Print**.

To export a suite as a .txt file, from the open suite:

- Click **File > Export**.

## Saving a Suite

After you have finished modifying a suite, save your changes. A suite that is not saved has an asterisk in the title bar.

To save a suite:

- Click **File > Save**.

To save more than one suite:

- Click **File > Save All**.

**Note:** If you click **Tools > Options**, click the **Create Suite** tab, then select the **Check suite when saving** box, one verification screen appears for each suite being saved.

To save a suite under a different name:

- Click **File > Save As**.



This chapter describes how to run tests. It includes the following topics:

- About running tests
- Built-in support for running test scripts
- Running automated test scripts
- Running manual test scripts
- Running test cases
- Running suites
- Monitoring suites

**Note:** For detailed procedures, see the TestManager Help.

## About Running Tests

---

The activity of running your tests is primarily running the implementations of test cases to make sure that the system functions correctly.

In TestManager, you can run:

- Automated test scripts.
- Manual test scripts.
- Test cases.
- Suites.

## Built-in Support for Running Test Scripts

---

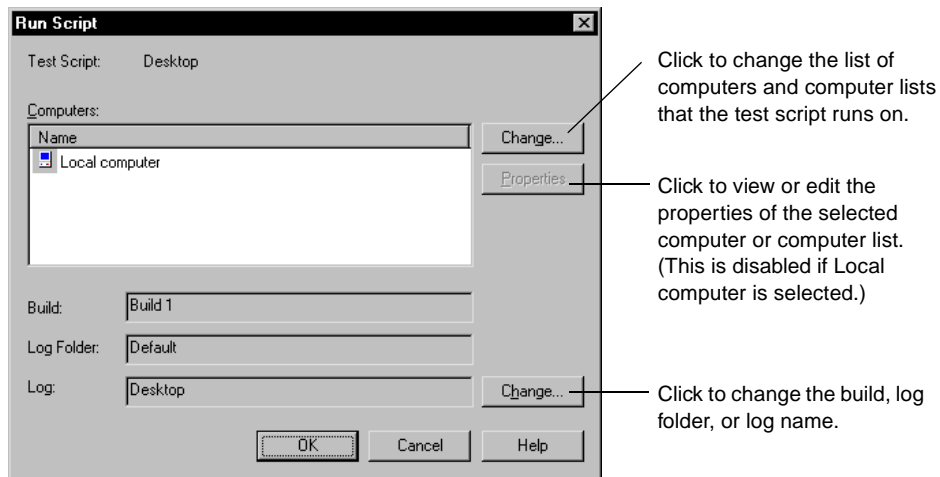
TestManager provides built-in support for running the following test script types:

Test Script Type	Description
GUI	A functional test script written in SQABasic, a Rational proprietary Basic-like scripting language.
VU	A performance test script written in VU, a Rational proprietary C-like scripting language.
Manual	A set of testing instructions to be run by a human tester.
Visual Basic	A test script written in the Visual Basic language.
Java	A test script written in the Java language.
Command line	A test script (for example, an .exe file, a .bat file, or a UNIX shell script) that can be executed from the command line.

## Running Automated Test Scripts

To run an automated test script:

- 1 Click **File > Run Test Script**, and select the test script type.
- 2 Select the test script to run.



When you click **OK**, TestManager runs the test script on the first available computer in the list. As the test script runs, you can monitor its progress and then view the results in the test log.

For information about computers and computer lists, see *Defining Computers and Computer Lists* on page 64.

For information about monitoring progress, see *Monitoring Suites* on page 102.

For information about test logs, see *About Test Logs* on page 137.

## Running Manual Test Scripts

---

If you used the grid or text editor when you created a manual test script, you can do the following when you run the script:

- Optionally, set the run option to log unchecked steps as warnings.
- Indicate that you have performed each step.
- Indicate whether each verification point passed or failed.

If you included an external file when you created the manual test script, you can do the following after you run the script:

- Indicate whether the entire script passed or failed.

**Note:** For information about creating manual test scripts, see *Creating Manual Test Scripts* on page 56.

To run a manual test script, do one of the following:

- In TestManager, click **File > Run Test Script > Manual** and select a test script. Rational ManualTest opens.
- In Rational ManualTest, click **File > Run** and select a test script.

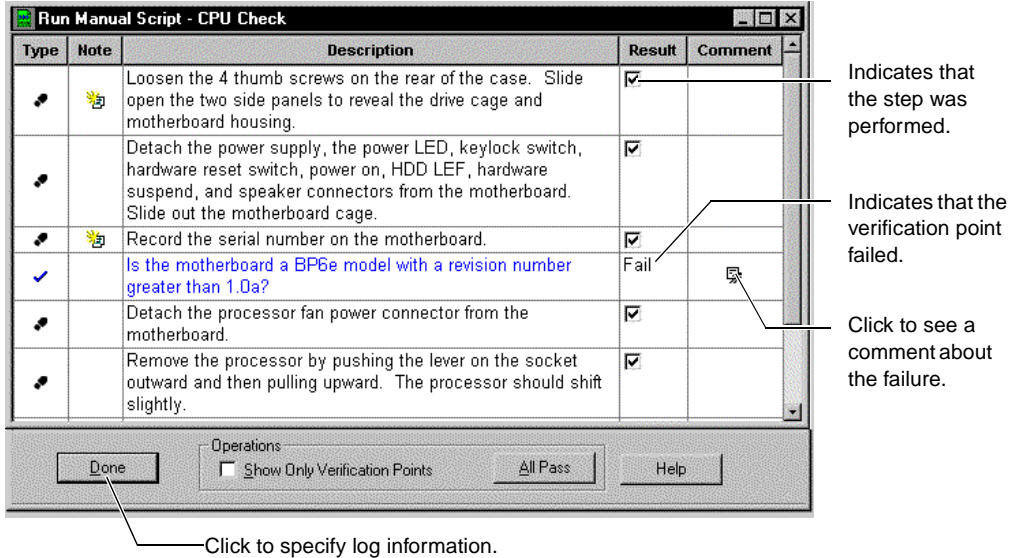
If you used the grid or text editor to create the manual script, perform each step and verification point listed in the Run Manual Script window:

- For a step, select the Result check box to indicate that you have performed the step.
- For a verification point, click the Result cell and click None, Pass, or Fail.

If you included an external file when you created the manual script, follow the instructions in the file. Fill in the Enter Results of the Manual Script dialog box with the results of the entire script.

## Example of Running a Manual Test Script

The following figure shows the results of running a manual test script. When this manual test script was run, the first verification point failed. The Comment icon indicates that there is a comment about the failure. When you view the log, you will be able to see the failure and the comment.



## Viewing the Results of Running a Manual Test Script

After you run a manual test script, you can view the results in the test log.

To view the results of running a manual test script:

- 1 Return focus to TestManager by clicking **Tools > Rational TestManager**.
- 2 Click **File > Open Test Log**.

## Running Test Cases

---

When you run a test case, you are actually running the implementation of the test case. The implementation is a test script or suite that is associated with the test case.

## Viewing the Associated Implementations

To view or change the implementations that are associated with a test case:

- 1 Right-click a test case in the Test Plan window and click **Properties**.
- 2 Click the **Implementation** tab.

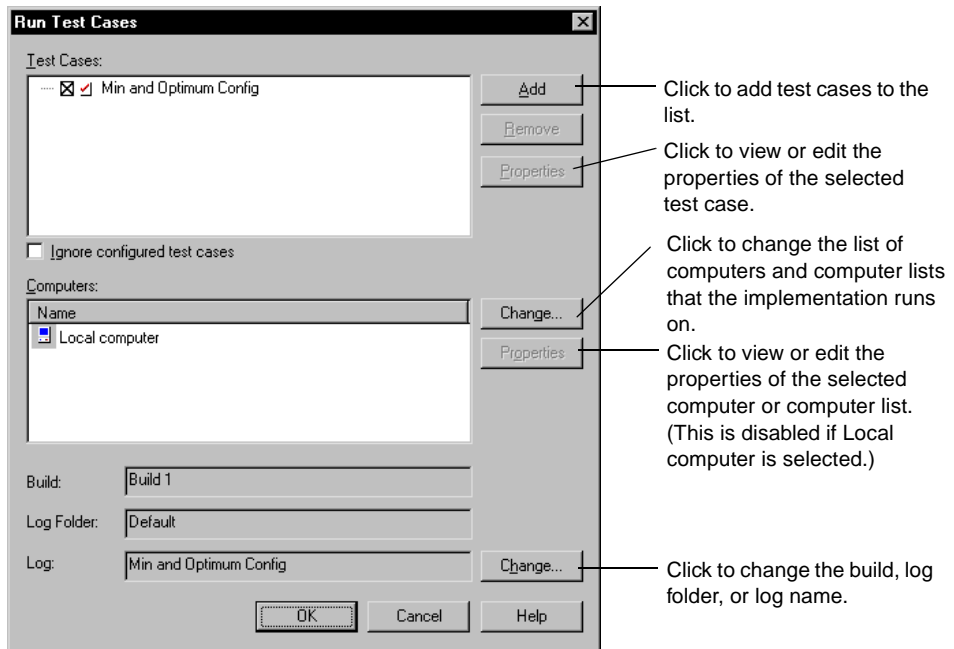
You can have at most two implementations associated with a test script: one manual and one automated. If both are associated with a test case, TestManager will run the automated implementation when you run the test case.

For more information about implementing test cases and associating implementations, see *Implementing Tests* on page 53.

## Running a Test Case

To run a test case and its implementation:

- 1 Click **File > Run Test Case**.
- 2 Select the test case or configured test case to run and click **OK**.



When you click **OK**, TestManager runs the test case as follows:

- If you run a test case that has an automated implementation, it runs on the available computers in the list.

If you run a configured test case, it runs on the computers in the list that match the values of the configuration attributes. For example, if the configured test case indicates that the test case should run on a computer with Windows 2000, TestManager examines each computer in the list until it finds one that has Windows 2000, and then runs the test case on that computer. If no computers in the list match the configuration, a message appears in the test log.

As the test case runs, you can monitor its progress and then view the results in the test log. For information about monitoring progress, see *Monitoring Suites* on page 102. For information about test logs, see *About Test Logs* on page 137.

- If you run a test case that has a manual implementation and no automated implementation, it starts Rational ManualTest. You can perform the steps and verification points in the manual test script, and then view the results in the test log. For more information, see *Running Manual Test Scripts* on page 91.

## Ignoring Configured Test Cases

Select the **Ignore configurations for test cases** check box to have TestManager ignore system configurations for test cases, and to run the test cases on any available computers.

TestManager has three ways of running test cases if this option is selected:

- If a test case has configured test cases, and the parent test case has an implementation (for example, a test script or suite), TestManager runs the parent test case on any available computer, but does not run any of the configured test cases.
- If a test case has configured test cases, and the parent test case does not have an implementation, TestManager does not run the parent or any configured test cases.
- If a single configured test case has an implementation, TestManager runs the test case on the specified computer.

## Running Suites

---

If you implemented a test as a suite, for either a functional or performance test, there are a number of steps involved with actually running the suite. They are:

- Checking the suite.
- Checking Agent computers.
- Controlling runtime information of the suite.

- Controlling how the suite terminates.
- Specifying virtual testers and configurations for the suite run.
- Stopping the suite.

For information on creating and running a functional testing suite, see Chapter 8, *Creating Functional Testing Suites*.

For information on creating and running a performance testing suite, see chapter 11, *Creating Performance Testing Suites*.

## Checking a Suite

While you are working on a suite, you might change it so that it does not run correctly. For example, you might insert a test script into a suite before it is recorded. Although TestManager automatically checks a suite before it runs, you can check a suite without actually running it at any time. This can help you identify and correct problems.

To check a suite:

- Click **Suite > Check Suite**.

TestManager checks a suite for many kinds of errors, including the following:

- The suite does not contain any user or computer groups.  
A suite must have at least one user or computer group to run.
- The suite contains an empty user or computer group.  
Either delete the user or computer group or add test scripts and other items to it.
- A user or computer group contains an empty scenario.  
Either delete the scenario or add items to it.
- The suite contains a selector that is empty.  
Either delete this selector or add properties to it.

**Note:** You can set options so that the suite is checked automatically whenever you save it. To check the suite automatically, click **Tools > Options**, click the **Create Suite** tab, and then select the **Check suite when saving** check box.

## Checking Agent Computers

If you are running virtual testers on Agent computers, it is a good idea to check the Agents before you run the suite. This way, you can determine whether any problems exist before you run the suite.

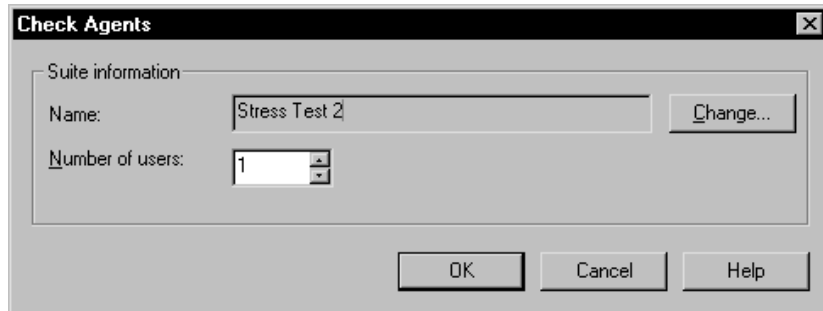
When you check Agent computers, TestManager ensures that:

- All of the Agent computers specified for virtual testers actually exist.  
For example, if you incorrectly typed the name of an Agent computer, TestManager notifies you.
- The Agent computers are available and running.
- The Agent software is running.

The same release of TestManager software must be installed on both the Local and the Agent computers.

To check the Agent computers:

- Click **Suite > Check Agents**.



TestManager displays any problems with the Agent computers in a separate window.

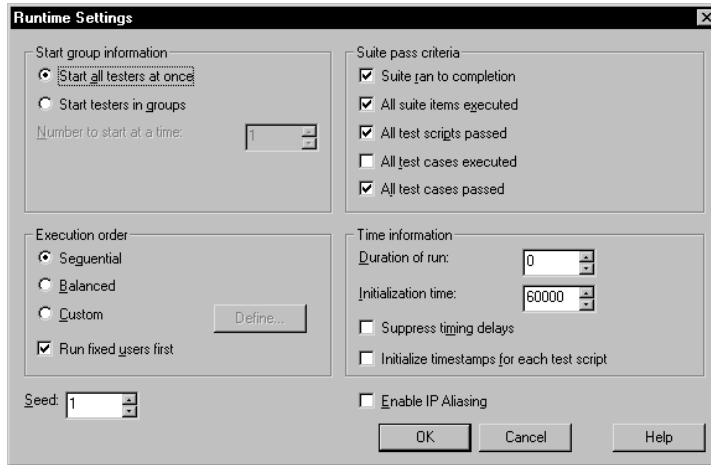
## Controlling Runtime Information of a Suite

TestManager lets you control the way a suite runs.

To set the runtime settings for a suite:



- Click **Suite > Edit Runtime**.



By modifying the runtime settings, you can manage:

- How virtual testers are started—either all at once or in groups.

Avoid overloading a server by choosing to start virtual testers in groups, and specifying the number of virtual testers in those start groups.

**Note:** If you start virtual testers in groups, you should also specify the number of start scripts for the group. To do this, click **Suite > Edit Settings**, and modify the **Start scripts** box.

- The criteria for whether a suite passes or fails.
  - *Suite ran to completion* – The suite ran to completion without manual termination of the run.
  - *All suite items executed* – All items in the suite were able to complete all of their assigned tasks.
  - *All test scripts passed* – All test scripts passed, which means that no events failed and no commands timed out.
  - *All test cases executed* – All test cases in the suite were able to complete all of their assigned tasks.
  - *All test cases passed* – All test cases passed, which means that the application being tested met the goals of the given test case.

If the suite does not meet the criteria, the Test Log window lists the Suite Start and Suite End events as “failed.”

- The order in which the user or computer groups run.

The execution order defines the order in which virtual testers are started, and therefore determines which user groups are executed if you run fewer virtual testers than the maximum number defined. Select one of the following:

- Suites that run in a *sequential* order run each virtual tester as it is encountered in the suite (from the top to the bottom).
- Suites that run in a *balanced* manner evenly distribute the run among the user groups in proportion to the suite.
- Suites that run in a *custom* order require you to select specific user groups or virtual testers to run. Apply a custom run order to fixed user groups only.

Running user groups in a custom order is useful for troubleshooting. For example, if a test script does not work, and that test script is used only by the Accounting group, run that group only.

To create a custom run order, click **Define** in the Runtime Settings dialog box.

**Note:** You can temporarily disable a fixed user group by setting the number of virtual testers to 0.

You can run fixed virtual testers first, thus running fixed user groups before the scalable user groups, regardless of the execution order. If your user groups are all fixed, specifying a run order has no effect.

- Timing information such as:
  - The maximum amount of time the run should take. A value of 0 imposes no time limit.
  - The maximum number of seconds for all virtual testers to confirm that they completed initializing. If you have changed the number of start test scripts, make sure that you set this time high enough.
  - Choosing to suppress timing delays, thus running the suite very quickly because all timing delays in test scripts are suppressed. This choice is useful if you are testing a suite to see whether it runs correctly and you are not interested in the timing delays. Note, however, that this creates a maximum workload on the server, Local, and Agent computers.

Do not suppress timing delays if you are running a large number of virtual testers.

- Choosing to initialize timestamps for each test script, which indicates whether timestamps are carried over from test script to test script or are reinitialized with each test script.
- The number to feed to the random number generator.

TestManager uses a specified seed to generate the random numbers for selectors and shared access in datapools.

- Whether to use IP aliasing.

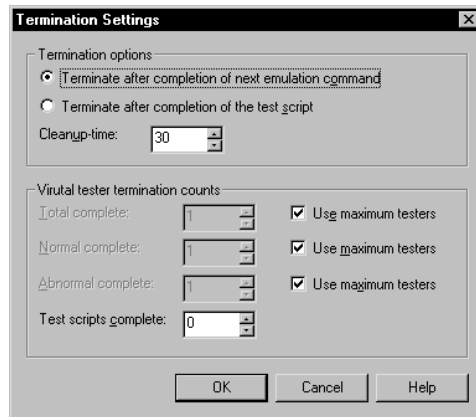
IP aliasing requires that each virtual tester has a different source IP address. This has meaning only if you are running HTTP test scripts, and your system administrator has set up your computer to use IP aliasing. For information on setting up IP Aliasing, see Appendix A, *Configuring Local and Agent Computers*.

## Controlling How a Suite Terminates

TestManager lets you set the conditions that force a suite to stop running. For example, you may want to stop a suite if you discover that a large number of virtual testers are completing abnormally, indicating that something is wrong with the run.

To control how a suite terminates:

- Click **Suite > Edit Termination**.



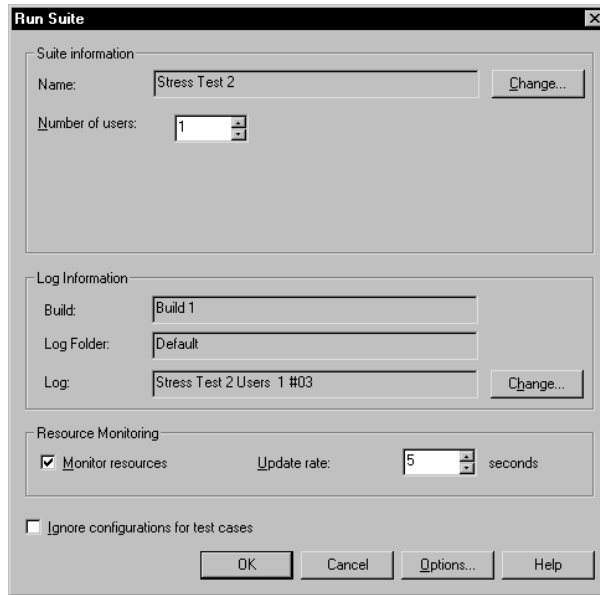
## Running a Suite

When you run a suite, you supply runtime-specific information. Each virtual tester that executes its assigned suite items run within these guidelines. The results of running the suite are stored in *logs*.

After you run the suite, run reports to analyze the data stored in the logs. You can display this information in the form of graphs and charts, or in more traditional report formats.

To run a suite:

- Click **File > Run Suite**.



TestManager checks the suite, and compiles any uncompiled or out-of-date test scripts.

When you run a suite, you specify:

- Suite name.
- Number of virtual testers, if you are running a performance test.

If a suite includes both fixed and scalable user groups, the fixed user groups are assigned first. So, for example, if your suite includes one user group fixed at 10 virtual testers, and you run 100 virtual testers, 10 virtual testers are assigned to the fixed user group, and the remaining 90 virtual testers are distributed among the scalable user groups.

**Note:** The number of available virtual testers depends on the type of license you have. If your license does not support the number of virtual testers you specify, you see an error message.

- Number of computers on which to run the suite and a list of available computers, if applicable.

If you did not specify the computers on which to run the suite when you added a computer group, you must specify computers now.

- Log information, including build number, log folder, and log file name.

By default, the name of the log folder is based on the suite, and the log name is based on the number of virtual testers and the number of times you have run the suite. For example, if you run the sample suite three times, with 10 virtual testers, 15 virtual testers, and 20 virtual testers, all three logs will be in the sample suite folder. The log names will be Users 10 #01, Users 15 #02, and Users 20 #03. Therefore, the log name Users 20 #03 indicates that this is the third time you have run the suite, and the suite is being run with 20 virtual testers.

You can change these settings on the **Run** tab of the Options dialog box. For more information, see the TestManager Help.

- Resource monitoring.

Monitoring observes computer resource usage when you play back the suite and then graphs this usage data over the corresponding virtual tester response times when you analyze your results. Specify the interval at which the views are updated; the lower the interval, the faster the update.

- Whether to ignore associated configurations.

Select the **ignore configurations for test cases** check box to have TestManager ignore system configurations for test cases, and to run the test cases on any available computers.

TestManager has three ways of running suites if this option is selected:

- If a suite contains a test case with configured test cases, and the parent test case has an implementation (for example, a test script or suite), TestManager runs the parent test case on any available computer, but does not run any of the configured test cases.
- If a suite contains a test case with configured test cases, and the parent test case does not have an implementation, TestManager does not run the parent or any configured test cases.
- If a single configured test case has an implementation, TestManager runs the test case on the specified computer.

## Stopping the Suite

To cancel the suite run while TestManager is checking the suite:

- Click **Cancel**.

To stop a suite after TestManager checks it, compiles test scripts, and begins monitoring the suite:

- Click **Monitor > Stop**.

When the suite completes the test run—whether it completes normally or you manually terminated it—TestManager displays default log data in the Test Log window if log information is available. For more information on the Test Log window, see Chapter 6, *Evaluating Tests*.

## Monitoring Suites

---

This section discusses how to monitor a suite. It includes the following topics:

- About monitoring suites
- Displaying the Suite views
- Displaying a histogram
- Displaying the User/Computer views
- Displaying the Shared Variables view
- Displaying the Script view
- Displaying the Sync Points view
- Displaying the Computer view
- Displaying the Transactor view
- Displaying the Group views
- Filtering and sorting views
- Changing shared variable information manually
- Debugging a test script
- Changing monitor defaults
- Controlling the suite during a run

**Note:** This section focuses on monitoring suite runs. However, you can also monitor test case and test script runs in much the same way.

### About Monitoring Suites

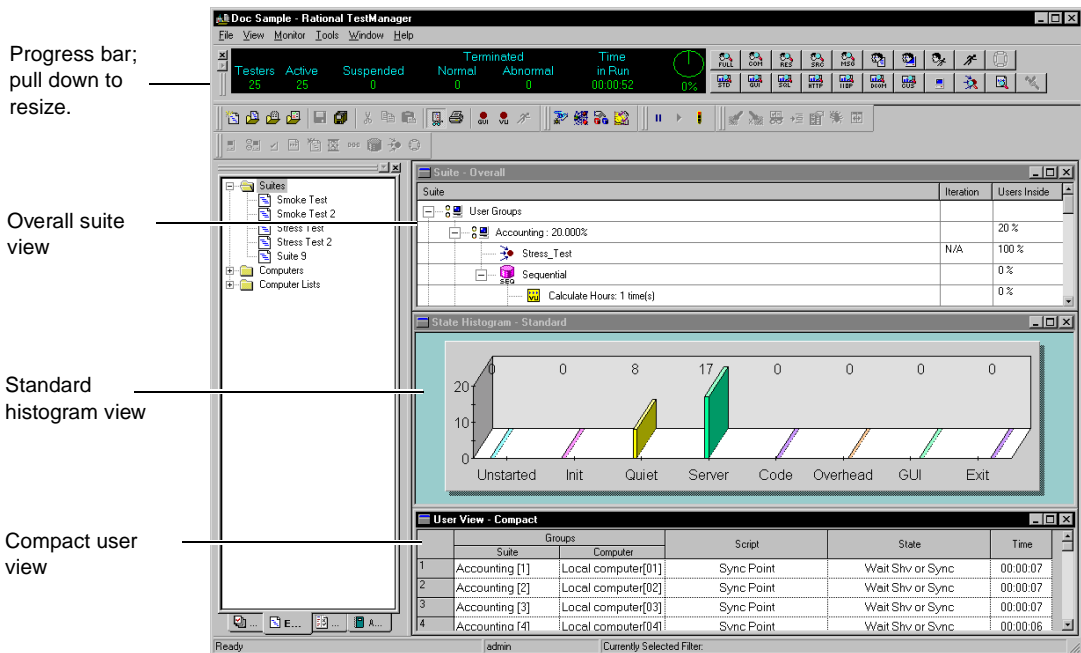
While a suite is running, you may want to monitor its progress. Monitoring a suite lets you not only confirm that a suite is progressing successfully, but also lets you discover potential problems early in the run so you can intervene if necessary. You can suspend and restart virtual testers, change the values of shared variables, and release virtual testers waiting on synchronization points.

TestManager's monitoring tools provide you with up-to-date information that is dynamically updated as the suite runs. This information includes:

- The number of commands that have executed successfully and the number of commands that have failed.
- The general state of the virtual testers: whether they are initializing, connecting to a database, exiting the suite, or performing other tasks.
- Whether any virtual testers have terminated abnormally.

When you run a suite, TestManager displays the monitoring information in a Progress bar and in views. The Progress bar gives you a quick summary of the state of the run and cannot be changed. You can change the views, however, to provide summary information or detailed information about each virtual tester.

The following figure shows the Progress bar and the default views:



The Progress bar lets you quickly assess how successfully the suite is running. The Progress bar provides the following information:

- **Testers** – The total number of virtual testers in the run.
- **Active** – The number of virtual testers that are neither suspended nor terminated.
- **Suspended** – The number of virtual testers in a paused state.

- **Terminated: Normal** – The number of virtual testers that completed their tasks successfully.
- **Terminated: Abnormal** – The number of virtual testers that terminated without completing all of their assigned tasks.
- **Time in Run** – The time the suite has been running, expressed in *hours:minutes:seconds*.
- **%Done** – The approximate percentage of the suite that has completed.

TestManager also displays three views of the running suite:

- The Overall Suite view, which displays general information about the status of virtual testers. For more information, see *Displaying the Suite Views* on page 104.
- The Standard Histogram view, which is a bar chart that provides a general idea of what tasks the virtual testers are performing. For example, some virtual testers might be initializing, some virtual testers might be executing code, and some virtual testers might be connecting to the database. This chart shows the number of virtual testers in each state.

TestManager displays the Standard Histogram view by default. However, if your virtual testers are running GUI test scripts, or if you are testing a SQL database or Web performance, you may want to display a bar chart specifically geared to those tests. For more information, see *Displaying the Histogram Views* on page 107.

- The Compact User view, which displays information about the current state of the virtual testers. In this view, you can click on a particular virtual tester to display additional information about that virtual tester or control its operation. For more information, see *Displaying the User/Computer Views* on page 114.

## Displaying the Suite Views

The suite views are very similar to the actual suite that you have designed. Columns show you which iteration is being executed and what percentage of the virtual testers in a group are currently in a test script or a selector.

The two suite views are:

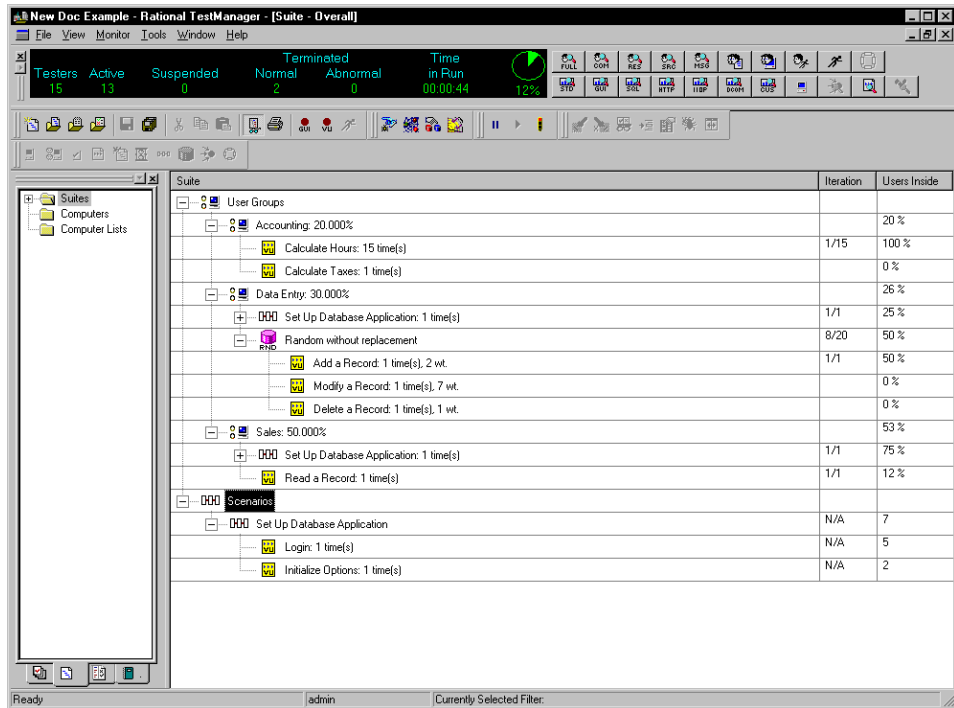
- **Overall** – Use this view to display general information about the status of the suite. TestManager displays this view by default when you run a suite.
- **User/Computer** – Use this view to display the exact suite progress of a particular virtual tester.



## The Overall Suite View

To display the overall suite view:

- Click **Monitor > Suite > Overall**.



The Overall suite view is similar to the actual suite that you have designed. However, it contains two additional columns:

The **Iteration** column shows how many iterations are in the suite item and the iteration in progress, averaged over all virtual testers currently executing that suite item.

For example, 8/20 indicates that, for the virtual testers currently executing that suite item, on the average, the virtual testers are executing the 8th of 20 total iterations.

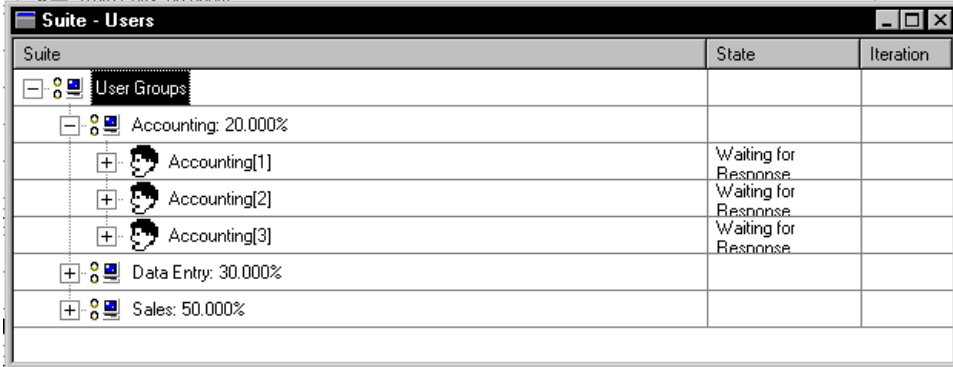
The **Users Inside** column shows the percentage of virtual testers that are currently executing each portion of the suite. The percentage next to the user group shows the percentage of total virtual testers that have been assigned to the group and have not yet exited the suite. The percentage next to the items within a user group shows the percentage of virtual testers within that group that are executing that item.

For example, if the Sales user group contains 50 percent of the total virtual testers, then the Users Inside column for that group is 50 percent. If all virtual testers in the Sales group are executing the Read Record test script, then the Users Inside column for that test script is 100 percent.

### The User/Computer View

To display the User/Computer Suite view:

- Click **Monitor > Suite > Computer** or **Monitor > Suite > User**.



The screenshot shows a window titled "Suite - Users" with a table view. The table has three columns: "Suite", "State", and "Iteration". The "Suite" column contains a tree view of user groups. The "State" column shows the status of each user group, and the "Iteration" column is empty.

Suite	State	Iteration
[-] User Groups		
[-] Accounting: 20.000%		
[+] Accounting[1]	Waiting for Response	
[+] Accounting[2]	Waiting for Response	
[+] Accounting[3]	Waiting for Response	
[+] Data Entry: 30.000%		
[+] Sales: 50.000%		

**Note:** Whether the Computer Suite view or the User Suite view is available depends on whether you are monitoring a computer group-based suite for functional testing or a user group-based suite for performance testing.

## Displaying the Histogram Views

The histogram views group the virtual testers into various states, such as exiting and initializing. Use a histogram view to display a bar graph of how many virtual testers are in each state.

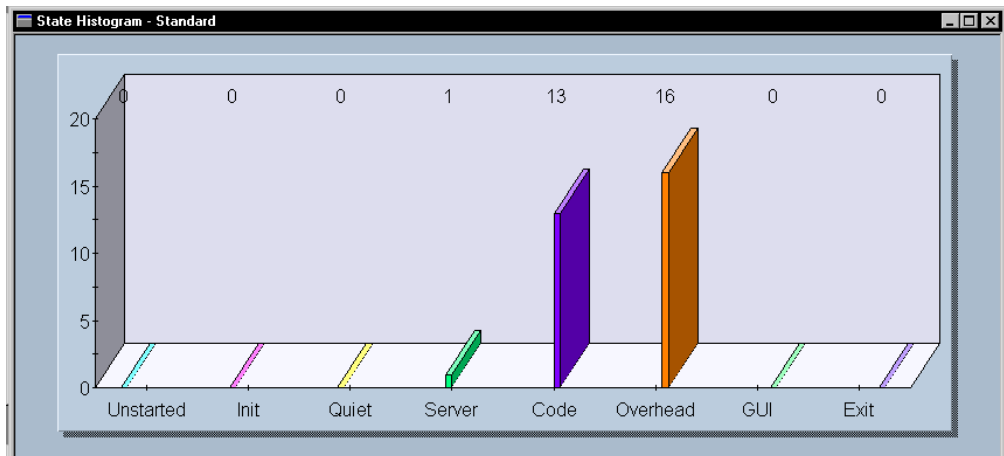
To display a histogram view:

- Click **Monitor > Histogram**, and select the desired histogram.

The histogram views are:

- **Standard** – Data is grouped in a general way. Select this histogram if you want a general overview of the virtual tester states.
- **GUI** – Data is grouped appropriately for tests that run GUI test scripts.
- **SQL** – Data is grouped appropriately for tests that access SQL databases.
- **HTTP** – Data is grouped appropriately for tests that access Web servers.
- **IIOP** – Data is grouped appropriately for tests that access IIOP servers.
- **DCOM** – Data is grouped appropriately for the tests that access DCOM functions.
- **Custom** – Data is grouped according to your needs. For information about customizing a histogram, see *Configuring Custom Histograms* on page 132.

The following figure shows a Standard histogram:



In this histogram, one virtual tester is in the Server state, 13 virtual testers are in the Code state, and 16 virtual testers are in the Overhead state.

The following sections describe the different types of histograms that you can display.

## Standard Histograms

In a Standard histogram, which is displayed by default, data is grouped in a general way. The following table describes the bars in a Standard histogram:

Bar Name	Description
Unstarted	The process associated with the virtual tester has not started. If you see this state for a while, you have probably started virtual testers and test scripts in groups. Until a group completes initialization, subsequent groups are in this state.
Init	Virtual testers that are initializing.
Quiet	Virtual testers that are thinking, delaying, or suspended, or are waiting on a shared variable or synchronization point.
Server	States related to communicating with the server.
Code	Virtual testers that are executing code such as VU, VB, Java, Manual, or external C, or executing a <code>testcase</code> or <code>emulate</code> command. This state does not include SQABasic code.
Overhead	States related to run overhead. These states are <code>GetTask</code> (getting the next suite task), <code>InitScript</code> (initializing a test script), <code>Match</code> (pattern matching), and <code>Read_Shv</code> (reading a shared variable over the network).
GUI	Virtual testers performing GUI-related operations.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

## GUI Histograms

A GUI histogram displays information pertinent to tests that run GUI test scripts. The following table describes the bars in a GUI histogram:

Bar Name	Description
Init	Virtual testers that are initializing.
Input	Virtual testers that are typing input.
WaitApp	Virtual testers that are waiting on an application output.
Quiet	Virtual testers that are delayed.
Code	Virtual testers that are executing code such as VU, VB, Java, Manual, or external C, or executing a <code>testcase</code> or <code>emulate</code> command. This state does not include SQABasic code.

Bar Name	Description
Overhead	States related to run overhead. These states are GetTask (getting the next suite task) and InitScript (initializing a test script).
Other	All other states.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

## SQL Histograms

A SQL histogram displays information pertinent to tests that access SQL databases. The following table describes the bars in a SQL histogram:

Bar Name	Description
Init	Virtual testers that are initializing.
Connect	Virtual testers that are waiting to connect to the database server.
Exec	Virtual testers that are executing VU <code>sqlprepare</code> or <code>sqlexec</code> statements.
Query	Virtual testers that are waiting on the results of a query.
Quiet	Virtual testers that are thinking, delaying, or suspended, or are waiting on a shared variable or synchronization point.
GUI	Virtual testers that are performing GUI-related operations.
Other	All other states.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

## HTTP Histograms

An HTTP histogram displays information pertinent to tests that access Web servers. The following table describes the bars in an HTTP histogram:

Bar Name	Description
Init	Virtual testers that are initializing.
Connect	Virtual testers that are waiting to connect to the Web server.
Send	Virtual testers that are sending data to the Web server.

Bar Name	Description
Recv	Virtual testers that are waiting for data from the Web server.
Linespeed	Virtual testers that are being artificially delayed to achieve a specific network linespeed.
Quiet	Virtual testers that are thinking, delaying, or suspended, or are waiting on a shared variable or synchronization point.
GUI	Virtual testers that are performing GUI-related operations.
Other	All other states.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

## IIOp Histograms

An IIOp histogram displays information pertinent to tests that access IIOp. The following table describes the bars in an IIOp histogram:

Bar Name	Description
Init	Virtual testers that are initializing.
Bind	Virtual testers that are obtaining an object reference.
Connect	Virtual testers that are waiting to connect to the server.
Invoke	Virtual testers that are invoking a remote operation.
Quiet	Virtual testers that are thinking, delaying, or suspended, or are waiting on a shared variable or synchronization point.
GUI	Virtual testers that are performing GUI-related operations.
Other	All other states.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

## DCOM Histograms

A DCOM histogram displays information pertinent to tests that access DCOM protocol. The following table describes the bars in a DCOM histogram:

Bar Name	Description
Init	Virtual testers that are initializing.
Create Object	Virtual testers that are creating a distributed object.
Connect	Virtual testers that are waiting to connect to the server.
Call Method	Virtual testers that are making object method calls.
Quiet	Virtual testers that are thinking, delaying, or suspended, or are waiting on a shared variable or synchronization point.
GUI	Virtual testers that are performing GUI-related operations.
Other	All other states.
Exit	Virtual testers that have finished the suite, with either normal or abnormal termination.

### Zooming In on Histogram Bars

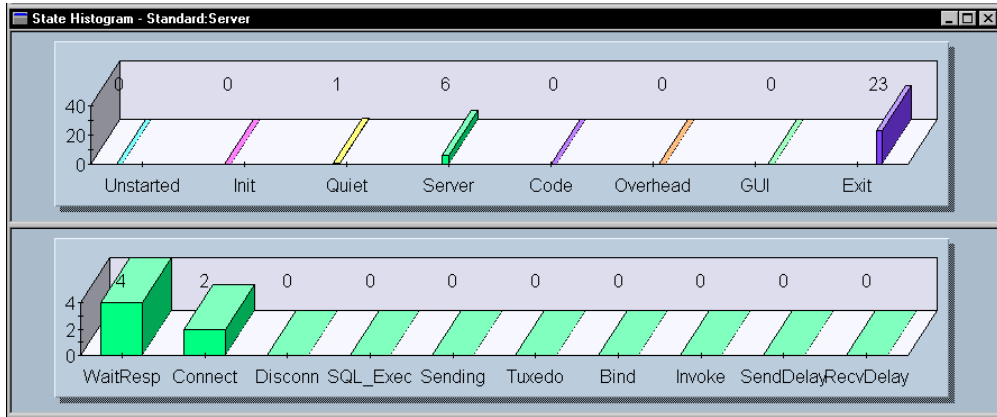
Each bar in a histogram shows a summary state that contains individual states. You can zoom in on a bar to see a breakdown of how many virtual testers are in each state.

To zoom in on a histogram bar:

- Double-click on a bar that contains virtual testers. A window appears that displays the individual states.

**Note:** To restore the window to its original state, click **View > Reset**.

The following figure shows an expanded histogram, after you have clicked on the Server bar:



Six virtual testers are classified as Server. The expanded histogram shows that four are in the WaitResp state and two are in the Connect state.

The individual states in the histogram bars are as follows:

This bar	Indicates that
Bind	Virtual testers that are obtaining an object reference.
Call Method	Virtual testers that are making object method calls.
Cleanup	A virtual tester is cleaning up (process cleanup) before exiting.
CPU_Delay	A virtual tester is emulating a CPU (client application) delay before submitting a command to the server.
Connect	A virtual tester is executing a connect emulation function.
Create Object	A virtual testers is creating a distributed object.
Disconn	A virtual tester is executing a disconnect emulation function. However, the client has not yet disconnected.
Delay	A virtual tester is executing the TSSDelay function. For VU, the routine is <code>delay</code> , for SQABasic the command is <code>DelayFor()</code> , for VB the method is <code>TSSUtility.Delay</code> , and for Java the method is <code>TSSUtility.delay</code> .
Exit	A virtual tester has exited.
ExitSQABasic	A virtual tester has finished executing an SQABasic test script.



<b>This bar</b>	<b>Indicates that</b>
ExternC	A virtual tester is executing an external C routine called from within the test script.
GetTask	A virtual tester is waiting to be assigned its next suite task.
Keys	A virtual tester running an SQABasic test script is keystroking.
Init	A virtual tester is being initialized (process initialization).
InitScript	A virtual tester is preparing to execute a test script.
Invoke	A virtual tester is invoking a remote operation.
Match	A virtual tester is executing <code>tux_precv</code> pattern-matching code while the virtual tester is waiting for completion of the server's response.
Read_Shv	A virtual tester is running on an Agent computer and reading a shared variable from the Local computer.
RecvDelay	A virtual tester encountered a linespeed delay when receiving.
ScriptCode	A virtual tester is executing code unrelated to emulation commands. For example, the virtual tester may be accessing a datapool or performing logic that you added to the test script.
SendDelay	A virtual tester encountered a linespeed delay when sending.
Sending	A virtual tester is sending data with an <code>http_request</code> or <code>sock_send</code> emulation command.
ShvBlock	A virtual tester is temporarily blocked while trying to obtain exclusive access to change the value of a shared variable.
SQABasic	A virtual tester is executing SQABasic code.
SQL_Exec	A virtual tester is executing or preparing a SQL command ( <code>sqlexec</code> or <code>sqlprepare</code> ) and waiting for the server to complete the operation.
StartApp	A virtual tester is starting an application within a SQABasic test script.
SuiteDelay	A virtual tester is executing a delay that you set in the suite.
Suspend	A virtual tester has been suspended.
TestCase	A virtual tester is executing a <code>testcase</code> or <code>emulate</code> command.
Think	A virtual tester is thinking before submitting a command to the server.

This bar	Indicates that
TransDelay	A virtual tester in a transactor is delayed waiting for the next transaction.
Tuxedo	A virtual tester is executing a TUXEDO emulation command and waiting for the server to complete the operation.
Unstarted	The process associated with the virtual tester has not started. If you see this state for a while, you have probably started virtual testers in groups. Until a group completes initialization, subsequent groups are in this state.
VB	A virtual tester is executing Visual Basic code.
WaitResp	A virtual tester is waiting for completion of the server's response (a receive emulation command).
WaitShvSync	A virtual tester is waiting on an event in a suite, waiting to be released from a synchronization point, or executing the wait routine, but the event has not yet occurred.
WaitObj	A virtual tester running an SQABasic script is waiting for a window or control to appear.

## Displaying the User/Computer Views

The User/Computer views dynamically display the status and details of virtual tester operations depending on the type of user the virtual tester is emulating. Display one of the views to see the status of individual virtual testers.

To display a view:

- Click **Monitor > User** or **Monitor > Computer**, and select a view.

The User/Computer views are:

- **Full** – Contains complete information about all virtual testers.
- **Compact** – Contains summary information about all virtual testers. This is the most efficient view to use when you are running Agent computers.
- **Results** – Contains information about the success and failure rate of each emulation command.
- **Source** – Displays the line number and the name of the source file being executed.
- **Message** – Similar to the Compact view, but also displays the first 20 letters of text from the `display` library routine.

The following items apply to all computer or user views:

- To make tracking certain virtual testers easier, you can change which virtual testers are displayed. For more information, see *Filtering and Sorting Views* on page 127.
- When you display a computer or user view, you can also display the test script that the virtual tester currently is running. Double-click on the number in the first column, next to the virtual tester. TestManager displays the test script. For more information, see *Displaying the Script View* on page 120.
- When a virtual tester terminates abnormally, TestManager writes a message stating the reason for termination to the running Suite window. Right-click on the terminated virtual tester, and then select the **View Termination Message** option.

A virtual tester that terminates abnormally can be identified easily in the views because its **Exited** state is displayed in red.

The rest of this section describes and gives examples of each view.

## Compact View

The Compact view contains summary information about all virtual testers, as shown in the following figure:

	Groups		Script	State	Time
	Suite	Computer			
1	Accounting[1]	Local computer[01]	Calculate Hours	Client Connection	00:00:00
2	Accounting[2]	Local computer[02]	Calculate Hours	Thinking	00:00:08
3	Accounting[3]	Local computer[03]	Calculate Hours	VU Code	00:00:00
4	Accounting[4]	Local computer[04]	Calculate Hours	VU Code	00:00:00
5	Accounting[5]	Local computer[05]	Calculate Hours	Thinking	00:00:00
6	Data Entry[1]	Local computer[06]		Exited	
7	Data Entry[2]	Local computer[07]		Exited	
8	Data Entry[3]	Local computer[08]		Exited	
9	Data Entry[4]	Local computer[09]		Exited	
10	Data Entry[5]	Local computer[10]		Exited	
11	Data Entry[6]	Local computer[11]		Exited	
12	Data Entry[7]	Local computer[12]		Exited	
13	Sales[01]	Local computer[13]		Exited	

The Compact view displays the following information:

- **Groups** – Contains information about the computer or user group. The **Suite** column displays the group to which the virtual tester belongs, as well as a number identifying the individual virtual tester within the group. This identification remains constant throughout the run. The **Computer** column displays the computer on which a virtual tester is running.

- **Test Script** – The test script that each virtual tester is running. In this example, all Accounting virtual testers are running the `Calculate Hours` test script.
- **State** – The state that the virtual tester is in. In this example, one virtual tester is connecting to the client, two virtual testers are thinking, two virtual testers are executing VU code, and eight virtual testers have exited the run. If a tester has terminated abnormally, TestManager displays the word **Exited** in red.
- **Time** – The time each virtual tester has been in that state. In this example, the Accounting[2] virtual tester has been thinking for 8 seconds.

## Results View

The Results view contains information about the success and failure rate of each emulation command, as shown in the following figure:

	Groups		Script	Command	State	Time	Streak	Failure Rate		
	Suite	Computer						Last 10	Script	Overall
1	Accounting[1]	Local computer[01]	Calculate H	http_header	Waiting for	00:00:01	240 Succe	0	0	0
2	Accounting[2]	Local computer[02]	Calculate T	http_request	Client Conn	00:00:00	262 Succe	0	0	0
3	Accounting[3]	Local computer[03]	Calculate H	http_request	Thinking	00:00:01	239 Succe	0	0	0
4	Accounting[4]	Local computer[04]	Calculate H	http_request	Thinking	00:00:01	239 Succe	0	0	0
5	Accounting[5]	Local computer[05]	Calculate H	http_header	Waiting for	00:00:01	49 Succes	0	1	1
6	Data Entry[1]	Local computer[06]			Exited					
7	Data Entry[2]	Local computer[07]			Exited					
8	Data Entry[3]	Local computer[08]			Exited					
9	Data Entry[4]	Local computer[09]			Exited					
10	Data Entry[5]	Local computer[10]			Exited					
11	Data Entry[6]	Local computer[11]			Exited					
12	Data Entry[7]	Local computer[12]			Exited					

In addition to the information displayed in the Compact view, the Results view contains the following information:

- **Command** – The emulation command that is executing. In this example, two Accounting virtual testers are executing the `http_header_request` command and three Accounting virtual testers are executing the `http_request` command.
- **Streak** – Succession of successes or failures of emulation commands. For example, 49 successes indicates that 49 emulation commands in a row have been successfully executed.
- **Failure Rate** – The number of failures in the last ten emulation commands, the number of failures in the current test script, and the number of failures overall.

## Source View

The Source view displays the line number and the name of the source file that is being executed, as shown in the following figure:

	Groups		Script	Command	State	Time	Source		Cmd Count
	Suite	Computer					File	Line	
1	Accounting[1]	Local computer[01]	Calculate Hours	http_header_re	Waiting for Res	00:00:19	Calculate	2075	241
2	Accounting[2]	Local computer[02]	Calculate Taxe	http_nrecv	VJ Code	00:00:00	Calculate	543	63
3	Accounting[3]	Local computer[03]	Calculate Taxe	http_request	Client Connect	00:00:00	Calculate	139	19
4	Accounting[4]	Local computer[04]	Calculate Taxe	http_header_re	Waiting for Res	00:00:00	Calculate	195	26
5	Accounting[5]	Local computer[05]	Calculate Hours	http_header_re	Waiting for Res	00:00:19	Calculate	2075	241
6	Data Entry[1]	Local computer[06]			Exited				
7	Data Entry[2]	Local computer[07]			Exited				
8	Data Entry[3]	Local computer[08]			Exited				
9	Data Entry[4]	Local computer[09]			Exited				
10	Data Entry[5]	Local computer[10]			Exited				
11	Data Entry[6]	Local computer[11]			Exited				
12	Data Entry[7]	Local computer[12]			Exited				

In addition to the information displayed in the Compact view, the Source view contains the following information:

- **Command** – The emulation command that is executing. In this example, three Accounting virtual testers are executing the `http_header_request`, one is executing `http_nrecv`, and one is executing `http_request`.
- **Source** – Generally the same as the test script. However, if a test script calls another test script, or if a test script contains an include file, the called test script or the include file is displayed.
- **Cmd Count** – The number of emulation commands that have been executed in the current test script. This number helps you distinguish between executions of the same command on different loop iterations. In this example, the first virtual tester is on line 2075 and the command count is 241. If line 2075 is part of a loop, the next time TestManager executes that line, the line number is the same but the command count increases.

## Message View

The Message view is similar to the Compact view, but it also displays messages from a `display` library routine. If you have added such a routine to a test script, you may want to show this view.

The following figure shows an example of a Message view:

	Groups		Script	State	Time	Message
	Suite	Computer				
1	Accounting[1]	Local computer[01]	Calculate Hours	Waiting for Response	00:00:51	
2	Accounting[2]	Local computer[02]		Exited		
3	Accounting[3]	Local computer[03]	Calculate Taxes	VU Code	00:00:00	
4	Accounting[4]	Local computer[04]		Exited		
5	Accounting[5]	Local computer[05]	Calculate Hours	Waiting for Response	00:00:51	
6	Data Entry[1]	Local computer[06]		Exited		
7	Data Entry[2]	Local computer[07]		Exited		
8	Data Entry[3]	Local computer[08]		Exited		
9	Data Entry[4]	Local computer[09]		Exited		
10	Data Entry[5]	Local computer[10]		Exited		
11	Data Entry[6]	Local computer[11]		Exited		

In addition to the information displayed in the Compact view, the Message view contains the following information:

- **Message** – Text displayed from a running test script. If the virtual tester executes a `display` library routine (VU Language `display()` function and the TSS `TSSDisplay()` function), the first 20 characters of its text appear here, and remain until they are overwritten by the next `display` routine.

## Full View

The Full view contains complete information about all virtual testers, as shown in the following figure:

	Groups		Script	Command	State	Time	Source		Cmd Count	Streak	Last
	Suite	Computer					File	Line			
1	Accounting[1]	Local computer[01]	Calculate	http_heade	Waiting for	00:00:00	Calculate	36	2	1 Success	0
2	Accounting[2]	Local computer[02]	Calculate	http_heade	Waiting for	00:00:00	Calculate	36	2	1 Success	0
3	Accounting[3]	Local computer[03]	Calculate	http_reque	Client Conn	00:00:01	Calculate	22	1	None	0
4	Data Entry[1]	Local computer[04]	Login	http_heade	Waiting for	00:00:00	Login.s	75	2	1 Success	0
5	Data Entry[2]	Local computer[05]	Login	http_heade	Waiting for	00:00:00	Login.s	75	2	1 Success	0
6	Data Entry[3]	Local computer[06]	Login	http_reque	Client Conn	00:00:00	Login.s	61	1	None	0
7	Data Entry[4]	Local computer[07]	Login	http_heade	Waiting for	00:00:00	Login.s	75	2	1 Success	0
8	Sales[1]	Local computer[08]	Login	http_heade	Waiting for	00:00:00	Login.s	75	2	1 Success	0
9	Sales[2]	Local computer[09]	Login	http_reque	Thinking	00:00:00	Login.s	85	3	2 Success	0
10	Sales[3]	Local computer[10]	Login	http_reque	Client Conn	00:00:00	Login.s	61	1	None	0
11	Sales[4]	Local computer[11]	Login	http_reque	Client Conn	00:00:00	Login.s	61	1	None	0
12	Sales[5]	Local computer[12]	Login	http_heade	Waiting for	00:00:00	Login.s	75	2	1 Success	0

In addition to the information displayed in the Compact view, the Full view contains the following information:

- **Command** – The emulation command that is executing. In this example, all virtual testers are executing either `http_header_request` or `http_request`.
- **Source** – Generally the same as the test script. However, if a test script calls another test script, or if a test script contains an include file, the called test script or the include file is displayed.
- **Cmd Count** – The number of emulation commands that have been executed in the current test script. If this test script is part of a loop, the next time TestManager executes that line, the line number stays the same but the command count increases. Thus, the command count helps you distinguish between executions of the same command on different loop iterations.
- **Streak** – Succession of successes (S) or failures (F) in the entire suite run. For example, “2 Success” means that 2 emulation commands in a row have been successfully executed from the time the suite began running.
- **Failure Rate** – The number of failures in the last ten emulation commands, the number of failures in the current test script, and the number of failures overall.

## Displaying the Shared Variables View

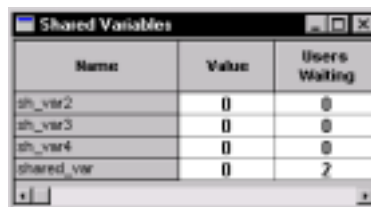
The Shared Variables view lets you inspect the values of any shared variables that you have set in your suite or test script.

**Note:** This view contains information that is more focused on performance testing than on functional testing.

To display the Shared Variables view:

- Click **Monitor > Shared Variable**.

The following figure shows a Shared Variables view:



Name	Value	Users Waiting
sh_var2	0	0
sh_var3	0	0
sh_var4	0	0
shared_var	0	2

This view shows the name of each shared variable, the value of the variable, and the number of virtual testers waiting for the shared variable to reach a certain value.

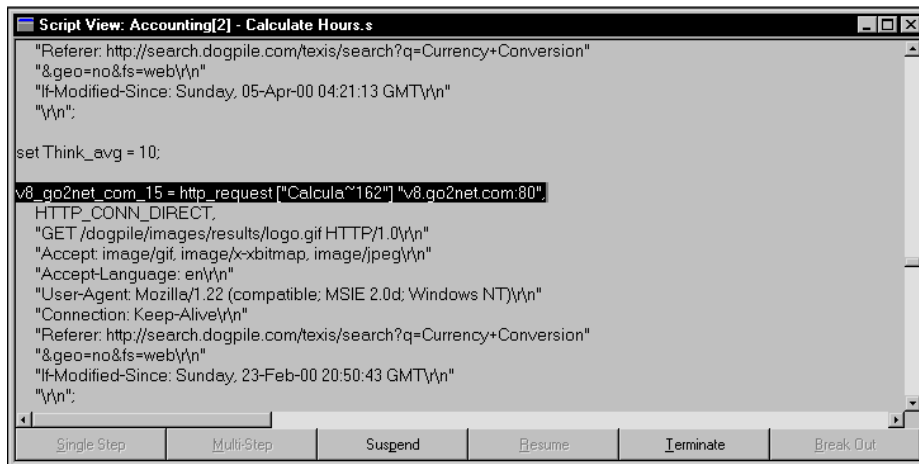
You can change the value of a shared variable from this view. For information about changing the value, see *Changing the Value of a Shared Variable* on page 130.

## Displaying the Script View

The Script view displays the line of code that a virtual tester is running. The Script view is useful if you want to watch the progress of a virtual tester through a test script.

To display the Script view:

- Click **Monitor > Script**.



The screenshot shows a window titled "Script View: Accounting[2] - Calculate Hours.s". The main area contains the following text:

```
"Referer: http://search.dogpile.com/taxis/search?q=Currency+Conversion"
"&geo=no&fs=web\y\n"
"If-Modified-Since: Sunday, 05-Apr-00 04:21:13 GMT\y\n"
"\y\n";

set Think_avg = 10;

v8_go2net_com_15 = http_request["Calcula"162"] "v8_go2net.com:80";
HTTP_CONN_DIRECT;
"GET /dogpile/images/results/logo.gif HTTP/1.0\y\n"
"Accept: image/gif, image/x-bitmap, image/jpeg\y\n"
"Accept-Language: en\y\n"
"User-Agent: Mozilla/1.22 (compatible; MSIE 2.0d; Windows NT)\y\n"
"Connection: Keep-Alive\y\n"
"Referer: http://search.dogpile.com/taxis/search?q=Currency+Conversion"
"&geo=no&fs=web\y\n"
"If-Modified-Since: Sunday, 23-Feb-00 20:50:43 GMT\y\n"
"\y\n";
```

At the bottom of the window, there is a control bar with buttons: Single Step, Multi-Step, Suspend, Resume, Terminate, and Break Out.

When you display a Script view, you must specify one or more virtual testers that are running the particular test script.

The Script View window shows the test script that is currently running. The test script displays, line by line, what the virtual tester is doing.

## Displaying the Sync Points View

The Sync Points view displays information about the synchronization points that you have set in the suite or that you have included in a test script. This view also lets you manually release virtual testers that are waiting on a synchronization point.

**Note:** This view contains information that pertains to performance testing.

To display the Sync Points view:



- Click **Monitor > Sync Points**.

Name	State	Time	Timeout	Virtual Testers			Delay (ms)	
				Arrived	to Sync	Late	Min	Max
Stress_Test	Released	00:05:13	Infinite	9	9	0	00:00:00	00:00:00
Accounting	Released	00:05:13	Infinite	1	1	0	00:00:00	00:00:00

The Sync Points view displays the following information:

- **Name** – The name of the synchronization point.
- **State** – The state the synchronization point is in. The states can be:
  - **Empty** – No testers have arrived at the synchronization point.
  - **Waiting** – At least one tester has arrived at the synchronization point, but not all of the testers have arrived.
  - **Released** – The testers are released from the synchronization point. This column also indicates whether the testers were released because they all reached the synchronization point (**Normal**), whether you have released the testers manually (**Monitor**), or whether the synchronization points have timed out (**Timeout**).
- **Time** – The time the synchronization point has been in the current state.
- **Timeout** – The timeout period that you set in the suite, or **Infinite**, if you did not set a timeout period.
- **Virtual Testers** – The number of virtual testers that have reached the synchronization point:
  - **Arrived** – The number of testers that have arrived at the synchronization point before it was released.
  - **To sync** – The number of testers that must arrive to release the synchronization point.

- **Late** – The number of testers that arrived at the synchronization point after it was released.
- **Delay** – If the release time is together, this is the release delay that you have set in the suite. If the release type is staggered, the minimum and maximum release times.

## Displaying Virtual Testers Waiting on a Synchronization Point

To display the virtual testers waiting on a synchronization point:

- Click **Monitor > Sync Points**.

## Releasing a Synchronization Point

You might decide to release a synchronization point, even though the required number of virtual testers has not yet been reached. Subsequent testers that arrive at the synchronization point are not held. However, if you have set a restart time and a maximum time in the suite, the testers will be delayed. So, for example, if you release a synchronization point but have set a restart time of 1 second and a maximum time of 4 seconds, each virtual tester who reaches that synchronization point is delayed from 1 to 4 seconds.

To release a synchronization point:

- Click **Monitor > Sync Points**.

## Displaying the Computer View

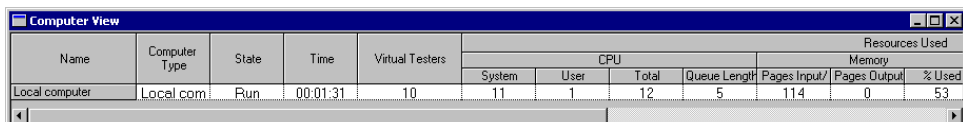
Use the Computer view to check the computer resources used during a suite run, as well as the status of the Local and Agent computers at the beginning and end of a run.

## Viewing Resource Usage During a Run

When you choose to view resource usage during a run, TestManager displays the computer resources used for each Local and Agent computer in the run.

To check computer resources used during a suite run:

- Click **Monitor > Computers**.



Name	Computer Type	State	Time	Virtual Testers	Resources Used						
					CPU			Memory			
					System	User	Total	Queue Length	Pages Input/	Pages Output	% Used
Local computer	Local.com	Run	00:01:31	10	11	1	12	5	114	0	53

The Computer view displays the following information:

- **Name** – The name of the computer that you specified in the suite. Local is the local computer.
- **Computer Type** – The type of computer: either Local, Agent, or Server.
- **State** – The state the computer is in. When you display the Computer view manually, it is generally in the run state.
- **Time** – The time the computer has been in the current state. The time is displayed in *hours:minutes:seconds*.
- **Virtual Testers** – The total number of virtual testers assigned to run on the computer.
- **CPU System** – The percent of CPU cycles servicing the operating system.
- **CPU User** – The percent of CPU cycles servicing virtual tester processes.
- **CPU Queue Length** – The number of processes or threads that are ready to run but have to wait in a queue.

This number should be 0 or very small unless the **CPU System** and **CPU Virtual Tester** percentages are close to 100.

- **Memory Pages Input/Sec.** – The number of pages per second that are read into memory.
- **Memory Pages Output/Sec.** – The number of pages per second that are swapped onto disk. This number should be considerably smaller than the memory pages that are input.

Together, these numbers can indicate memory bottlenecks.

- **Memory % Used** – The percentage of memory used.
- **Disk Transfers/Sec.** – The disk access speed (seek, rotation, and transfer time) for up to four disks. If one disk is much slower than the others, it might be fragmented. If the transfer rate peaks when the response time is down, this also could indicate a problem with your disk.

**Note:** If you are monitoring resources for a computer that runs HP-UX, AIX, or LINUX, the **Disk Transfers/Sec.** column will always show **n/a**. This is because those platforms are unable to supply TestManager with disk transfer information. However, all other columns will be correct.

- **% Disk Used** – The percent of used disk space on the monitored disk.

- **Delay** – This lets you gauge the general state of your network. At regular intervals, TestManager sends a small ICMP packet to the remote computer and times that request, in ms. This time does not include any service time used by a virtual tester-level process on the remote computer. The time should stay relatively consistent and quite small. A large number or a number that varies widely might indicate network problems.
- **Service Time** – This lets you gauge the general state of your network. At regular intervals, TestManager sends a small TCP packet to the specified computer and times that request, in ms. This time includes the service time for a virtual tester-level process on the remote computer to reply to the packet. The time should stay relatively consistent and quite small. A large number or a number that varies widely might indicate network problems.

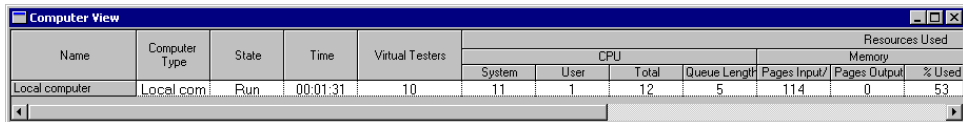
**Note:** In addition to monitoring your computer resources, you can report on them. The Response vs. Time report lets you compare your response time with your computer resource usage.

## Graphing Resource Usage During a Run

You can graph the resources that your computer uses during a suite run. This provides you with a visual representation of resource usage. Within this graph of resources, you can change the color of an item in the graph, remove an item from the graph, or remove all items in the graph.

To graph computer resources:

- Click **Monitor > Computers**.



Name	Computer Type	State	Time	Virtual Testers	Resources Used						
					CPU			Memory			
					System	User	Total	Queue Length	Pages Input/	Pages Output	% Used
Local computer	Local.com	Run	00:01:31	10	11	1	12	5	114	0	53

## Viewing Computers at the Start or End of a Run

The Computer view appears automatically when Agent computers start up. When all Agents are up and running, the Computer view closes. When Agents begin shutting down, the Computer view reappears automatically so you can watch the cleanup activities such as transferring files to the Local computer.

The Computer view includes **Progress** messages, which indicate when the computer is creating or initializing processes, transferring files, terminating virtual testers, and so on.

The Computer view displays the following information:

- **Name** – The name of the computer that you specified in the suite. Local is the local computer.
- **State** – The state the computer is in. It can be one of the following:
  - **Not Connected** – The Local computer has not yet connected to the Agent computer.
  - **Initializing** – The computer is being initialized, or is transferring compiled test scripts and datapools that are out of date.
  - **Run** – The computer is running virtual testers.
  - **Termination** – The computer is in termination mode, waiting for virtual testers to exit.
  - **Clean Up** – The computer is cleaning up before exiting. This includes transfer and removal of the result files.
  - **Exit** – The computer has exited.
- **Time** – The time the computer has been in the current state. The time is displayed in *hours:minutes:seconds*.
- **Virtual Testers** – The total number of virtual testers assigned to run on the computer.

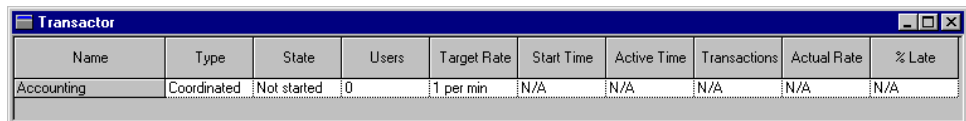
## Displaying the Transactor View

The Transactor view shows the status of the transactors that you inserted into the suite.

**Note:** This view contains information that is more focused on performance testing than on functional testing.

To display a Transactor view:

- Click **Monitor > Transactors**.



Name	Type	State	Users	Target Rate	Start Time	Active Time	Transactions	Actual Rate	% Late
Accounting	Coordinated	Not started	0	1 per min	N/A	N/A	N/A	N/A	N/A

The Transactor view contains the following information:

- **Name** – The name that you gave the transactor when you inserted it in the suite.
- **Type** – Whether the transaction is Independent or Coordinated.

- **State** – The state that the transactor is in. It can be one of the following:
  - **Not Started** – An initial state, when the transactor has not run any virtual testers.
  - **Arriving** – This state pertains to Coordinated transactors only. At least one tester has arrived at the sync point, but not all testers have arrived.
  - **Active** – The transactor is running at least one virtual tester.
  - **Inactive** – The transactor is not running any virtual testers.
- **Users** – The number of virtual testers in the **Arriving** or **Active** states.
- **Target Rate** – The rate that you set for the transactor when you inserted it in the suite.
- **Start Time** – The time the transactor first entered the **Active** state.
- **Active Time** – The total amount of time the transactor has been in the **Active** state.
- **Transactions** – The number of transactions that are scheduled by the transactor, but not necessarily completed by the virtual tester.
- **Actual Rate** – The rate that the transactions are actually running. TestManager calculates this rate by dividing **Transactions** by the **Active Time**.

The **Target Rate** specifies the number of *started* transactors, but the **Actual Rate** calculates the number of *completed* transactions. Because the transactions take some time to complete, the **Actual Rate** will approach, but will not reach, the **Target Rate**. However, over time and enough transactions, the **Actual Rate** should become close to the **Target Rate**.

- **% Late** – The percent of transactors that were unable to begin running at the desired time.

For coordinated transactors, this usually means that not enough testers are available to run the transactors. You may want to run the suite again with more testers.

For independent transactors, this usually means that the time it takes to run one transaction is longer than the time between two transactions.

If too many transactors are late, the target transaction rate will not be maintained or the transactor will not accurately simulate peaks in the transaction rate.

## Displaying the Group Views

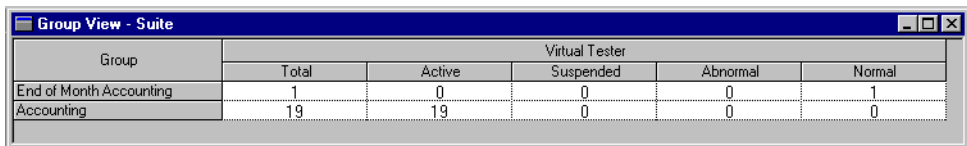
The Group views show the status of all user groups that you defined in the suite. Both Group views show the same information, but the Suite view shows the information by user group, and the Computer view shows the information by computer.

To display a Group view:

- Click **Monitor > Groups**.

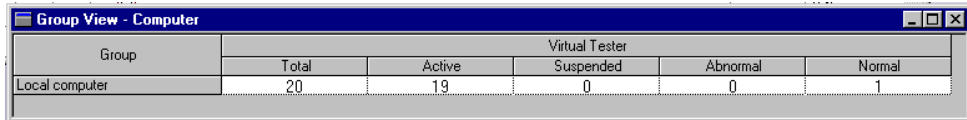
The two Group view types are:

- **Suite** – A list of the user groups in a suite. The following figure shows the Suite view:



Group	Virtual Tester				
	Total	Active	Suspended	Abnormal	Normal
End of Month Accounting	1	0	0	0	1
Accounting	19	19	0	0	0

- **Computer** – A list of the user groups assigned to the same computer. The following figure shows the Computer view:



Group	Virtual Tester				
	Total	Active	Suspended	Abnormal	Normal
Local computer	20	19	0	0	1

The Group views contain the following information:

- **Total** – The total number of virtual testers in the group.
- **Active** – The number of virtual testers in the group currently running.
- **Suspended** – The number of virtual testers in the group that are suspended.
- **Abnormal** – The number of virtual testers that terminated without completing all of their assigned tasks.
- **Normal** – The number of virtual testers that completed their tasks successfully.

To display the virtual testers in the groups, right-click the group name in the left column, and then click **See Users**.

## Filtering and Sorting Views

This section discusses how to customize a view. For example, you can sort virtual testers in various ways, or you can filter virtual testers and groups so that only certain information is displayed.

## Sorting the Virtual Testers Displayed in a User/Computer View

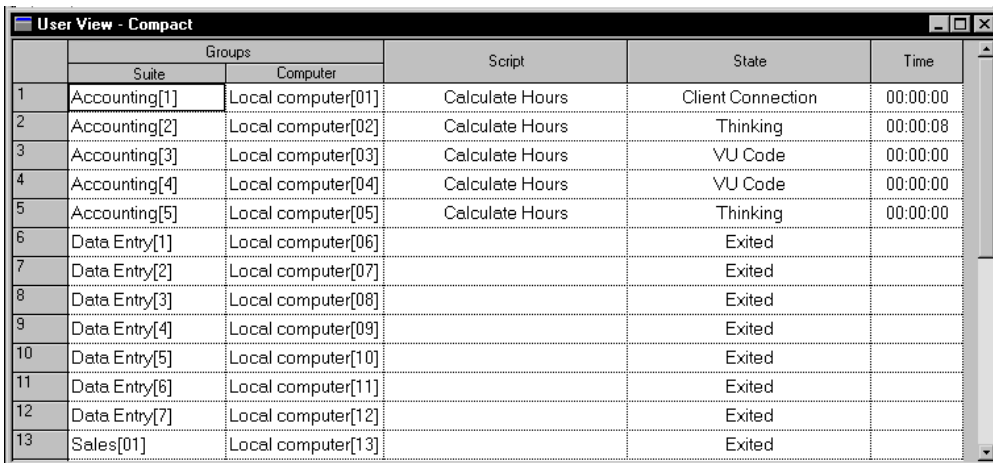
While displaying a User/Computer view, you may want to see the virtual testers in a particular order. For example, you can sort the virtual testers alphabetically, or you can sort them in the order in which they started.

You can sort virtual testers in the following orders:

- **Suite Order** – The order in which the user group appears in the suite.
- **Execution Order** – The order in which the virtual testers are started.
- **Suite Groups** – Alphabetical listing of suite groups.
- **Computer Groups** – Alphabetical listing of computer groups.

**Note:** The current sort order is unavailable. In this example, Execution Order is unavailable because it is the sort order being used.

To change the order in which the virtual testers are displayed, select a column under the **Suite** or **Computer** heading from an open user or computer view and right-click to view the context menu.



	Groups		Script	State	Time
	Suite	Computer			
1	Accounting[1]	Local computer[01]	Calculate Hours	Client Connection	00:00:00
2	Accounting[2]	Local computer[02]	Calculate Hours	Thinking	00:00:08
3	Accounting[3]	Local computer[03]	Calculate Hours	VU Code	00:00:00
4	Accounting[4]	Local computer[04]	Calculate Hours	VU Code	00:00:00
5	Accounting[5]	Local computer[05]	Calculate Hours	Thinking	00:00:00
6	Data Entry[1]	Local computer[06]		Exited	
7	Data Entry[2]	Local computer[07]		Exited	
8	Data Entry[3]	Local computer[08]		Exited	
9	Data Entry[4]	Local computer[09]		Exited	
10	Data Entry[5]	Local computer[10]		Exited	
11	Data Entry[6]	Local computer[11]		Exited	
12	Data Entry[7]	Local computer[12]		Exited	
13	Sales[01]	Local computer[13]		Exited	

## Filtering a View

When you display a User/Computer view, you can filter virtual testers so that only certain testers appear. This is useful if your suite contains many virtual testers and you want to focus on the progress of a few of these testers.

### Filtering Virtual Testers

You can filter on virtual testers by including or excluding selected virtual testers:



- **Include** – only the virtual testers that you selected are displayed.
- **Exclude** – all virtual testers except those that you selected are displayed.

To filter on virtual testers:

- Select the virtual testers that you want to filter from an open user or computer view and right-click to display the shortcut menu.

### **Filtering a Virtual Tester by Value**

You can filter a virtual tester on any value that stays constant during the run, such as the name of its group, the type of test script it is running, or the name of the computer on which a virtual tester is running.

For example, you might be running a test with 200 virtual testers in the Accounting user group, 300 virtual testers in the Data Entry user group, and 500 virtual testers in the Sales user group. You want to see only virtual testers in the Data Entry group. Filter the group so that TestManager displays only the group with the “Data Entry” value.

To filter a virtual tester by value:

- Right-click in any cell under the Suite, Group, or Type headings of an open User or Computer view.

### **Filtering a Group View**

If the Group view displays many columns, you can filter out some columns to provide more room to view the columns that you want to see. You can filter a group on any value that stays constant during the run, such as the name of the computer or user group or the type of test script.

To filter a Group view:

- Right-click in the Group or Type heading of an open User or Computer view to see the shortcut menu.

### **Restoring the Default Views**

If you have zoomed in on a histogram bar, filtered a view, or changed the widths of a column in a view, you may want to restore the bar or views to their original settings.

To restore a view to its original setting, from the view you want to restore:

- Click **View > Reset**.

## Changing the Value of a Shared Variable

You can change the value of a shared variable when you are monitoring a suite.

If the shared variable is being dynamically updated, however, you cannot type in a new value. By the time you read the value, determine the new value, and change it, a virtual tester may have modified the value. If this occurs, your change is lost. Instead:

- 1 Type an operand in the **Value of** box.
- 2 Under **Operators**, choose an operator. If you choose the subtract (-) or divisor (/) operators, the order for operations is:

existing value - new value

existing value / new value

For example, assume the shared variable has a current value of 6. If you type 4 in the **Value of** box and click the - operator, the new value of the shared variable is 2, because  $6 - 4 = 2$ .

To change the value of a shared variable:

- Click **Monitor > Shared Variable**.

### Displaying the Virtual Testers Waiting on a Shared Variable

If your test scripts contain shared variables, you can see the virtual testers waiting on each shared variable.

To display the virtual testers waiting on a shared variable, double-click the variable name, or click the right mouse button.

## Debugging a Test Script

You may encounter problems when you are monitoring a suite. TestManager provides you with tools that enable you to debug a test script. When you debug a test script, it is a good idea to run the suite with just one virtual tester, correct the test script, and then run the suite as usual.

To debug a test script:

- Click **Monitor > Script** and select the virtual tester running the script you want to debug.

Debug a test script through the Script view. Through the Script view you have the following debugging choices:

- **Single Step** – Steps through a test script one emulation command at a time, allowing you to see what happens at each command. To use this choice, first suspend a virtual tester. This is useful for pinpointing problems.
- **Multi-step** – Steps through a test script multiple emulation commands at a time. To use this choice, first suspend the virtual tester. Then you can select a number of commands to execute at a time.
- **Suspend** – Suspends a virtual tester at the beginning of the next emulation command.
- **Resume** – Allows a suspended virtual tester to resume its progress through a test script.
- **Terminate** – Ends the virtual tester’s execution of a test script.
- **Break Out** – Moves a virtual tester out of the following three states:
  - Waiting on a shared variable
  - Waiting on a response
  - delay function

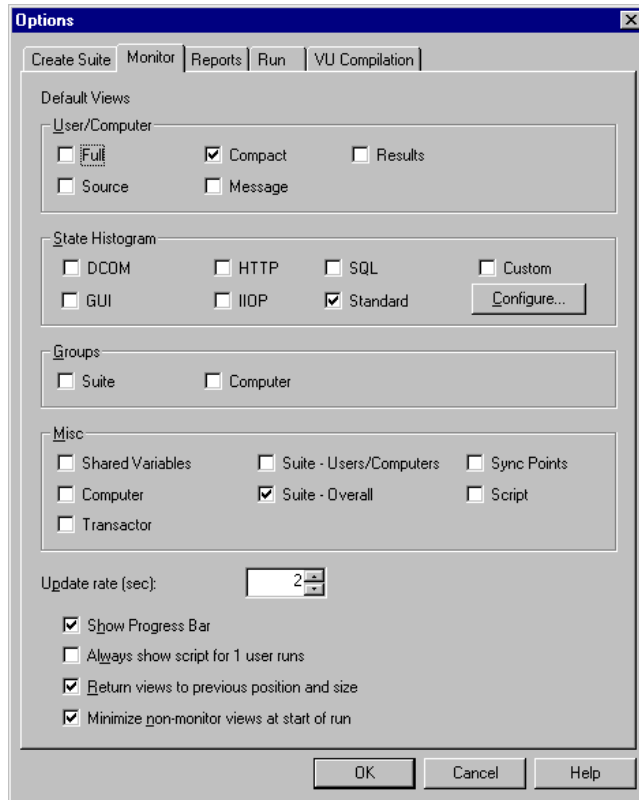
To debug Visual Basic, Java, or test scripts of other types, refer to the API documentation associated with that script type.

## Changing Monitor Defaults

When you monitor a suite, you can set which views are displayed automatically, how often the views are refreshed, and whether toolbars are displayed automatically when you run a suite. You can even configure the Custom histogram, and change its colors, as described in the next section.

To change monitoring defaults:

- Click **Tools > Options**, and then click the **Monitor** tab.

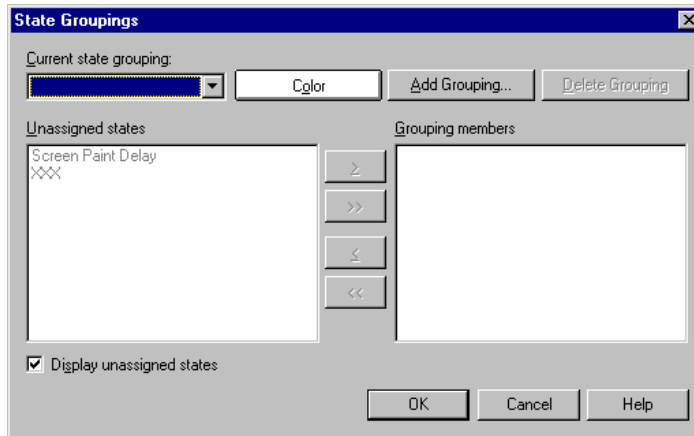


## Configuring Custom Histograms

By default, the Custom histogram is identical to the Standard histogram. However, unlike the other histograms, you can configure the Custom histogram. You can configure the groups, create new groups, and change the colors that designate a group.

To configure custom histogram states:

- From the **Monitor** tab of the Options dialog box, in the **State Histogram** area, click the **Configure** button.

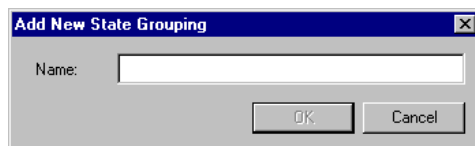


To assign or remove a state in a Custom Histogram group:

- In the State Groupings dialog box, select a group from the **Current State Grouping** field.

To add an entire group to the Custom histogram:

- In the State Groupings dialog box, click the **Add Grouping** button.



To delete a group from the Custom histogram:

- In the State Groupings dialog box, select the group you want to delete from the **Current State Grouping** box, and then click **Delete Grouping**.

When you delete a group, all states that were in the group are unassigned.

## Controlling the Suite During a Run

TestManager provides a variety of ways to help you control a suite while it is running. For example, you can suspend a suite to change settings or examine its progress.

## Suspending and Resuming Virtual Testers in a Suite

While a suite is running you can suspend and resume all virtual testers in the suite, or you can suspend and resume individual virtual testers. This is useful if a problem occurs during the run and you want to investigate it.

To suspend and resume all virtual testers in the suite:

- Click **Monitor > Suspend** or **Monitor > Resume**.

To suspend or resume individual virtual testers:

- Click **Monitor > Users** or **Monitor > Computers**, and select the virtual tester row that you want to suspend.

## Stopping a Suite

You can stop the execution of a suite. This is useful if there is a serious problem and you do not want to wait for the test to finish.

To stop a run:

- Click **Monitor > Stop**.



Stop a suite in one of these ways:

- **Abort** – Stops the run and does not save any results. Click this option if you do not plan to run any reports or look at any Virtual Tester Error or Virtual Tester Output file in the Test Log window.

- **Process Results** – Stops the run but saves the results so that you can run reports, and look at any Virtual Tester Error or Virtual Tester Output file in the Test Log window.
- **Save and Run Reports** – Stops the run, saves the results, and produces reports, just as if your run completed normally.

You can also specify **Clean-up time**. This is the amount of time allowed from the time you request termination until TestManager forces the termination of the run.

**Note:** When you abort a large suite that includes multiprocessor Local or Agent computers, choose a Clean-up time of 60 seconds or more to allow virtual testers (rtsvui processes) time to exit on their own. The default **Clean-up time** of 1 second often causes the Local computer to terminate many processes at once, and can result in leftover rtsvui processes. Although not harmful, they clutter the process table. They can be killed individually using Task Manager, or all at once by logging off.

When the suite completes the test run—whether it completes normally or you manually terminated it—TestManager displays default log data in the Test Log window if log information is available. For more information on the Test Log window, see Chapter 6, *Evaluating Tests*.





This chapter explains how to use the Test Log window of TestManager to view logs and interpret their contents. It also explains how to create and run reports to help you manage your testing efforts. This chapter includes the following topics:

- About test logs
- Viewing test log results
- Viewing test script results recorded with Rational Robot
- Reporting results

**Note:** For detailed procedures, see the TestManager Help.

## About Test Logs

---

Use the Test Log window of Rational TestManager to view the test logs created after you run a test case, test script, or suite.

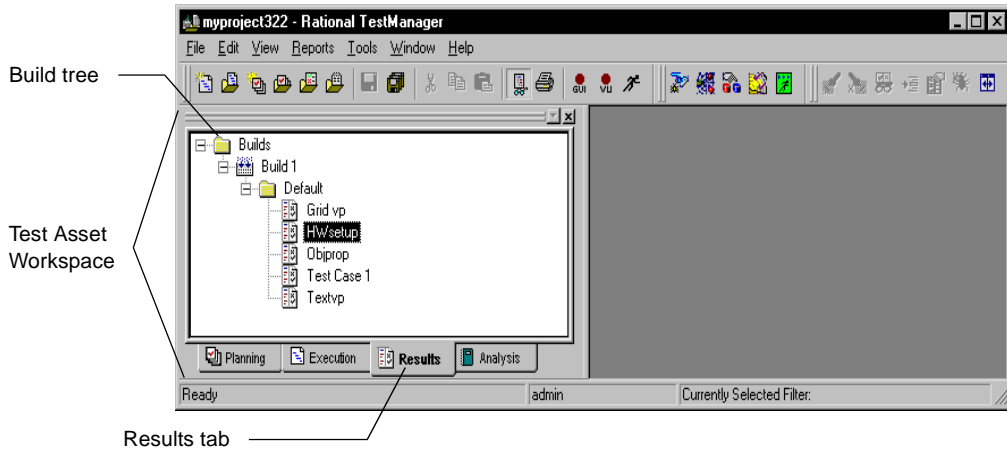
You can use the Test Log window to:

- Open a test log to view a result.
- Filter the data of a test log to view only the information you need.
- View all test cases with an unevaluated result in the Test Case Results tab of the Test Log window. This is particularly useful in evaluating the results of performance test cases. You can sort the test cases by Actual Result and then review and update all unevaluated test cases.
- Submit a defect for a failed log event. The test log automatically fills in build, configuration, and script information in the Rational ClearQuest defect form.
- Open the test script of a script-based log event in the appropriate test script development tool. For example if you created a GUI test script, Robot opens and displays the test script. If you created a manual script, Rational ManualTest opens and displays the test script.
- Preview or print data displayed in the active test log in the Test Log window.

## Opening a Test Log in TestManager

To open the Test Log window manually, do one of the following:

- Click **File > Open Test Log**.
- In the **Results** tab of the Test Asset Workspace, expand the Build tree and select a log.



You can open more than one test log in the Test Log window. If you have more than one test log open, the log that is currently active is the one that is acted upon when you use most menu commands.

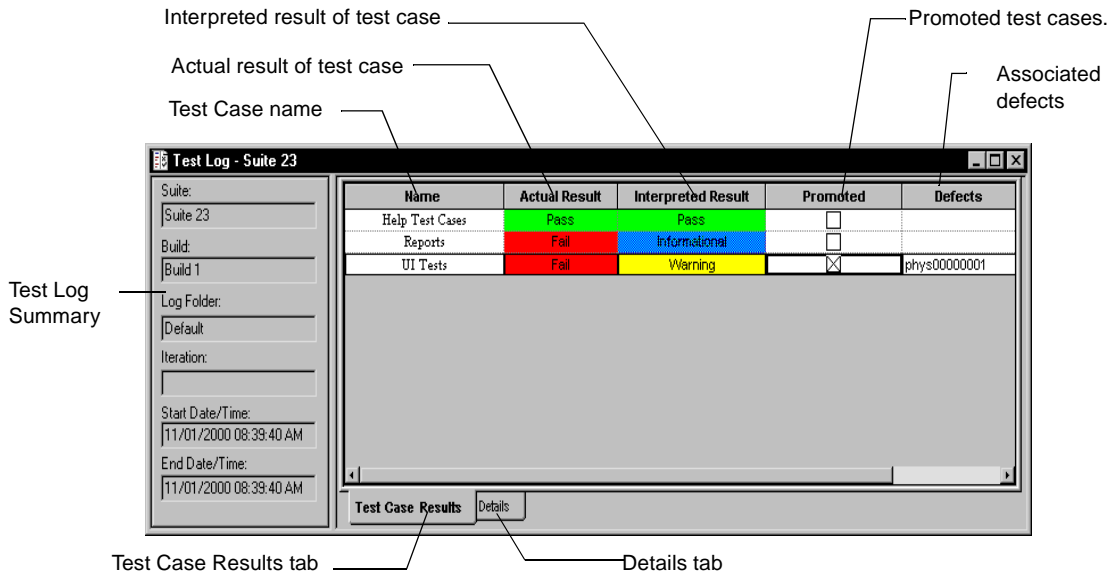
To control when the test log opens after running a test case, test script, or suite:

- Click **Tools > Options** and click the **Run** tab.

**Note:** You can also start the test log from a selected test script in a Rational TestFactory application map. For more information, see the *Using Rational TestFactory* manual or the Rational TestFactory Help.

## The Test Log Main Window

The Test Log window of TestManager contains the Test Log Summary area, the Test Case Results tab, and the Details tab.



- **Test Log Summary** – Lists the build, the name of the log folder, the iteration, the start date and time, and the end date and time of the run. If you run a suite, lists the suite name.
- **Test Case Results** tab – Displays all test case results. Only displays data when you run a test case.
- **Details** tab – Displays all events in the test log.

## Test Case Results Tab

The Test Case Results tab displays the test case results. The following information appears in the Test Case Results tab:

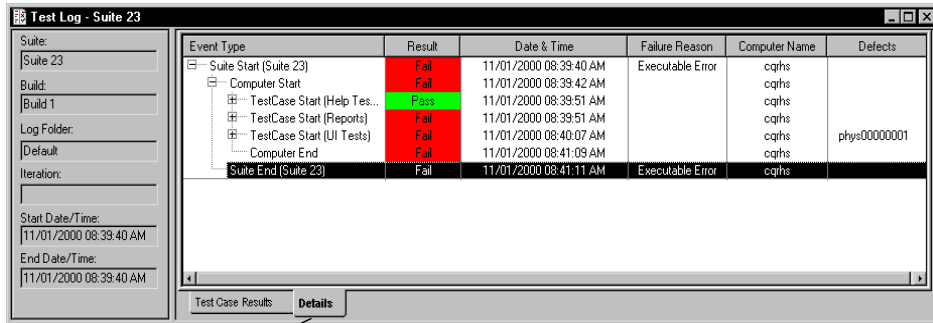
Name	Actual Result	Interpreted Result	Promoted	Defects
Help Test Cases	Pass	Pass	<input type="checkbox"/>	
Reports	Fail	Informational	<input type="checkbox"/>	
UI Tests	Fail	Warning	<input checked="" type="checkbox"/>	phys00000001

Test Case Results tab

- **Name** – The test case name.
- **Actual Result** – The result generated when you run a test case. The results in this column are color coded to reflect severity. Green indicates a test case passed its testing criteria, red indicates failure, yellow indicates warning, and blue indicates informational results.
- **Interpreted Result** – The interpreted results. You can interpret the actual result and change it according to your testing criteria. Green indicates a test case passed testing criteria, red indicates failure, yellow indicates warning, and blue indicates informational results.
- **Promoted** – Whether you have promoted the results of a test case. When you promote a test case, you indicate that the results of running a test case are valid and should be used to generate analysis reports.
- **Defects** – The defect numbers associated with this test case (if there are any).

## Details Tab

The **Details** tab of the Test Log window contains log events that are generated when you run a test script, test case, or suite.



Event Type	Result	Date & Time	Failure Reason	Computer Name	Defects
Suite Start (Suite 23)	Fail	11/01/2000 08:39:40 AM	Executable Error	cqhrs	
Computer Start	Fail	11/01/2000 08:39:42 AM		cqhrs	
Test Case Start (Help Tes...)	Pass	11/01/2000 08:39:51 AM		cqhrs	
Test Case Start (Reports)	Fail	11/01/2000 08:39:51 AM		cqhrs	
Test Case Start (UI Tests)	Fail	11/01/2000 08:40:07 AM		cqhrs	phys00000001
Computer End	Fail	11/01/2000 08:41:09 AM		cqhrs	
Suite End (Suite 23)	Fail	11/01/2000 08:41:11 AM	Executable Error	cqhrs	

Details tab

The following information appears in the **Details** tab:

- **Event Type** – Lists all log events, such as the script start and end, verification points, manual steps, and unexpected active windows.
- **Result** – Indicates the result of an event. Green indicates an event passed its testing criteria, red indicates failure, yellow indicates warning, and blue indicates informational results. If a verification point fails in a test script created using Rational Robot or Rational Quality Architect, you can open the appropriate Comparator. The Comparator shows the baseline and actual files so you can evaluate any failures found and determine whether they are intentional changes or defects.
- **Date & Time** – Shows the date and start time of a log event.
- **Failure Reason** – Shows the failure reason for the failure of a log event.
- **Computer Name** – Shows the computer on which the script was run.
- **Defects** – Shows the defect number associated with this test case (if there is one).

## About Log Filters

TestManager allows you create a log filter to narrow down the amount of data displayed in the Test Log window. A log filter can make it easier to view large test logs.

You can:

- Create or edit log filters.

- Choose a filter to narrow down the amount of logged data displayed in the Test Log window.
- Copy, rename, or delete a log filter. The copy feature is useful when you want to create multiple filters that are similar to one another. After you create the first filter, you can make a copy of it. Then you can edit the copied filter to make the necessary modifications. You can also rename and delete a log filter.

## Creating and Editing a Log Filter

To create or edit a log filter:

- Click **Tools > Manage > Log Filters**. Click **New**, or select a filter and click **Edit**. Then select the type of event information you want to filter on each of the tabs.

If you filter an event type, the filter includes all information included in the event type as well as the event type itself.

## Applying a Test Log Filter

After you create a test log filter, you need to set the filter that you want to use to narrow down the amount of logged data displayed in the Test Log window.

To apply a test log filter:

- Open a log and click **View > Set Test Log Filter** and select the filter.

To turn off all log filters, click the test log filter **All**. This filter displays all information logged in the Test Log window when you run a test case, test script, or suite.

## Viewing Test Log Results

---

After running a test case, test script, or suite, you can quickly evaluate the results in the Test Log window.

### Viewing Test Case Results

The results of all test cases appear in the **Test Case Results** tab of the Test Log window.

In the **Test Case Results** tab you can:

- Sort by name, actual result, interpreted result, or promotion status.

To sort test cases:

- In the **Test Case Results** tab, click **View > Sort By** and then select how you want to sort the test cases.

- Show test cases by the following criteria:

- Actual result with pass, fail, warning, or other.
- Interpreted with pass, fail, warning, or other.
- You can also hide the equivalent results.

To show test cases by certain criteria:

- In the **Test Case Results** tab click **View > Show Test Cases**. Then select the criteria you want to see.

- Display event details for a particular test case in the Test Case Results tab.

To display event details:

- In the **Test Case Results** tab, select a test case, and then click **View > Event Details**. TestManager displays the **Details** tab and locates the event details for the test case you selected.

### Viewing Events Details

Detailed information on each event is available in the **Details** tab of the Test Log window.

In the **Details** tab, you can:

- Collapse and expand events.
- Find a particular result, event type, protocol, failure reason, verification point, or command, or search for the name and value of a specific event property.

You can view an event and navigate through all of the failures (appearing in red in the **Result** column of the **Details** tab) from the Log Event Properties window.

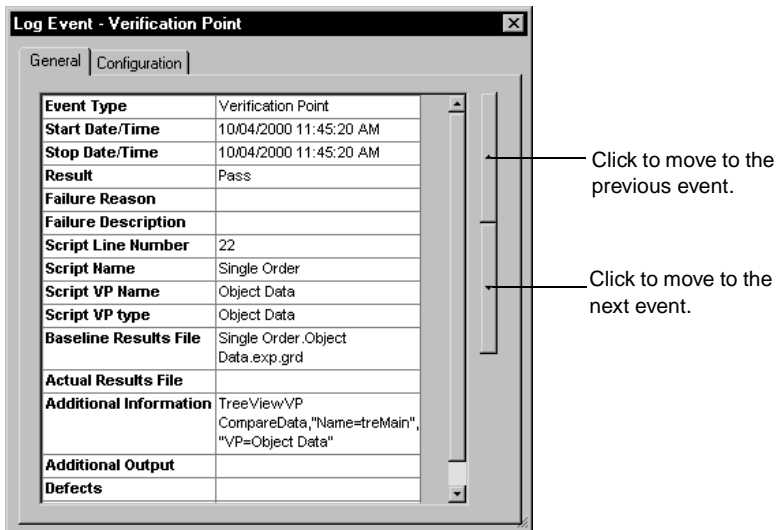
To view a particular event:

- 1 Click the **Details** tab in the Test Log window.
- 2 Click **View > Properties**.

You can keep the Log Event Properties window open while you move through each event in the **Details** tab of the Test Log window. You can also resize and move this window.

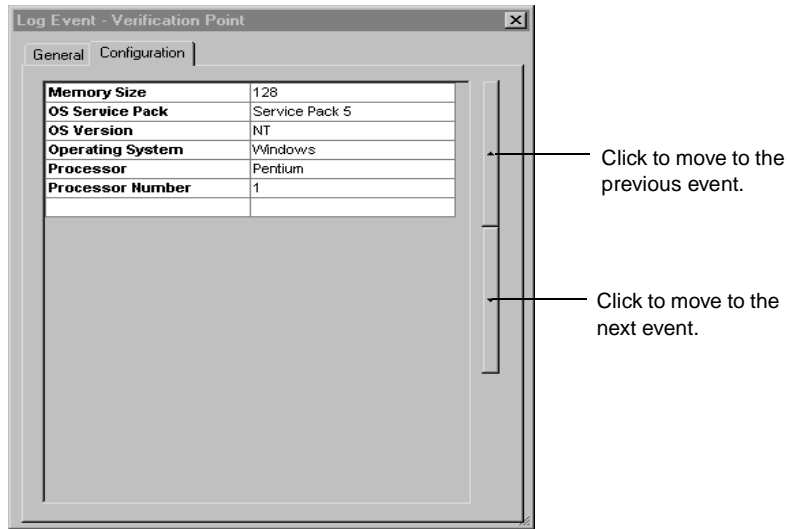
**Note:** If you previously used Rational Suite PerformanceStudio, the information found in the Trace and Analog reports is now available in the Log Event Properties window.

The **General** tab of the Log Event Properties window displays the type of event, the date and time the event was recorded, the script name, result information (if any), and other information about a log event.



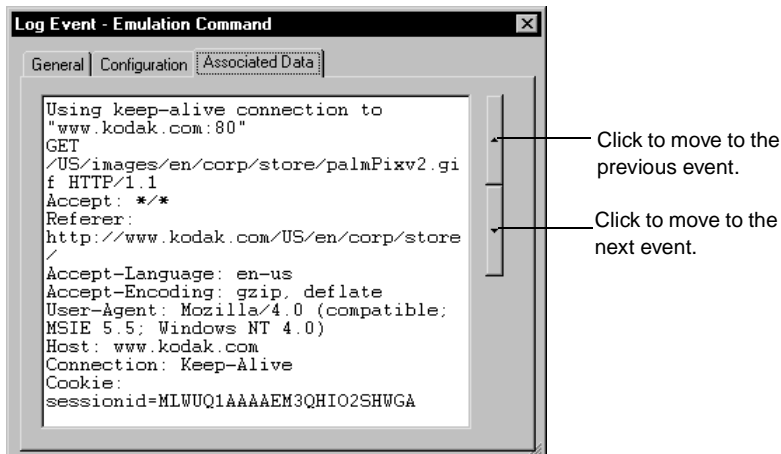
The **Configuration** tab of the Log Event Properties window displays the configuration information of the computer on which you recorded the test script.





If data is associated with an event, TestManager displays an **Associated Data** tab in the Log Event Properties window. Associated data can be a response to a query or even a graphics file.

**Note:** This tab is available *only* when data is associated with a selected event.



## Viewing a Test Script

You can select any log event that is associated with a script and view it in the tool that you used to create the script. For example, if you create a GUI script and open the script from the Test Log window, the script opens in Robot. If you create a manual script and open the script from the Test Log window, it opens in Rational ManualTest.

**Note:** When you double-click an event in an open log generated from Robot under Rational Purify, Quantify, or PureCoverage, the script opens in Robot, and the file opens in the diagnostic tool. For information about setting diagnostic tools options, see the Robot Help.

To view a script:

- 1 Open a test log.
- 2 Click the **Details** tab.
- 3 Right-click a script start or script end event, and click **Open Script**.

## Working with Test Logs

When working with test logs, you can:

- Open a test log.
- Rename a test log.
- View the properties of a test log (the name, description, build, and log folder).

To do any of these tasks:

- 1 Click the **Results** tab in the Test Asset Workspace.
- 2 Right-click a test log, then select an item on the shortcut menu.

**Note:** You can also print data displayed in the Test Log window. For information, see *Printing a Test Log* on page 150.

## About Test Logs

Test logs record everything that happens during a test script run from the time the script begins until it ends. Unless logging is specifically turned off, every virtual tester action, system call, verification point, and result is included in the logging process. You can view properties for every event from the Test Log window and you can also view certain test logs as a whole.

## Suite Log

The suite log contains all the messages associated with a suite run. It is the same information that you see in the Messages window when you run a suite. The log contains build, log folder and log name information, and messages about checking the suite, compiling test scripts, and any warnings or errors associated with the suite.

To view the suite log:

- Right-click a suite start event in the Test Log window, and click **View Suite Log**.

To print a suite log:

- From an open suite log, click **File > Print**.

## Virtual Tester Error File

The error file contains any runtime error information associated with a specific virtual tester.

**Note:** The error file does not always exist. If an error is encountered during playback, TestManager records the error in this file. If no errors occur, then there is no data in this file, and TestManager deletes the file automatically. For more information on test scripts, see the *Using Rational Robot* manual, or the Rational Test API documentation appropriate to the scripting language.

To view an error file:

- Right-click a virtual tester start event in the Test Log window, and click **View Virtual Tester Error File**.

To print an error file:

- From an open error file, click **File > Print**.

## Virtual Tester Output File

The virtual test output file contains any information a virtual tester specifically writes from test script output. This can be just about anything. For example, this file could log SQL commands.

**Note:** The output file does not always exist. If output is generated during playback, TestManager records the output in this file. If no output is generated, then there is no data in this file, and TestManager deletes the file automatically. For more information on test scripts, see the *Using Rational Robot* manual, or the Rational Test API documentation appropriate to the scripting language.

To view an output file:

- Right-click a virtual tester start event in the Test Log window, and click **View Virtual Tester Output File**.

To print an output file:

- From an open output file, click **File > Print**.

## Entering and Modifying Defects

A *defect* can be anything from a request for a new feature to an actual bug found in the application-under-test. Defect tracking is an important part of the software testing effort.

You can use the Test Log window of TestManager to enter defects for any verification points that fail during playback of a recorded script. When you enter a defect from the test log, TestManager opens the TestStudio defect form and fills in several fields for you with information from the log. (When you enter defects this way, TestManager does not actually start ClearQuest; it opens the defect form, which is part of ClearQuest.) You can also enter defects manually using ClearQuest, but none of the fields will be automatically filled in for you.

Once you have entered defects, you can use ClearQuest to review the data and decide upon further action.

During the course of developing your application, you can update the state of each defect to keep the information current with the development-test-repair cycle. You can then use ClearQuest's reporting options to retrieve current information about the defects being tracked and the overall progress of development. You can also send defect information to other members of your development team using ClearQuest's e-mail features. For information about working with ClearQuest, see the Rational ClearQuest Help.

**Note:** To use ClearQuest to store defects, an administrator must first set up the ClearQuest schema repository, and then create or attach a ClearQuest user database as part of a Rational project. For information, see the *Administering Rational ClearQuest* manual.

## About ClearQuest and Defect Tracking

ClearQuest is a change-request management system designed for the dynamic and interactive nature of software development. With ClearQuest, you can manage all of the change-request needs of software development—for example, enhancement requests, defect reports, and documentation modifications.

For your convenience, a specially designed schema for defect tracking is included with your software. In ClearQuest, the term *schema* refers to all attributes associated with a change-request database. This includes field definitions, field behaviors, the state transition tables, actions, and forms. For more information about ClearQuest schemas, see the Rational ClearQuest Help. If you associate a test case with a test input, the test input information appears automatically in the defect form.

**Note:** The version of ClearQuest that comes with Rational TeamTest differs from standard ClearQuest in one way: you cannot add a field to a ClearQuest schema. To add fields to a schema, you must purchase standard Rational ClearQuest software or a Rational Suite product.

### **About the Rational TestStudio Schema**

The *TestStudio schema* includes two TestStudio defect forms: one for entering new defects, and one for modifying and tracking defect information.

**Note:** To use the TestStudio schema, you must select it when you create or attach a ClearQuest user database as part of a Rational project. For information, see the *Using the Rational Administrator* manual.

### **About the TestStudio Defect Form**

You can use the TestStudio defect form to track as many or as few details about a defect as you want.

**Note:** To display information about each item in the defect form, right-click the item and click Help.

### **Entering Defects**

You can enter defects from the Test Log window of TestManager or ClearQuest. If you open a test log in TestManager, TestManager fills in many of the fields in the defect form. If you use ClearQuest, you must enter the fields manually.

To enter a defect from TestManager, do one of the following:

- Right-click the failed event in the **Event Type** column, and click **Submit Defect**.
- Click **Edit > Submit Defect**.

**Note:** TestManager attempts to connect to ClearQuest using your user name and password. However, if TestManager still cannot connect to the ClearQuest database, the Login dialog box appears. In this case, type your ClearQuest user name and password. Select the database in which you want to enter the defect.

If you enter a defect from the test log, the number of the new defect appears in the **Defect** column of the test log.

**Note:** You can also enter defects from SiteCheck, after you play back a Web verification point. From the test log, right-click the failed Web verification point and click **Submit Defect**. In SiteCheck, click **Tools > Enter a Defect**.

## Printing a Test Log

You can preview or print the information displayed in the active test log to analyze test results, as shown in this example:

Example of a print preview

Event Type	Result	Date & Time	Failure Rea...	Computer Name	Defect
Suite Start (Suite 1)	Fail	10/06/2000 10:48:34 ...	Executable...	cqrhs	
TestCase	Fail	10/06/2000 10:48:51 ...	Unknown		
TestCase	Fail	10/06/2000 10:48:58 ...	Unknown		
TestCase	Fail	10/06/2000 10:49:03 ...	Unknown		
User Start	Fail	10/06/2000 10:48:36 ...			
Script Start (Push Button)	Fail	10/06/2000 10:48:43 ...			
Application Start	Pass	10/06/2000 10:48:44 ...			
Verification Point (O...	Fail	10/06/2000 10:48:50 ...			
Script End (Push But...	Fail	10/06/2000 10:48:50 ...			
Script Start (Single Order)	Fail	10/06/2000 10:48:50 ...			
Application Start	Pass	10/06/2000 10:48:50 ...			
Verification Point (O...	Pass	10/06/2000 10:48:56 ...			
Verification Point (O...	Fail	10/06/2000 10:48:57 ...			
Script End (Single Or...	Fail	10/06/2000 10:48:57 ...			
Script Start (Single Order)	Fail	10/06/2000 10:48:57 ...			
Application Start	Pass	10/06/2000 10:48:58 ...			
Verification Point (O...	Pass	10/06/2000 10:49:02 ...			
Verification Point (O...	Fail	10/06/2000 10:49:02 ...			
Script End (Single Or...	Fail	10/06/2000 10:49:03 ...			
User End	Fail	10/06/2000 10:49:03 ...			
Suite End (Suite 1)	Fail	10/06/2000 10:49:04 ...	Executable...	cqrhs	

To print an active test log:

- 1 Display the log you want to print in the Test Log window.
- 2 Click **File > Print**.

To get more details in the report, click the (+) plus sign in the **Event Type** column. To reduce the number of details, click the minus sign (-) in the **Event Type** column.

## Managing Log Event Property Types

Managing log event property types allows you to specify which log event data is logged and how it is displayed in TestManager. When you add a new log event property type—depending on how it is defined—it displays in a separate tab in the Log Event window as text or as an HTML file, or in a separate application.

**Note:** Log event property types are related to the extensibility of TestManager. For custom defined test script types in testing, for example, you can define specific events to log. For more information, see the *Rational Test Extensibility Reference* manual.

Choosing to log a specific log event type can help you to identify how a particular event performs during a test. For example, if you are running a test against a web server, you could set up an event type that specifically logs HTTP requests on the web server.

TestManager supplies the default log event property type “Associated Data.” For more information on the information logged by this property type, see *Viewing Events Details* on page 143.

To create or edit a log event property type:

- Click **Tools > Manage > Log Event Property Types**. Click **New**.

Enter a property name.

Click **Internal viewer** and select **Text** or **HTML** to either add a tab to the Log Event Properties dialog (**Text**) or launch a Web browser displaying logged information (**HTML**).

Click **External viewer** to see logged data in the application of your choice.

Click to select a **Format type** to specify whether the data logged is the actual data or the reference to the data in a separate file.

The screenshot shows the 'New Log Event Property Type' dialog box. It has two tabs: 'General' and 'Statistics'. The 'General' tab is selected. The dialog contains the following fields and options:

- Name:** A text input field.
- Description:** A text area with a scroll bar.
- Owner:** A dropdown menu.
- Viewer type:** Two radio buttons:  Internal viewer and  External viewer. Below the 'Internal viewer' radio button is a dropdown menu showing 'Text'.
- Format type:** Two radio buttons:  Data and  File reference.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons at the bottom.

## Viewing Test Script Results Recorded with Rational Robot

---

You can use Rational Robot to record test scripts that contain verification points. After you play back the test script, Robot writes the results to a log. Certain verification points also have *baseline data files* that are saved. If a verification point fails during playback, *actual data files* are also saved. You can use the appropriate Comparators to view actual data or image files, and view and edit the baseline files as needed.

In addition to using the Test Log window to view the playback results of verification points, you can use it to view procedural failures, aborts, and any additional playback information.

A testing cycle can have many individual tests for specific areas of an application. Reviewing the results of tests in the Test Log window reveals whether each passed or failed. Analyzing the results in a Comparator helps determine why a test may have failed. Review and analysis help determine where you are in your software development effort and whether a failure is a defect or a design change.

### Viewing a Verification Point in the Comparators

In the **Details** page of the Test Log window, failed events are indicated in red in the **Result** column. If the event is a failed verification point of a script created using Robot, you can analyze the failure using one of the Comparators.

To view a verification point in a Comparator:

- 1 Open a test log.
- 2 Click the **Details** tab.
- 3 Right-click a verification point and click **View Verification Point**.

The appropriate Comparator opens based on the type of verification point, as shown in the following table. You can then analyze the results to determine whether the failure was caused by a defect or an intentional change in the application.



Comparator	Verification points
Text Comparator	Alphanumeric
Grid Comparator	Object Data Menu Clipboard
Image Comparator	Window Image Region Image
Object Properties Comparator	Object Properties

**Note:** Rational QualityArchitect uses the Grid Comparator to display verification point information.

For more information about the four Comparators, see Chapter 9, *Using the Comparators*.

Failure indications in test logs do not necessarily mean that the application-under-test has failed. You need to evaluate each verification point failure with the appropriate Comparator to determine whether it is an actual defect, a playback environment difference, or an intentional design change made to a new build of the application-under-test.

## Playback/Environmental Differences

Differences between the recording environment and the playback environment can generate failure indications that do not represent an actual defect in the software. This can happen if there are applications or open windows in the recorded environment that are not in the environment, or vice versa.

For example, if you have the Calculator open in the recorded environment but not open in the playback environment, Robot can generate a failure that has nothing to do with the software that you are actually testing.

You should analyze these failure indications with the appropriate Comparator to determine whether the window that Robot could not find is an application window that should have opened during the script playback or an unrelated window.

## Intentional Changes to an Application Build

Revisions to the application-under-test can generate failure indications in scripts and verification points developed using a previous build as the baseline. This is especially true if the user interface has changed.

For example, the Window Image verification point compares a pixel-for-pixel bitmap from the recorded baseline image file to the current version of the application-under-test. If the user interface changes, the Window Image verification point will fail. When intentional application changes result in failures, you can easily update the baseline file to correspond to the new interface using the Image Comparator. Intentional changes in other areas can also be updated using the other Comparators.

For information about updating the baseline, see Chapter 9, *Using the Comparators*.

## Reporting Results

---

### About Reports

TestManager provides you with a set of standard reports that you can use to analyze test case results. In addition to these standard reports, you can customize reports to your needs by editing the design layouts. You can also use queries to narrow down the data displayed in a report.

TestManager provides three types of reports to help you in your testing efforts:

- Test case distribution, test case results distribution, and test case trend reports.
- Listing reports.
- Performance reports.

### About Test Case Distribution and Trend Reports

Test case distribution and trend reports help you track the progress of your planning, implementation, and execution of test cases. You can run a report to find out who is testing a particular component or what percentage of test cases have been executed. These reports have multiple display formats including pie, bar, line, and tree charts.

TestManager includes three types of test case reports:

- **Test Case Distribution** reports provide information about the number of test cases created, who created them, configuration information, and whether they have been implemented manually or with an automated test script. You can also see the number of test cases implemented versus those planned. Test case distribution reports can be especially useful during the test planning phase of a project.
- **Test Case Results Distribution** reports provide crucial information about the quality of a specific build and the progress of your ability to test that build. These reports tell you the number of test cases that have passed results, failed results, warnings

results, and informational results, and that were stopped or completed for a specific build. They can also tell you the number of test cases implemented, test cases executed, and the test case instances executed. (Instances of test cases can be added to performance testing suites. See Chapter 11, *Creating Performance Testing Suites*.)

- **Test Case Trend** reports provide information about the number of test inputs, and test cases that have been planned, developed, executed, or met the testing criteria over several builds, iterations, or dates.

## About Listing Reports

Listing reports display lists of the different test assets stored in a Rational project. TestManager includes listing reports for builds, computers, computer lists, configurations, iterations, suites, sessions, test logs, test plans, test scripts, and virtual testers.

Each listing report comes with one or more design layouts that you can run without changing. A design layout defines the look of each report and the specific information included in a listing report. You can also customize the design layout or create new design layouts using Crystal Reports. For information, see *Customizing Design Layouts for Listing Reports* on page 155.

By using different combinations of layouts and listing reports, you can create a wide variety of ready-to-run reports.

For example, using the available design layouts and listing reports, you can create a test script listing report that:

- Lists the details of all of the test scripts in your project.
- Summarizes all of the test scripts in a project.

You can create a query to specify which data to include in a listing report. For information about creating a query, see the Crystal Reports Help.

## Customizing Design Layouts for Listing Reports

A design layout defines the look of each report and the specific information included in a listing report. To customize existing design layouts, or create a new design layout, you must install Crystal Reports 8.0 Professional Edition. The Crystal Reports software comes in a separate CD-ROM in your Rational software kit.

When you create a new listing report in TestManager, you can optionally create new or customize existing design layouts. Crystal Reports uses report dictionaries of assets and properties stored in the Rational Test datastore. These dictionaries link the various assets together using the database schema.

For more information about using Crystal Reports to create new or customize existing design layouts, see the Crystal Reports Help.

## About Performance Testing Reports

Performance testing reports help you analyze the relative success or failure of a given suite run, and the performance of the server under specified conditions. For example, you can determine how long it took for a virtual tester to execute a command, and how response times varied with different suite runs. You can also define custom reports based on standard report types. These custom reports can help you zoom in on a given application element and further refine tests to show exactly the data you need as determined by your test plan or test case.

Performance testing reports include:

- Performance reports.
- Compare Performance reports.
- Response vs. Time reports.
- Command Status reports.
- Command Usage reports.

For detailed information about performance testing reports, see Chapter 13, *Reporting Performance Testing Results*.

## Selecting Which Reports to Use

The following table summarizes the types of TestManager reports.

To	Use this report	For information, see
Categorize test cases by a particular property. (For example, you can view how many test cases are in each iteration or how many test cases were created by people in a particular testing group.)	Test Case Distribution	<i>About Test Case Distribution and Trend Reports</i> on page 154
Determine the number of test cases that meet your test criteria.	Test Case Results Distribution	<i>About Test Case Distribution and Trend Reports</i> on page 154

Determine the percentage of test cases planned, implemented, or executed for several builds, iterations, or dates: to view the percentage of test inputs tested, not tested, satisfied, or not satisfied for several builds, iterations, or dates.	Test Case Trend	<i>About Test Case Distribution and Trend Reports on page 154</i>
List the builds in your project.	Build Listing	<i>About Listing Reports on page 155</i>
List the computers in your project.	Computer Listing	<i>About Listing Reports on page 155</i>
List the list of computers in your project.	Computer List Listing	<i>About Listing Reports on page 155</i>
List the configurations in your project.	Configuration Listing	<i>About Listing Reports on page 155</i>
List the iterations in your project.	Iteration Listing	<i>About Listing Reports on page 155</i>
List the sessions in your project.	Session Listing	<i>About Listing Reports on page 155</i>
List the suites in your project.	Suite Listing	<i>About Listing Reports on page 155</i>
List the test logs in your project.	Test Log Listing	<i>About Listing Reports on page 155</i>
List the test plans in your project.	Test Plan listing	<i>About Listing Reports on page 155</i>
List the test scripts in your project.	Test Script Listing	<i>About Listing Reports on page 155</i>
List the users in your project.	User Listing	<i>About Listing Reports on page 155</i>
Display the response times, and calculate the mean, standard deviation, and percentiles for each command in the suite run.	Performance	<i>Chapter 13, Reporting Performance Testing Results</i>
Compare the response times measured by several Performance reports.	Compare Performance	<i>Chapter 13, Reporting Performance Testing Results</i>
Display individual response times and whether a response has passed or failed.	Response vs. Time	<i>Chapter 13, Reporting Performance Testing Results</i>

Obtain a quick summary of which commands passed or failed.	Command Status	Chapter 13, <i>Reporting Performance Testing Results</i>
View cumulative response time and summary statistics, as well as throughput information for emulation commands for all test scripts, and for the suite run as a whole.	Command Usage	Chapter 13, <i>Reporting Performance Testing Results</i>

## Additional Reports

Additional reports are available in Rational ClearQuest and Rational SoDA.

You can use ClearQuest reports, as well as design layouts, queries, and charts to help you manage your defect database. These reports and other items are automatically created for you when you create a project that contains an associated ClearQuest database. For information about using these defect reports see the ClearQuest Help. For information about creating a project, see the *Using the Rational Administrator* manual.

You can also create reports using Rational SoDA. Rational SoDA is a report generation tool that supports reporting as well as formal documentation requirements. With SoDA you can retrieve information from different information sources, such as Rational Rose and Rational RequisitePro, to create a single document or report. For information about creating reports using Rational SoDA, see the SoDA Help. To use SoDA, click **Reports > SoDA Report**.

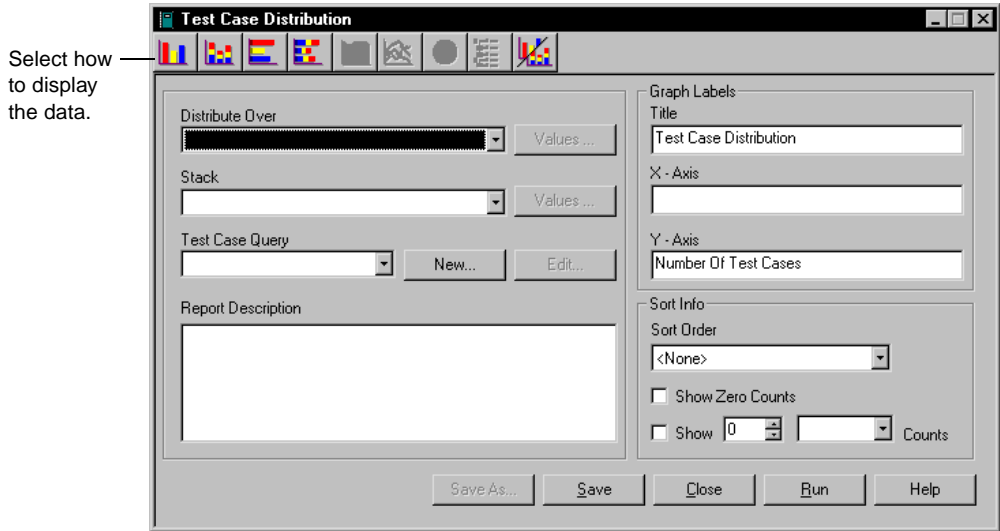
## Creating Reports

To create a report:

- Click **Reports > New**, and then select the type of report you want to create.

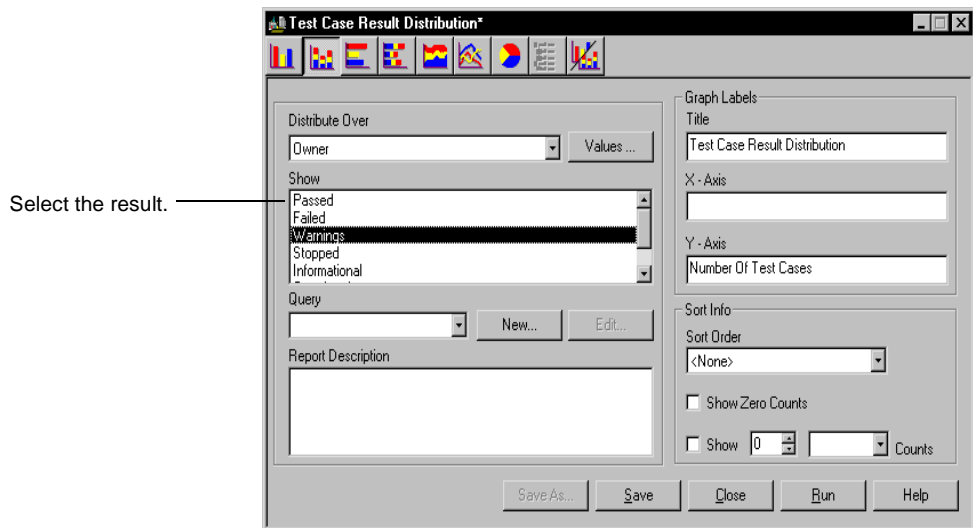
### Creating a Test Case Distribution Report

When you create a test case distribution report, you can select how the data appears: either in bar, stack, line, pie, or tree reports depending on the type of report you select.



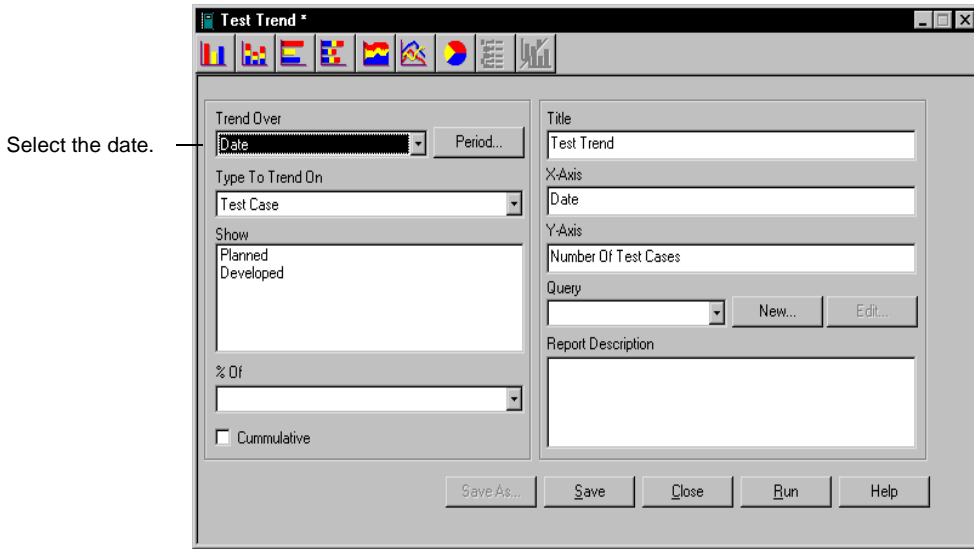
## Creating a Test Case Results Distribution Report

When you create a test case results distribution report, you select the test case results that you want in a report:



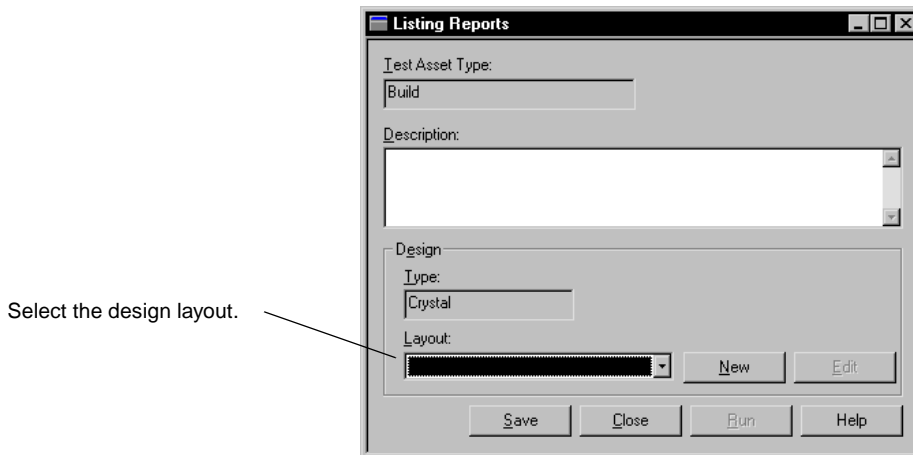
## Creating a Test Case Trend Report

When you create a test case trend report, you select the information you want about test cases or test inputs over several builds, iterations, or dates.



## Creating a Listing Report

When you create a listing report, you determine how you want the information to appear by choosing a Crystal Reports design layout. You can create new or customize existing Crystal Reports design layouts. For more information, see *Customizing Design Layouts for Listing Reports* on page 155.





## Creating Performance Reports

When you create performance reports, you can specify the log data on which to run the report and how to manipulate the log data so that you see just the information you need. For detailed information about creating performance reports, see Chapter 13, *Reporting Performance Testing Results*.

## Opening a Report

After creating a report, you can open it and, if necessary, make changes to the report.

To open or change a report, do one of the following:

- Click **Reports > Open**, select a report from the list, and then click **OK**.
- In the **Analysis** tab of the Test Asset Workspace select the type of report you want to open. Select the particular report you want to open or change.

For more details about opening a report, see the TestManager Help.

## Running Reports

You can run reports from:

- The Report Bar.
- The Report menu.

### Running a Report from the Report Bar

**Note:** You can run only Performance Testing reports from the Report bar.

The quickest way to run a report is to click its name on the Report bar. On the Report bar, TestManager displays the log of the last suite you ran. Unless you specify another log, TestManager runs the report using the information in this log.

To run a report from the Report bar:

- If the Report bar is not open, click **View > Report Bar**. Click any one of the report buttons.

**Note:** You can customize the Report bar by populating it with your own reports. For more information, see *Changing the Reports that Run from the Report Bar* on page 162.

## Running a Report from the Menu Bar

Although TestManager lets you run reports quickly from the Report bar, you can run only one report of each type against a log using this method. You may want to run a number of reports from a series of logs. (For example, if you have defined some new Performance reports, and you want to run each report against the same log.) Run these reports from the menu bar.

To run a report from the menu bar:

- Click **Reports > Run**, and select the type of report to run.

## Changing the Reports that Run Automatically

**Note:** You can run only Performance Testing reports automatically after a suite run.

TestManager automatically displays Performance and Command Status reports at the end of the suite run. However, you can change the reports that TestManager displays.

To change the reports that TestManager displays at the end of a suite run:

- Click **Tools > Options**, and then click the **Reports** tab.

## Changing the Reports that Run from the Report Bar

**Note:** You can run only Performance Testing reports from the Report bar.

The Report bar lets you run reports by pressing a button. This bar automatically runs the default reports unless you specify otherwise. For example, you may have defined a new report that you want to run instead of the default reports.

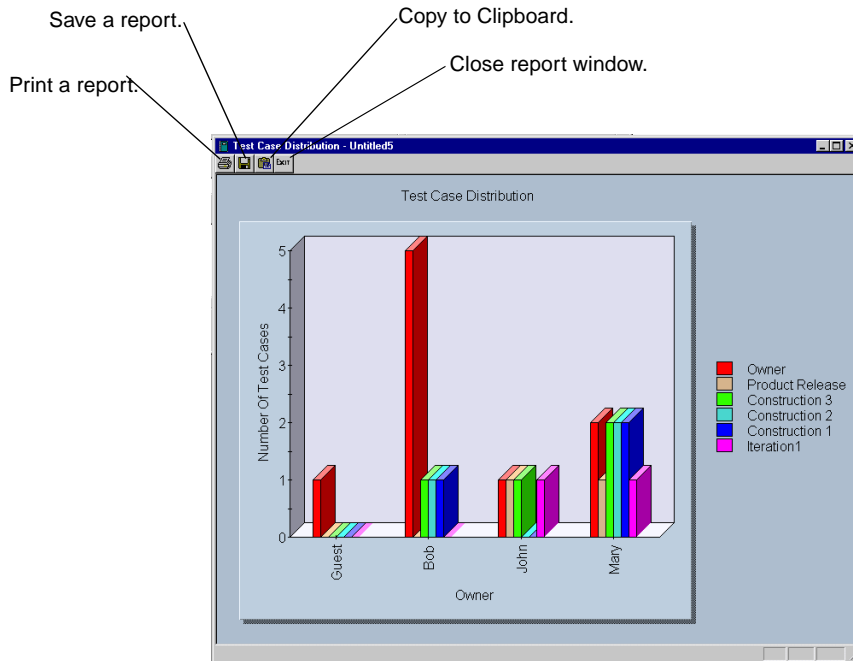
To specify the reports that TestManager runs from the Report bar:

- Click **Tools > Options**, and then click the **Reports** tab.

**Note:** To reset the Report bar so that it generates the default reports, click **Tools > Options**, click the **Reports** tab, and then click the **Reset Report Bar** button.

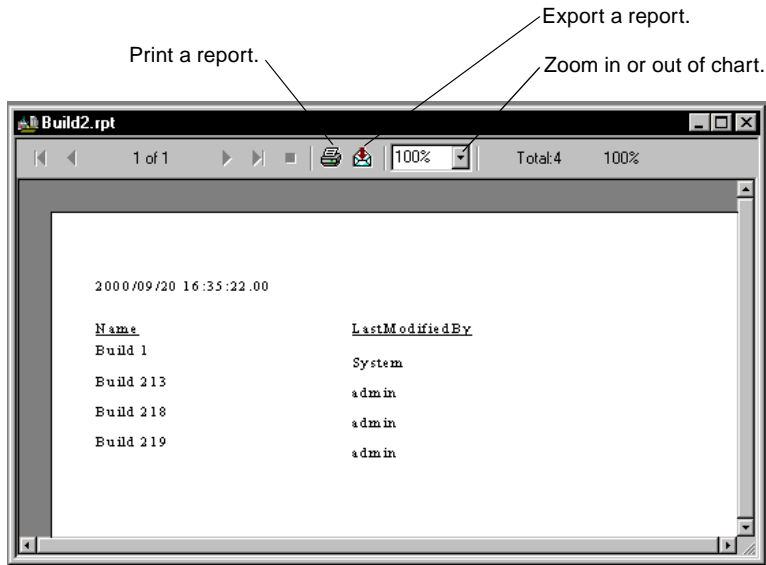
## Print, Save, or Copy a Test Case Trend or Distribution Report

After you run a test case trend or distribution report, you can print, save, or copy it to the Clipboard.



## Print, Export, or Zoom a Listing Report

After you run a listing report, you can print it or export it to a different file format and save it on your computer. You can export a finished report to a number of popular spreadsheet and word processor formats, as well as to HTML, ODBC, and common data interchange formats. This makes it easy to distribute information. For example, you may want to use the report to project trends in a spreadsheet or to mail to other members of your testing team.



## Print, Save, Copy, Delete, or Export a Performance Report

TestManager allows you to perform common management tasks with reports. You can print, copy, rename, delete, and export performance reports as necessary. For more information on these administrative tasks, see Chapter 13, *Reporting Performance Testing Results*.

## Copying Reports to a New Project

If you create a report or a new design layout and want to use it in a new project, use the Rational Administrator to copy them when you create a new project. The Rational Administrator copies any saved listing reports and listing design layouts to the new project.

For information, see the Administrator Help.

## Creating a Query

A query is a request for specific information from a Rational Test datastore. You can create a query for each type of TestManager report.

## Queries for Test Case Distribution, Test Case Trend, and Performance Reports

TestManager provides pre-defined queries to narrow down the data in test case distribution, test case trend, and performance reports. You can edit the existing queries and create your own queries for these reports.

To create a query, do one of the following:

- Create or open a report, and click the **New** button next to the **Query** field.
- Click **Tools > Manage > Queries > Test Case**.

## Queries for Listing Reports

To create a query for a listing report, you must install Crystal Reports 8.0 Professional Edition. The Crystal Reports software comes in a separate CD-ROM in your Rational software kit. For more information about creating a query for listing reports, see the Crystal Reports Help.



# **Part 2: Functional Testing with Rational TestManager**





This chapter includes the following topics:

- Planning functional tests
- Distributed functional testing
- Recording considering for functional tests

## Planning Functional Tests

---

In the simplest sense, a functional test determines whether the software functions as designed.

### Identifying Functional Testing Requirements

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

- Server computers – The server computers that will be accessed
- Client computers – The characteristics of the computers that run the application (for example, processor speed, memory, and disk space)
- Databases that will be accessed
- Applications to be tested

### Setting Pass and Fail Criteria for Functional Tests

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

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

- The same error recovery as the baseline

Typically, changes to software can be classified in one of two ways:

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

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

## Distributed Functional Testing

---

With TestManager, you can perform functional tests in distributed mode, meaning that you can have many computers running concurrently. This enables you to:

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

### Distributing Tests Among Different Computers

With functional testing, you might want to run your tests immediately on any computer that is available. In this case, follow these steps:

- When you insert user groups into a suite, click the **Multiple Computers** button and add your computers to the computer list that appears. For more information about setting up test scripts to run on different computers, see *Inserting User Groups into a Suite* on page 246.
- After you have inserted your user groups, insert a Parallel selector. The scripts that you insert under the selector will be continuously sent out to the next available computer. Of course, the scripts must be designed so that they are self-contained and do not rely on one another. For more information about the parallel selector, see *Inserting a Selector* on page 257.

### Running Tests on a Specific Computer

If you are testing functionality on a group of computers that have a variety of hardware or software, you need to set up the user groups to run on a particular computer. For information, see *Inserting User Groups into a Suite* on page 246.

## Example of a Distributed Functional Test

In the following example, assume that you want to test your Accounting software. You want to distribute your tests over different computers so that they can run as quickly as possible.

The following table summarizes how you set up this test.

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

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

## Recording Considerations for Functional Tests

---

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

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

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

To avoid this problem, you can use a *datapool* to supply the data values to your test script. Typically, you use a datapool so that each virtual tester that runs the test script can send realistic data to the server. You can also use a datapool so a single virtual tester that performs the same transaction multiple times can send realistic data to the server in each transaction. If you do not use a datapool, each virtual tester would send the same values to the server.

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

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

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

**Note:** For more information about datapools, see *Working with Datapools* on page 279. Additionally, see the following Rational documents:

- For datapool procedures, see the Rational TestManager Help.
- For information about using datapools in VB or Java test scripts see the appropriate Rational Test Script Services API documentation.
- For information about datapools in custom test script types, see the *Rational TestManager Extensibility Reference* manual.
- For more information about creating datapools during test script recording, see *Using Rational Robot* and Robot Help.
- For information about datapools and GUI test scripts see the *SQABasic Reference*.manual.

This chapter describes how to design functional testing suites. It includes the following topics:

- About suites
- Creating a suite
- Inserting computer groups into a suite
- Inserting test scripts into a suite
- Inserting other items into a suite
- Advanced functional testing features
- Using events and dependencies to coordinate execution
- Executing suites

## About Suites

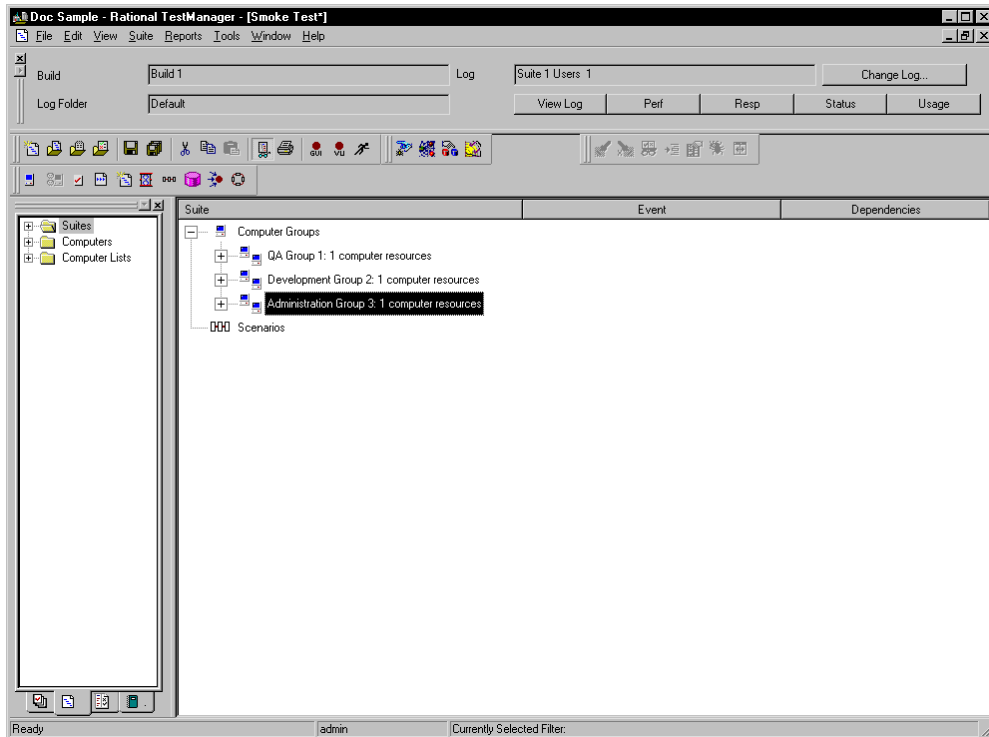
---

A suite shows a hierarchical representation of the tasks that you want to test. It shows such items as the computer groups, resources assigned to each computer group, which test scripts the computer groups run, and how many times each test script runs.

Through a suite, you can:

- Run test scripts.
- Group test scripts to emulate the actions of virtual testers.
- Set the order in which test scripts run.
- Synchronize virtual testers.

The following figure shows a suite with three computer groups: QA Group 1, Development Group 2, and Administration Group 3.



The examples of suites in this chapter show GUI test scripts. A suite, however, can contain GUI scripts, VU scripts, VB scripts, or other user-defined test script types. For more information on defining other test script types and using them in TestManager, see the *Rational TestManager Extensibility Reference*.

## Creating a Suite

---

A suite enables you to not only run test scripts, but more importantly, to emulate the actions of virtual testers using a system. A suite can be as simple as one virtual tester executing one test script, or as complex as hundreds of virtual testers in different groups, with each group executing different test scripts at different times using different resources.

You can create a suite in several different ways. For example, you can create a suite:

- Using the performance testing suite wizard
- Using the functional testing suite wizard
- Based on an existing suite of any type
- Based on an existing Robot session
- Using a blank performance testing suite
- Using a blank functional testing suite

To create a new suite using any of these methods:

- Click **File > New Suite**.



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

## About Creating a Suite from a Wizard

If you are new to testing, using the suite wizards may be the easiest and fastest way to create a working suite. Each wizard guides you through the process of creating a suite.

When you create a suite using the functional testing wizard, TestManager helps you choose test cases and scripts that become the basis for the test.

When you create a suite using the performance testing wizard, TestManager helps you choose the computer on which the test will run and helps you associate scripts that become the basis for the test.

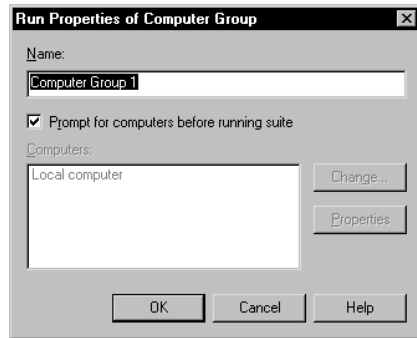
## Inserting Computer Groups into a Suite

---

A *computer group* is the basic building block for all functional testing suites. A computer group is one or more computers (or computer lists) running the same the test scripts and therefore testing the same application. For example, the suite on page 174 contains three computer groups: QA Group 1, Development Group 2, and Administration Group 3.

To insert a computer group into a suite:

- Click **Suite > Insert > Computer Group**.



**Note:** The name of a computer group cannot be identical to the name of a shared variable, a test script, or the following reserved words: MASTER, ALL, ASSIGN, TO, THRU, END, UNION, DELAY, delay, shared, SHARED, SYNC, DLB\_FREQ, DLB\_TIME, PERMUTE, TSIDX, CIDX, TC\_START, TC\_END.

When you add a computer group to a suite, you must specify one of the following: whether the group is associated with specific computers or computer lists, or whether to have TestManager prompt for available computer resources at suite runtime.

If you choose to have TestManager prompt for resources at runtime, you have the flexibility of specifying the computers available when you run the suite. Conversely, if you specify a computer or computer list for a computer group, and that resource is unavailable at suite runtime, the suite cannot run.

**Note:** When adding computer groups to suites, you can specify either one computer group for which to prompt for resources at runtime, or numerous computer groups to prompt for resources at runtime. You cannot mix the following within a suite: computer groups with specific resources, and computer groups without specific resources.



If you specify that a computer group is associated with a computer or computer list, you must specify those resources when you create the group. The default computer is the TestManager Local computer, but you can specify that the computer group runs on any defined Agent computer.

Typically, you run the computer group on an Agent computer if a functional test is designed for a particular computer with a particular configuration.

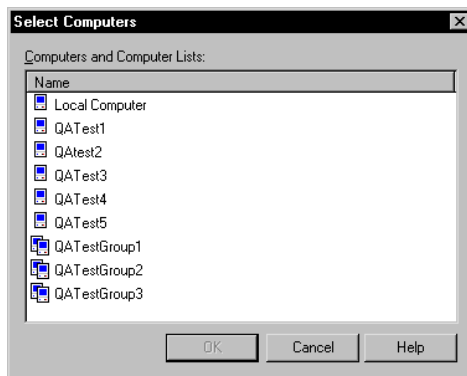
For information on adding computers and computer lists to TestManager, see Chapter 4, *Implementing Tests*.

**Note:** Copy any custom-created external C libraries, Java class files, or COM components necessary for the test to the Agent computer.

Typically, you run a computer group on multiple computers if you have a functional test that must execute as quickly as possible. You can save time by running virtual testers simultaneously on different computers.

To distribute the virtual testers in a computer group over multiple computers:

- Click **Suite > Insert > Computer Group**, and then click **Change**.



## Inserting Test Scripts into a Suite

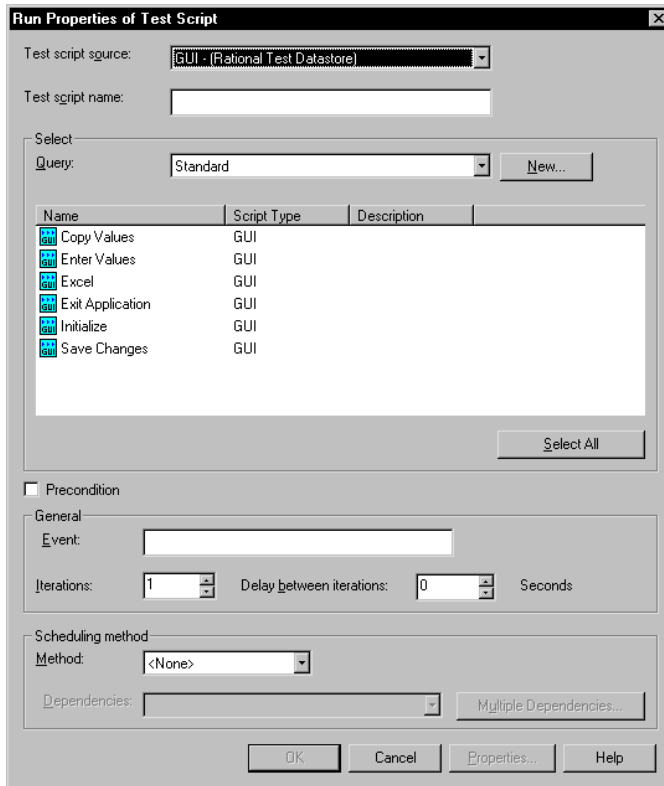
---

After you insert a computer group into a suite, you add the test scripts that the computer group should run. The computer groups in the suite on page 174 must have GUI test scripts associated with them.

Any test scripts that model the behavior you are testing are valid additions to a suite. You can mix test script types in computer groups with one exception: you cannot mix GUI and VU scripts in one computer group. You can, however, mix such test scripts in a suite by placing them in different computer groups.

To insert a test script into a suite:

- From an open suite select the computer group to run the test script, and then click **Suite > Insert > Test Script**.



## Preconditions

When you specify a test script, suite, or test case to be included in a suite, you can specify that successful completion of the test script, suite, or test case is a *precondition* for the remainder of that suite sequence. This means that the test script, suite, or test case must complete successfully for subordinate items in the suite sequence to run.

To set a precondition on a test script, suite, or test case:

- Right-click the test script, suite, or test case to which to apply the precondition, and select **Run Properties**.

Since suites can be complex and contain subordinate suites, test cases, and user groups, preconditions apply to their immediate sequence of events. For example, suppose you have a suite that includes two subordinate suites, each of which contains an initialization type of script (logging on to a network, for example) and several test cases. If, in the first suite, a precondition is applied to an initialization script and the script fails, TestManager skips all remaining actions (test cases) within that subordinate suite *only*. The suite resumes at the beginning of the next suite (or whatever is next in the larger suite).

Although preconditions are most commonly applied to test scripts, they can also be applied to test cases and suites within a suite. The precondition property applies only to the specific instance of the test script, test case, or suite. If a test script is used multiple times within a suite, preconditions must be set for each instance of the test script individually.

Unlike events or dependencies, when a precondition is applied to a test script, suite, or test case, that test script, suite, or test case *must* pass for subsequent items in that section of the suite to continue.

Preconditions on test scripts, test cases, and suites can be used to ensure that the precondition of a test case is met correctly. For more information on test case preconditions and postconditions, see Chapter 3, *Designing Tests*.

## Inserting Other Items into a Suite

---

A suite requires only computer groups and test scripts to run. However, a suite that realistically models the work that actual virtual testers perform is likely to be more complex and varied than this simple model. A realistic suite might also contain test cases, subordinate test suites, and scenarios. Advanced functional tests could also include scenarios, delays, and synchronization points to represent a variety of virtual tester actions.

### Inserting a Test Case into a Suite

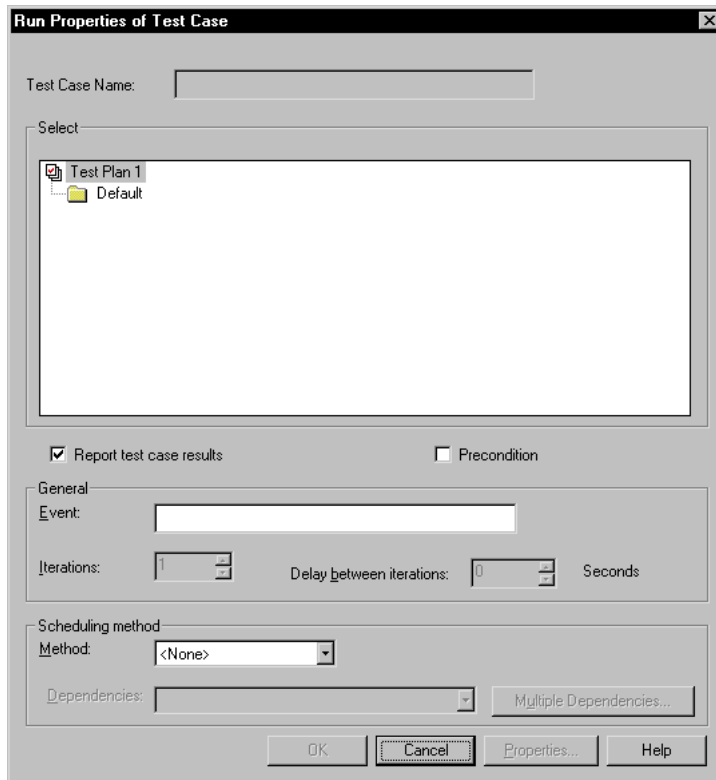
As discussed in Chapter 1, a *test case* is a testable and verifiable behavior in a target test system. It can include:

- Test inputs—the defined test requirement, possibly including Rational RequisitePro documents, Rational Rose models, or other kinds of items.
- Execution conditions—where, what, and how the input is tested, such as the operating system on the target computer.
- Expected results—the actual behavior to be verified.

The behavior can be as varied as a simple mouse click or a combination of server response times.

To insert a test case into a suite:

- Click **Suite > Insert > Test Case**.



A test case can be considered *configured* depending on its properties.

- Test cases define a behavior to be verified in the system. Test cases that are not configured are more flexible as they are not system-dependent; they can be run on a system with any configuration.
- Configured test cases not only define a behavior to be verified in the system, but also specify the setup of the system on which the behavior will be verified. Configured test cases are more specific; for the test criteria to be met and verified, the system on which the test case is run must exactly match the defined configuration.

Test cases can be included in suites for a number of reasons. Using a test case as a building block lets you create a test that can be used and applied in a variety of different ways depending on the resources specified at runtime. This can be useful for a set of test cases run on a regular basis. When you include configured test cases in suites, TestManager pairs available systems with matching configurations for you at runtime. Thus, different configured test cases may run each time depending on system availability, simulating variations and randomness in system use.

Preconditions can be applied to test cases. For information on preconditions, see *Preconditions* on page 178.

To set a precondition on a test case:

- Right-click the test case to which to apply the precondition and select **Run Properties**.

## Inserting a Suite

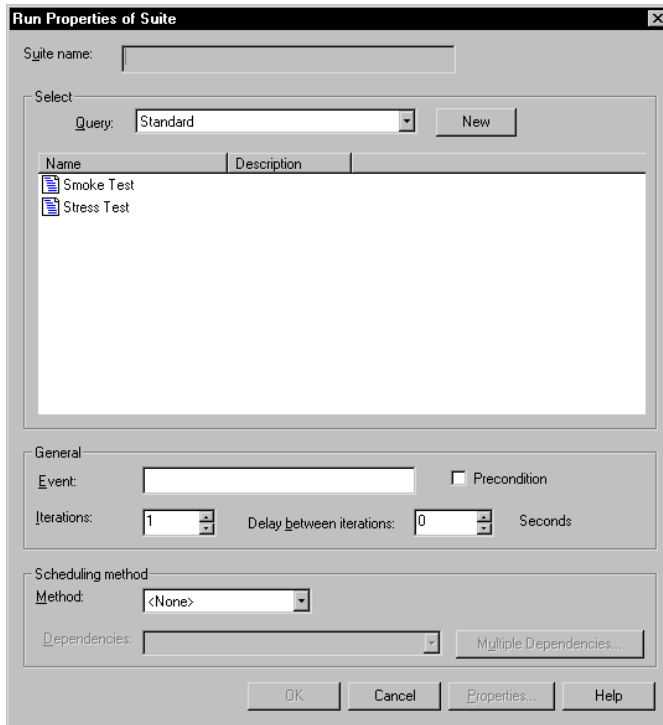
For maximum flexibility and power, TestManager allows you to insert complete computer-based suites into other suites. This allows you to use suites as building blocks of tests just as you would any other suite item.

**Note:** You cannot place a user-based suite into another suite. In addition, suites placed into a suite must have been created with the **Prompt for resources** option selected.

Using suites as building blocks is particularly helpful when you are creating a large, complex tests, or when you are creating multiple tests that perform several duplicate functions. You can create and check a smaller suite, then insert it to any other suite. You save time by not having to redefine the same test assets in each separate suite. Any change made to a suite is replicated in every instance of that suite.

To insert a suite into a suite:

- Click **Suite > Insert > Suite**.



Preconditions can be applied to suites. For information on preconditions, see *Preconditions* on page 178.

To set a suite as a precondition:

- Right-click the suite to which to apply the preconditions, and then select **Run Properties**.

## Inserting a Selector

TestManager allows you to set suite items to run in different sequences by setting a *selector*. A selector provides more sophisticated control than running a simple sequence of consecutive items in a suite. A selector tells TestManager which items each virtual tester executes, and in what sequence. For example, you might want to repeatedly select a test script at random from a group of test scripts. A selector helps you to do this.

To insert a selector into a suite:

- Select the computer group or a scenario that will contain the selector, and then click **Suite > Insert > Selector**.



## Types of Selectors

TestManager provides the following types of selectors:

- **Sequential** – Runs each test script or scenario in the order in which it appears in the suite. This is the default.
- **Parallel** – Distributes its test scripts or scenarios to an available virtual tester (one virtual tester per computer). This is the type of selector used most often in functional testing. The items are parceled out in order, based on which virtual testers are available to run another test script. Once an item runs, it does not run again.

A parallel selector distributes each test script without regard to its iterations.

- **Random with replacement** – The selector runs the items under it in random order, and each time an item is selected, the odds of it being selected again remain the same. Random with replacement selectors are rarely used in functional testing.
- **Random without replacement** – The selector runs the items under it in random order, but each time an item is selected, the odds change.
- **Dynamic load balancing** – With dynamic load balancing, items are not selected randomly. Items are selected to balance the workload according to the weight that you have set. You can balance the workload either for time or for frequency. Dynamic load balancing selectors are rarely used in functional testing.

## Advanced Functional Testing

---

While computer groups, test scripts, suites, test cases and selectors provide most of the features you need during functional testing, you can use scenarios, delays and synchronization points to do additional fine tuning of your suites.

### Inserting a Scenario

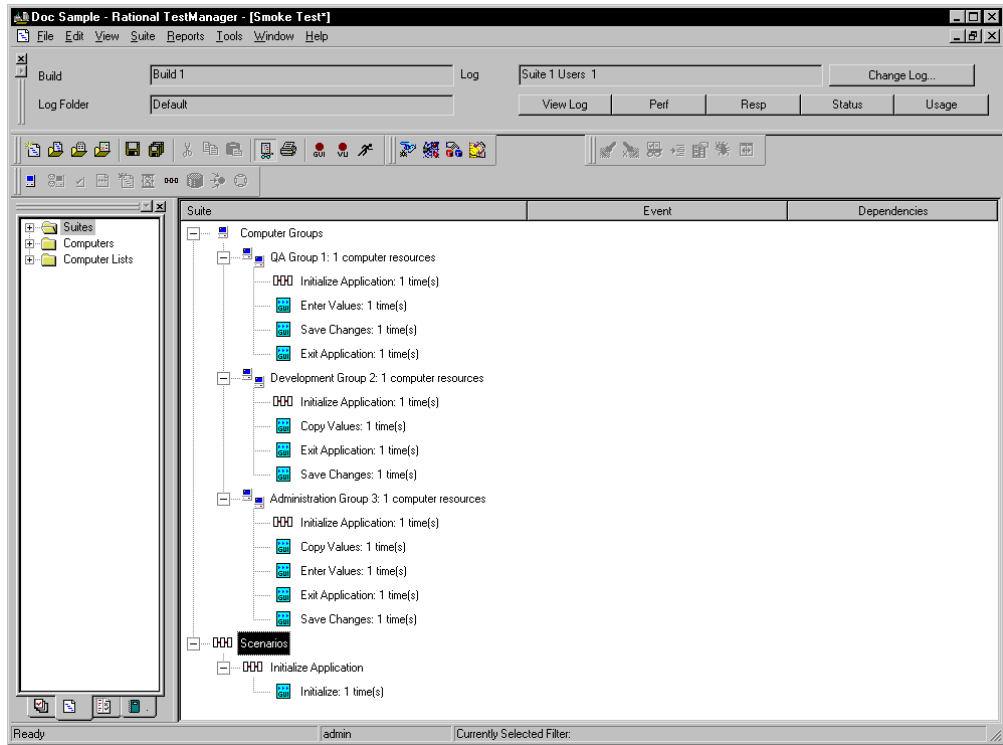
A *scenario* lets you group test scripts together so they can be shared by more than one computer group. If you have a complicated suite that uses many test scripts, grouping the test scripts under a scenario has the added advantage of making your suite easier to read and maintain.

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

In the following suite, all three computer groups run the test scripts needed to initialize the application before testing various parts of it. You can simplify this suite by storing the required initialization test script in a scenario. The suite shows the test

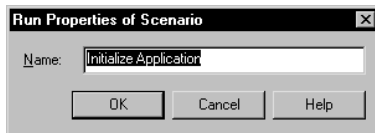


script Initialize as part of the Initialize Application scenario. A delay could be added to this scenario after the test script is run, and that change would filter to all instances of the Initialize Application scenario.



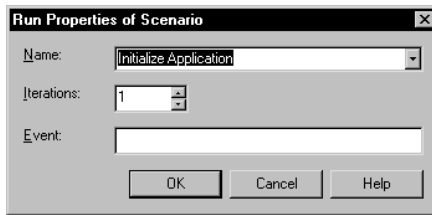
To create a new scenario:

- From the Scenarios section of the suite, click **Suite > Insert > Scenario**.



To insert a scenario into a suite:

- Click where you want to place the scenario, then click **Suite > Insert > Scenario**.



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

## Suite or Scenario?

The results of inserting a suite or scenario into a suite are similar. But each has advantages and disadvantages.

Use a suite when:

- You want to reuse a series of events in a variety of suites, and you want to be sure that any change made to the suite filters to all instances of it. Suites are reusable among different suites.

Use a scenario when:

- You want to reuse a series of events in a suite, and you want to be sure that any change made to that scenario filters to all instances of it within a suite. Scenarios are not reusable among different suites.

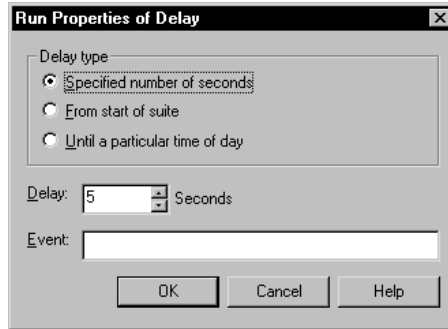
For example, you could create three suites, each testing a different aspect of an accounting application: one tests creating opening and editing spreadsheets, one tests all the menus, and one tests complex formulas within the spreadsheet. Each suite needs to have virtual testers open the application to perform their tasks. Yet within each suite, tasks unique to the suite need to be repeated. You could use a suite for opening the application that could be inserted in each suite, but within each suite, use a scenario for the repeated functions unique to the suite.

## Inserting a Delay

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

To insert a delay into a suite:

- Click the computer group, scenario, or selector to which to add a delay, and then click **Suite > Insert > Delay**.



In functional testing, you use delays to cause test scripts to wait before executing. For example, if one virtual tester updates a record, you can insert a delay to give the application time to process and display the correct information. By providing a delay, you ensure that the application has enough time to complete a task, in case another virtual tester must perform an action as a result of that task.

You can insert a delay into a suite or a test script. The advantages of inserting a delay into a suite are that the delay is visible in the suite and the delay is easy to change without editing the test script.

## Inserting a Synchronization Point

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

- All virtual testers associated with the synchronization point arrive at the synchronization point.

When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. When the specified number of virtual testers reaches the synchronization point, TestManager releases the virtual testers and allows them to continue executing the suite.

- A timeout period is reached before all virtual testers arrive at the synchronization point.

When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. Other testers arrive at the synchronization point and wait. However, before all virtual testers arrive at the synchronization point, the timeout period expires and TestManager releases the virtual testers and allows them to continue executing the suite. Virtual testers that did not make it to the synchronization point before the timeout expired do not stop at the synchronization point. They also continue executing the suite.

- You manually release the virtual testers while monitoring the suite.

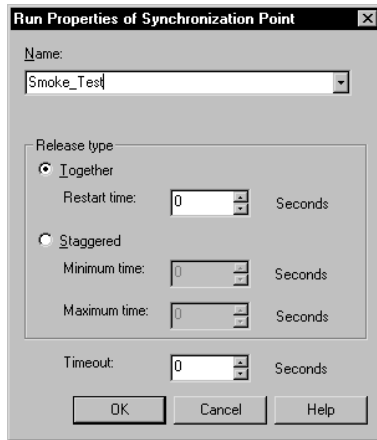
When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. Other testers arrive at the synchronization point and wait. However, this time you decide to release virtual testers from the synchronization point and continue executing the suite. All virtual testers may or may not have arrived at the synchronization point. Virtual testers that did not make it to the synchronization point before you released them manually do not stop at the synchronization point. They also continue executing the suite.

Insert a synchronization point into a suite through TestManager to:

- Pause execution before or between test scripts rather than within a test script. Inserting a synchronization point into a suite offers these advantages:
  - You can easily move the location of the synchronization point without having to edit a test script.
  - The synchronization point is visible within the suite rather than hidden within a test script.
- Specify how the virtual testers are released from the synchronization point. For example:
  - Specify whether you want the virtual testers to be released at the same time or at different times.
  - Specify the minimum and maximum times within which all virtual testers are to be released if the virtual testers are to be released at different times (staggered).
  - Specify a timeout period.

To insert a synchronization point into a suite:

- Click **Suite > Insert > Synchronization Point**.



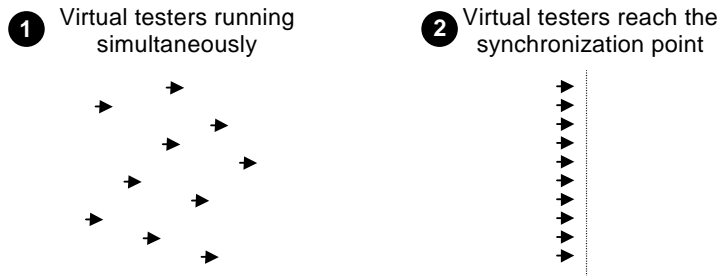
Use a synchronization point in a functional test, for example, when two groups of functional testers work together to test an application. Since both groups must be appropriately logged on to an application before testing can start, add a synchronization point that checks for this.

### How Synchronization Points Work

At the start of a test, all virtual testers begin executing their assigned test scripts. They continue to run until they reach the synchronization point. When specified in a test script, a synchronization point is a programmatic command (`sync_point` in a VU test script, `SQASyncPointWait` in a SQABasic test script, `TSSSync.SyncPoint` in a VB test script, or `TSSSync.syncpoint` in a Java test script). When specified in a suite, a synchronization point is placed similarly to other suite elements (delays, selectors, and so on).

The following figure illustrates a synchronization point:

The virtual testers pause at the synchronization point until TestManager releases them.



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

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

When setting synchronization points, you must specify how virtual testers are released from the synchronization point:

- **Together** – Releases all virtual testers at once.

Specify a *restart time* to delay the virtual testers. For example, if you set the Restart time to 4 seconds, after the virtual testers all reach the synchronization point (or the timeout occurs), they wait 4 seconds, and then they are all released.

The default restart time is 0, which means that when the last virtual tester reaches the synchronization point, all virtual testers are released immediately.

- **Staggered** – Releases the virtual testers one by one.

The amount of time that each virtual tester waits to be released is chosen at random and is uniformly distributed within the range of the specified *minimum time* and *maximum time*. For example, if the minimum time is 1 second and the maximum time is 4 seconds, after the virtual testers reach the synchronization point (or the timeout occurs) each virtual tester waits between 1 and 4 seconds before being released. All virtual testers are distributed randomly between 1 and 4 seconds.

The *timeout* period for a synchronization point specifies the total time that TestManager waits for virtual testers to reach the synchronization point. If all the virtual testers associated with a synchronization point do not reach the

synchronization point when the timeout period ends, TestManager releases any virtual testers waiting there. The timeout period begins when the first virtual tester arrives at the synchronization point.

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

The default timeout is 0, which means that there is no timeout. Setting a timeout is useful because one virtual tester might encounter a problem and might never reach the synchronization point. When you set a timeout, you do not hold up other virtual testers because of a problem with one virtual tester.

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

### **Release Times and Timeouts for Synchronization Points in Test Scripts**

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

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

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

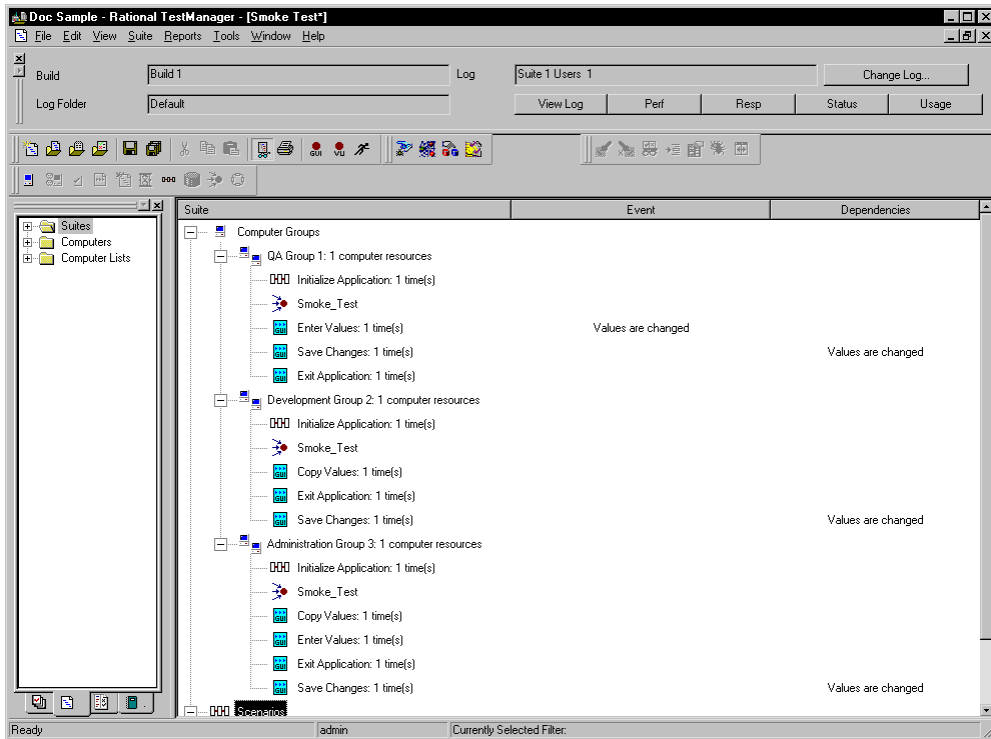
## **Using Events and Dependencies to Coordinate Execution**

---

An *event* is a mechanism that coordinates the way items are run in a suite. For example, you cannot test whether an application will save changes made to certain values unless those values have actually changes. You set a *dependency* on the test scripts that save changes, which blocks virtual testers until the *event* (the changes actually being made) occurs.

You can have multiple events in a suite. While only one item in a suite can *set* an event, many items can *depend* on the event.

The following suite shows virtual testers waiting until the first virtual tester changes values:



The second column in the suite lists the events, and the third column lists the dependencies.

To add a test script that sets an event, or to add a test script that depends on an event:

- Click **Suite > Insert > Test Script**.

**Note:** The previous example shows how to add a test script that sets an event and another test script that depends upon an event. However, scenarios and delays can also set events.



## Executing Suites

---

After you have created and saved your suite, and before you actually run it, you can:

- Check the suite for errors.
- Check the status of Agent computers.
- Control the runtime information of the suite.
- Control how the suite terminates.
- Run the suite.

Finally, while the suite is running, you can monitor the progress of the suite while it is running.

For information on all these topics, see Chapter 5, *Executing Tests*.



This chapter explains how to use the Comparators to compare and view data captured when you use verification points in a Rational Robot test script or in Rational QualityArchitect. This chapter includes the following topics:

- About the Four Comparators
- Starting a Comparator
- Using the Object Properties Comparator
- Using the Text Comparator
- Using the Grid Comparator
- Using the Image Comparator

**Note:** For detailed procedures, see the TestManager Help.

## About the Four Comparators

---

The Comparators are used to view and compare data captured when you use verification points in a Rational Robot test script.

When you record a test script that includes a verification point, Robot creates a *Baseline data file* that contains the data you captured.

When you play back a test script, Robot compares the properties in the Baseline data file with the properties in the application-under-test. If the comparison fails, Robot saves the data that caused the failure to an *Actual data file*. The results of the verification point appear in a test log.

The Comparators are:

- Object Properties Comparator – Use the Object Properties Comparator to view and compare the properties captured when you use the Object Properties verification point.
- Text Comparator – Use the Text Comparator to view and compare alphanumeric data captured when you use the Alphanumeric verification point.

- **Grid Comparator** – Use the Grid Comparator to view and compare data captured when you use the following verification points: Object Data, Menu, or Clipboard. Rational Quality Architect uses the grid comparator to display verification point information.
- **Image Comparator** – Use the Image Comparator to view and edit bitmap images captured when you use the following verification points: Region Image or Window Image. You can also view Unexpected Active Windows.

## Starting a Comparator

---

To start a Comparator from TestManager:

- 1 Click **File > Open Test Log**.
- 2 Expand the Build folder that contains the log, and then double-click the log.  
For the Test Log window of TestManager to open a Comparator, the log must contain a verification point for that particular Comparator.
- 3 Click the **Details** tab at the bottom of the Test Log window.
- 4 In the **Event Type** column, click the plus sign (+) to expand a test script and view all verification points.
- 5 Right-click a verification point and click **View Verification Point**.

The Comparator for that particular verification point opens and that verification point appears.

If the verification point failed, the Comparator opens with both the Baseline and Actual files displayed.

To start a Comparator from Robot, see the *Using Rational Robot* manual.

## Using the Object Properties Comparator

---

Use the Object Properties Comparator to view and compare the properties captured when you use the Object Properties verification point in a Rational Robot test script.

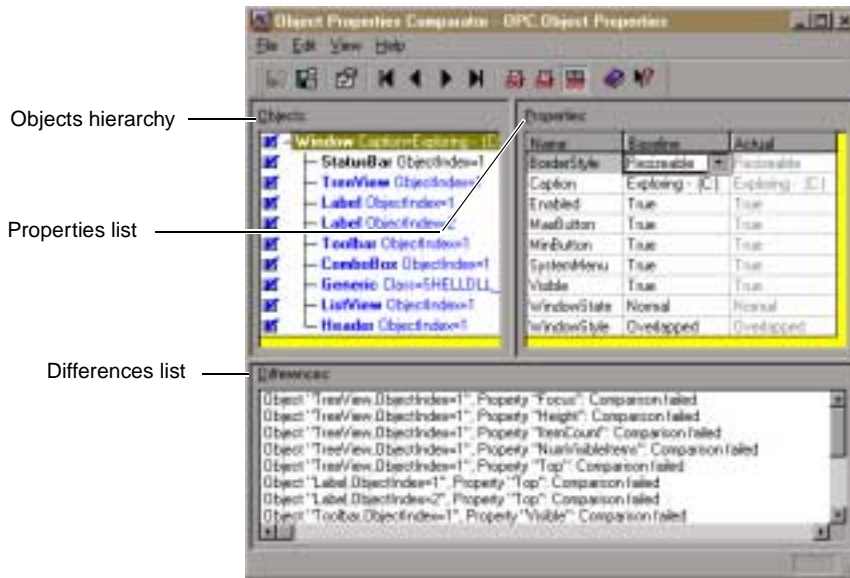
You can use the Object Properties Comparator to:

- Review, compare, and analyze the differences between the Baseline data file and the Actual data file.
- View or edit the Baseline data file for an Object Properties verification point.

To start the Object Properties Comparator from the test log window, see *Starting a Comparator* on page 196.

## The Main Window

The main window of the Object Properties Comparator contains the Objects hierarchy, the Properties list, and the Differences list.



The *Objects hierarchy* contains the list of all objects that Robot records in the Object Properties verification point. The *Properties list* contains the list of properties of those objects. When you select an object on the left, its properties appear on the right. You can control the display of both the Objects and Properties sections of the window by using the **View** commands.

The *Differences list* shows the objects that have differences between the Baseline and the Actual files. If you click an object in the list, that object is highlighted in the Objects hierarchy and Properties list. If you are viewing a file with no failures, this section does not appear. To show or hide this section, click **View > Show Difference List**.

## The Objects Hierarchy and the Properties List

When the Object Properties Comparator is opened, the Objects hierarchy and Properties list appear as follows:

- The Objects hierarchy appears in the left pane of the window. It displays the list of all the objects recorded by Robot using the Object Properties verification point and saved in the Baseline file.
- The Properties list appears in the right pane of the window. It displays the list of properties of the selected object, and the properties' values in the Baseline file and the Actual file (if there are differences).

If the verification point passed, the Comparator displays the Objects hierarchy and the Properties list with only the Baseline column.

If the verification point failed, the Comparator displays the Objects hierarchy and the Properties list with both the Baseline and Actual columns, so you can compare them.

**Note:** If the verification point contains just one object, the Objects hierarchy does not appear. To display it, click **View > Objects** or **View > Objects and Properties**.

### Changing the Window Focus

To change the focus between the Objects hierarchy and the Properties list, do one of the following:

- Click the mouse in the section.
- Press TAB.
- Press ALT+O to set the focus to the Objects hierarchy.
- Press ALT+P to set the focus to the Properties list.

### Working Within the Objects Hierarchy

To display the Objects hierarchy:

- Click **View > Objects** or **View > Objects and Properties**.

The object list is hierarchical. You can expand or collapse the view of objects by selecting a top-level object and using the **View > Expand** and **View > Collapse** commands.

When you select an object, the properties for that object are displayed in the Properties list.

Each object is listed by its object type and is bold. After the object name there may be information such as the object class or index, which can be used to identify the object. If the object is red, it has properties with different values in the Baseline and the Actual files. If the object is blue, it exists in the Baseline file but not in the Actual file.

You can do any of the following to work within the Objects hierarchy. The Objects hierarchy must have window focus.

- Press HOME, END, PAGEUP, PAGEDOWN, UP ARROW, and DOWN ARROW to move between objects.
- Click the check box that precedes each object to select or deselect it for testing. All objects preceded by a check mark are tested.
- Select an object preceded by a check mark to display its properties in the Properties list.
- Select an object and press INSERT to display a dialog box for adding and removing properties from the Properties list for that object.
- Double-click a parent object to expand or collapse its children.
- Press plus (+) to expand the highlighted object one level, or press minus (-) to collapse the highlighted object. Press asterisk (\*) to expand all objects.
- Right-click an object in the hierarchy to display the Objects shortcut menu.
- Double-click an object that is labeled **Unknown** to define the object. For information about defining unknown objects during recording, see the *Using Rational Robot* manual.

## Working Within the Properties List

To display the Properties list:

- Click **View > Properties** or **View > Objects and Properties**.

The Name column shows the name of the property. The Baseline and Actual columns display the values for the properties. Values in the Baseline column represent the properties from the original recording of the Object Properties verification point. Values in the Actual column represent the state of the properties in the latest played back version. By default, if there are differences between the Baseline and Actual, both columns are displayed.

Use the **View** commands to control which columns appear in the Properties list.

If a property is red, it has different values in the Baseline and the Actual files. If a property is blue, it exists in the Baseline file but not in the Actual file. If a value cell is blank, the property has an empty value.

You can do any of the following to work within the Properties list. The Properties list must have window focus.

- Type the first letter of a property's name to move to that property or to the first property beginning with that letter.
- Press HOME, END, PAGEUP, PAGEDOWN, UP ARROW, and DOWN ARROW to highlight a property.
- Press INSERT to display a dialog box for adding and removing properties from the Properties list.
- Select a property and press DELETE to remove it from the list.
- Double-click the value cell of a property to edit the value.
- Position the pointer on the vertical border between column title cells. Drag the pointer to the right or left to change the column widths.
- Point to a property and click the right mouse button to display the Properties shortcut menu.

## Loading the Current Baseline

To load the Current Baseline file:

- Click **File > Load Current Baseline**.

If the Current Baseline is already being displayed, this command will be disabled. In order to edit a Baseline, you must be viewing the Current Baseline. Editing can include creating a mask, cutting, copying, pasting, duplicating, moving, or deleting masks, or using the Auto Mask feature.

The Current Baseline is the latest saved baseline file and is used as the expected result for verification point comparisons. It is this Current Baseline that you see in the Comparator when the Comparator is opened through Robot. However, when the Comparator is opened through the Test Log window of TestManager, which is the more common method, the Comparator may display the historical Baseline and Actual pair. Since only the Current Baseline may be edited, if you have the historical Baseline or any other logged Baseline showing, you will not be able to use any of the editing commands—they will be disabled. You can manually force the Current Baseline to be loaded by using this command.

## Locating and Comparing Differences

The Object Properties Comparator begins its comparison with the first object in the Objects hierarchy and its properties in the Properties list.



Objects that contain differences between the Baseline and Actual lists are red. Objects that are in the Baseline list but not in the Actual list are blue.

To locate the first difference between the Baseline data and the Actual data:

- Click **View > First Difference**.

When the difference is located, the failure is highlighted. The Differences list indicates the failure number and provides information about the failure.

To navigate between differences, use the **View** commands.

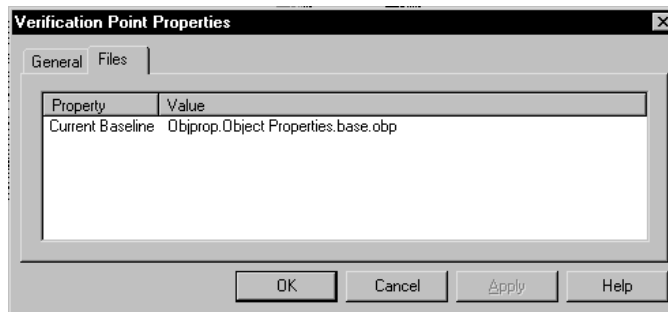
You can also select a description in the Differences list to highlight that failure in the Properties list.

## Viewing Verification Point Properties

To view verification point properties:

- Click **File > Verification Point Properties**.

The Verification Point Properties dialog box shows the verification point type, the name of the Baseline file, and the name of the Actual file.



## Adding and Removing Properties

When you first create an Object Properties verification point, you can specify the properties to test by adding and removing them from the Properties list. You can also add and remove properties from the list when you view the data file in the Object Properties Comparator. This lets you refine a test even after it has been created and played back.

For example, if the Properties list for a verification point contains a Height property that you decide you do not want to test, you can remove the property in the Comparator. You can also apply the properties in the list to all objects of the same type for this verification point, and define a list of default properties for each type of object.

To add a property to the Properties list:

- Click **Edit > Edit Property List**.

Removing a property removes it from the Properties list but does not from the verification point's Baseline file. Removing a property means that it will no longer be tested in future playbacks. Once removed, properties can be added back later.

To remove properties from the Properties list:

- Click **Edit > Remove Property**.

If you remove a property, you can add it back to the Properties list at a later time by using the **Edit > Edit Property List** command.

## Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the Baseline file to keep it up-to-date with the developing application.

When editing the Baseline file, you can:

- Edit a value in the Properties list.
- Cut, copy, and paste a value.
- Copy values from the Actual to the Baseline file.
- Change a verification method.
- Change an identification method.
- Replace the Baseline file.

**Note:** You cannot edit the Actual data file.

For step-by-step instructions on these tasks, search for the task in the Object Properties Comparator Help.

## Saving the Baseline File

To save changes made to the Baseline file:

- Click **File > Save Baseline**.

This command is enabled only if you have made changes to the Baseline file.

## Using the Text Comparator

---

Use the Text Comparator to view and compare alphanumeric data captured when you use the Alphanumeric verification point in a Rational Robot test script.

You can use the Text Comparator to:

- Review, compare, and analyze the differences between the Baseline data file and the Actual data file.
- View or edit the Baseline data file for an Alphanumeric verification point.

To start the Text Comparator, see *Starting a Comparator* on page 196.

### The Main Window

The main window of the Text Comparator contains the Text window.



### The Text Window

The Text window has two panes: Baseline and Actual. The Baseline pane shows the data file that serves as a Baseline file for a comparison. The Actual pane shows data from the current playback. You can control the display of the panes by using the **View** commands.

The Text window uses a typical text editor format. In general, you use the same rules and methods of typing, selecting, and deleting that you would use in a standard text editor (such as Notepad).

The Baseline pane has a white background and the Actual pane has a gray background. Data that failed the comparison between the Baseline file and the Actual file appears in reverse color when you use one of the locating commands to highlight it.

In the Text window, you can:

- Scroll the Text window
- Change the widths of the text panes
- Use word wrap

For step-by-step instructions, search for each task in the Text Comparator Help.

## Locating and Comparing Differences

To locate the first difference between the Baseline data and the Actual data:

- Click **View > First Difference**.

To navigate between differences, use the **View** commands.

The comparison starts in the upper left corner of the pane. The Comparator then scans for differences by going across each row of text in order, as it would in a text editor.

When a difference is found using the **View** commands, the difference between the Baseline file and the Actual file appears in reverse color.

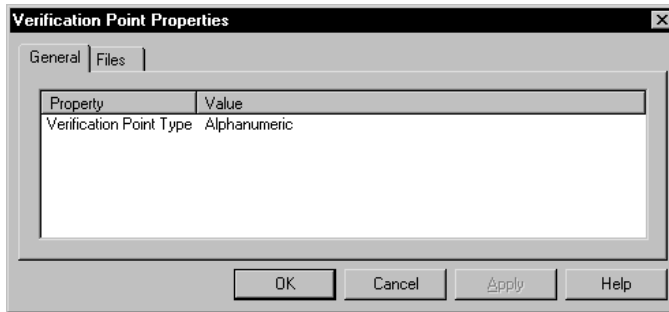
The Alphanumeric verification point stores the specified verification method as part of the test script command. For data files created by the Alphanumeric verification point, the Comparator assumes a case-sensitive comparison, regardless of how it was recorded. For numeric data, the Comparator assumes Numeric Equivalence as the verification method.

## Viewing Verification Point Properties

To view verification point properties:

- Click **File > Verification Point Properties**.

The Verification Point Properties dialog box shows the verification point type, the name of the Baseline file, and the name of the Actual file.



## Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the Baseline file to keep it up-to-date with the developing application.

When editing the Baseline file, you can:

- Edit the data.
- Cut, copy, and paste data.
- Copy data from the Actual to the Baseline file.
- Replace the Baseline file.

**Note:** You cannot edit the Actual data file.

For step-by-step instructions on these tasks, search for each task in the Text Comparator Help.

## Saving the Baseline File

To save changes made to the Baseline file:

- Click **File > Save Baseline**.

This command is enabled only if you have made changes to the Baseline file.

## Using the Grid Comparator

---

Use the Grid Comparator to view and compare data captured when you use the following verification points in a Rational Robot test script:

- Object Data
- Menu
- Clipboard

**Note:** Rational Quality Architect also uses the grid comparator to display verification point information.

You can use the Grid Comparator to:

- Review, compare, and analyze the differences between the Baseline data file and the Actual data file.
- View or edit the Baseline data file for a verification point.

To start the Grid Comparator, see *Starting a Comparator* on page 196.

## The Main Window

The main window of the Grid Comparator contains the Grid window and the Differences list. The Grid window contains the grids of data recorded in an Object Data, Menu, or Clipboard verification point. The Differences list displays descriptions of any items that failed during playback.



## The Grid Window

The Grid window has two panes: Baseline and Actual. The Baseline pane shows the data file that serves as a Baseline file for a comparison. The Actual pane shows data from the current playback. You can control the display of the Baseline and Actual files by using the **View** commands.






The grids in the panes show data in row and column format. Cells with a green background contain data that passed the comparison between the Baseline file and the Actual file. Cells with a red background failed the comparison.

You can set display options to control the Grid window. For more information, see *Setting Display Options* on page 207.

## Differences List

The Differences list displays the Actual items that failed during playback. It shows the reasons why a verification point failed, and it displays icons to graphically illustrate the failure type. If you click an item in the list, that item is highlighted in the grid. If you are viewing a file with no differences, this section does not appear.

The following icons may appear in the Differences list:

Icon	Meaning
	No differences found
	Comparison failed
	Item not found
	Different sizes
	Key not found

To work in the Differences list:

- Use the vertical scroll bar to scroll through the list of descriptions.
- Select a description in the Differences list to highlight the failure in the Baseline and Actual files.

## Setting Display Options

You can set the following display options in the Grid Comparator:

- Change the column widths.
- Transpose the grid data.
- Synchronize the scroll bars.

- Synchronize the cursors.

For step-by-step instructions on these tasks, search for each task in the Grid Comparator Help.

## Locating and Comparing Differences

To locate the first difference between the Baseline data and the Actual data:

- Click **View > First Difference**.

To navigate between differences, use the **View** commands.

You can also select a description in the Differences list to highlight that failure in the Baseline and Actual panes.

In the grid panes, the comparison starts with the first data cell in the grid (the cell in the upper-left corner). The Comparator then scans for differences by going down the first column. At the end of the column the comparison goes to the top of the second column, and so on.

When a difference is located, the Comparator highlights the area of difference using reverse color and highlights the description in the Differences list. You can also select a description in the Differences list to highlight that failure in the Baseline and Actual files.

Verification points that have entire rows or columns selected compare the data in each cell as well as the number of cells in the row or column. If the number of cells is different, the Comparator highlights the row or column and italicizes the header number or text. It also displays a red line around the header cell.

If the data displayed in the grid is larger than the window, you can use the scroll bars to view other areas of the data, or you can resize the window.

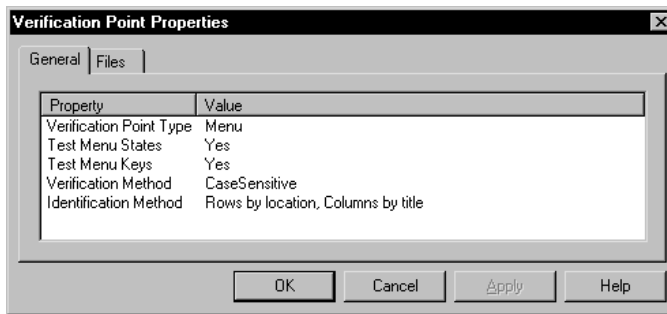
**Note:** If a difference is highlighted in the Baseline file and the description in the Differences list is *Item cannot be found*, it means that there is no difference to highlight in the Actual file, since the item is missing there.

## Viewing Verification Point Properties

To view verification point properties:



- Click **File > Verification Point Properties**.



The Verification Point Properties dialog box shows the verification point type, the name of the baseline file, the name of the actual file, the verification method, the test menu states, test menu keys, and the identification method.

## Using Keys to Compare Data Files

You can select the Key/Value identification method when you create Object Data or Clipboard verification points in Robot.

For verification points that have the Rows by Key/Value identification method, you can use the Grid Comparator to add or change keys in the Baseline file. As in a relational database, keys can be used to uniquely identify a row for comparison.

You can add or change keys to determine what the important comparisons are in a verification point and to possibly change a failed verification point into one that passes.

If the value of the data in a key column changes, Robot will not be able to locate the record, and the verification point will fail. You may then want to change the keys in the Comparator to gain more insight into why the verification point failed.

If you have not specified keys that ensure uniqueness, the test can fail because Robot may compare the selected record to a record that contains similar values but is not the record that you want to test. You can experiment by changing the keys in the Comparator to improve the predictability of the verification point.

If the database schema changes, you can change the keys in the Comparator to identify new and unique columns.

To use keys to compare data files:

- 1 Click the name of a column in the Baseline file.
- 2 Click the right mouse button, or press CTRL+K to add or remove a key.

The data in the Baseline and Actual files should be automatically compared again. At this point you can evaluate the new key placement.

If a key column in the Baseline file has different data from the Actual file, the Differences list displays `Row not found: Row x` and includes the value from the Baseline key column.

If there are no key columns and the row data in the Baseline and Actual files does not match exactly, the Differences list displays `Row not found: Row where x` and includes each column name and value from the Baseline file.

## Editing the Baseline File

When there are intentional changes to the application-under-test, you may need to modify the Baseline file to keep it up-to-date with the developing application.

When editing the Baseline file, you can:

- Edit the data.
- Edit a menu item.
- Cut, copy, and paste data.
- Copy data from the Actual to the Baseline file.
- Save the Baseline file.

**Note:** You cannot edit the Actual data file.

For step-by-step instructions on these tasks, search for each task in the Grid Comparator Help.

## Saving the Baseline File

To save changes made to the Baseline file:

- Click **File > Save Baseline**.

This command is enabled only if you have made changes to the Baseline file.

## Using the Image Comparator

---

Use the Image Comparator to open and view bitmap images captured when you use the following verification points in a Rational Robot test script:

- Region Image
- Window Image

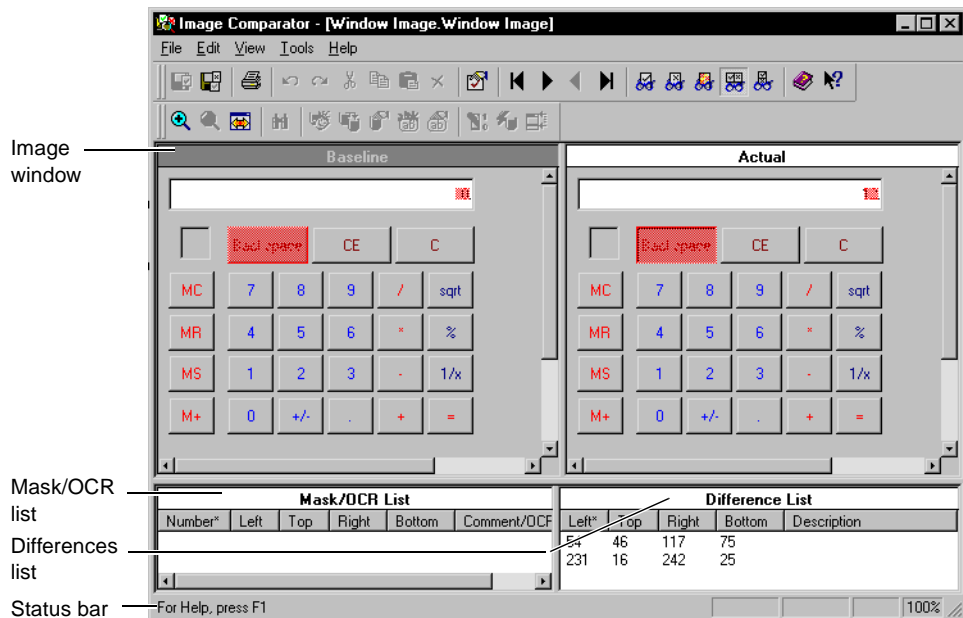
You can use the Image Comparator to:

- Review and analyze the differences between the Baseline image file and the Actual image file.
- Edit the Region Image or Window Image verification points by creating masks on the image.
- Create OCR regions to read the text within a region.
- View images of unexpected active windows that cause a failure during a test script's playback.

To start the Image Comparator, see *Starting a Comparator* on page 196.

## The Main Window

The main window of the Image Comparator contains the Image window, the Mask/OCR List, the Differences List, and the status bar.



## The Image Window

The Image window has two panes: Baseline and Actual. The Baseline pane shows the image file that serves as an expected file for a comparison. The Actual pane shows the image from the current playback. You can control the display of both panes by using the **View** commands.

The parts of the image that passed the comparison between the Baseline file and the Actual file appear exactly as they were recorded. The parts of the image that failed the comparison (that is, the differences) are shown as red regions.

You can move the image within a pane and zoom the image. For information, see *Moving and Zooming An Image* on page 215.

## Differences List

The Differences List displays a list of the items that failed during playback. The Left, Right, Top, and Bottom columns represent the measurement of the sides of the difference area, in numbers of pixels. The number in the Left column is the number of pixels from the left margin to the left edge of the difference region. The number in the Right column is the number of pixels from the left margin to the right edge of the difference region. In the same manner, the Top and Bottom columns define the number of pixels to the top and bottom edges of the difference region, from the top margin.

To work in the Differences List:

- Use the vertical scroll bar to scroll through the list of descriptions.
- Select a description in the Differences list to highlight the failure in the Baseline and Actual files.
- Double-click an item in the list to cause the image to be positioned so that the region is centered in the view. It will flash briefly and then become selected.
- The Difference list is sortable by column. The currently sorted column is indicated with an asterisk. To sort by a different column, click the column header. The list is sorted in ascending order of the selected column.

## Mask/OCR List

Masks are used to hide the underlying masked area from comparison when test scripts are played back. Any areas of the image that contain a mask will not be compared when you play back a test script containing an Image verification point.

Robot uses OCR regions to read the text within a designated region and to compare it in subsequent playbacks of the test script.

The Mask/OCR List in the lower left pane of the main window lists any masks and OCR regions being used in the verification point. When you select a mask or OCR region in the list, it is highlighted in the Baseline and Actual files. This list works in the same way that the Differences List works, as described in the previous section. This section is empty if you do not have any masks or OCR regions defined for the verification point.

The Left, Right, Top, and Bottom columns represent the measurement of the sides of the mask or OCR region in number of pixels. This measurement works in the same way as it works in the Difference List. The Comment column for masks contains optional comments, which you can add by selecting a mask and clicking **Edit > Mask Properties**. The OCR Text column for OCR regions contains the text in the region that will be tested.

## The Status Bar

The status bar at the bottom of the main window provides useful information as you work with the Comparator. To show or hide the status bar, choose **View > Status Bar**.

The message area in the left part of the status bar displays menu command descriptions and operational messages, such as progress updates while the Comparator is scanning the image for differences.

On the right side, there are four small panes for specific information:

**ReadOnly** – Indicates a read-only state. This happens if the current Baseline is not displayed since the current Baseline is the only file that you can edit.

**Load CBL** – Indicates that the current Baseline is not being displayed. If you want to make edits, click **File > Load Current Baseline** to display the current Baseline.

**BLINK** – Indicates that the Blink feature is turned on.

**<zoom percentage>** – Indicates the zoom percentage of the window. If you have the original or normal view, the zoom percentage is 100%. If you have zoomed to some percentage of the normal view, that percentage is shown. If you have fit the image to the window, *FITTED* appears.

## Locating and Comparing Differences

To display differences in the Baseline and Actual images:

- Click **View > Show Differences**.

To locate the first difference:

- Click **View > First Difference**.

When a difference is located, the Comparator flashes it briefly, centers the difference in the panes, and then selects it in both panes.

To navigate between differences, use the **View** commands.

You can also select a difference in the Differences List to highlight that failure in the Baseline and Actual images.

## Changing How Differences are Determined

Each difference region represents a logical set of differing pixels—a cluster of differing pixels close together. Depending on your preference setting, the Comparator determines whether this region is close enough to the last one to be classified as either the same or a different difference region. Every time the Comparator defines a new region around a differing pixel, it determines whether the region is close enough to any other previously defined region. If so, the Comparator combines the two rectangular regions. Otherwise, the region becomes a new difference region.

To change how differences are determined:

- 1 Click **Tools > Options**.

Use this setting to specify how close is close enough when a new differing pixel has been found.

- 2 Change the setting under **Difference Regions**. Move the sliding bar to choose whether you want more or fewer difference regions to be created.

When you move the bar, the picture next to the slide is a representation of that choice.

## Changing the Color of Masks, OCR Regions, or Differences

To change the color of masks, OCR regions, or differences in the Image window:

- 1 Click **Tools > Options**.
- 2 Change the setting under **Colors**.

**Masks** – Select the highlight color for masks in the image. The masks are displayed as a block of this color in the Baseline and Actual files. The default color is a light green. Click **Change** to select a different color.

**Differences** – Select the highlight color for differences in the image. The difference regions are displayed as a block of this color in the Baseline and Actual files. The default color is a light red. Click **Change** to select a different color.

**OCR regions** – Select the highlight color for OCR regions in the image. The regions are displayed as a block of this color in the Baseline and Actual files. The default color is a light blue. Click **Change** to select a different color.

## Moving and Zooming An Image

There are several ways to move the image within the Baseline and Actual panes:

- Use the horizontal and vertical scroll bars. Scrolling is synchronized if you are viewing both files.
- Use the moving hand pointer. If you hold down the left mouse button anywhere in the image that is not a mask or a difference region, the mouse pointer turns into a hand. You can then use it to move the image around in the window.

**Note:** You can use the zooming commands to move around the image. You can zoom in, zoom out, zoom by percentage, fit the image exactly to the window, or return to the normal image size.

- To zoom in on the image, click **View > Zoom > Zoom In**.

This zooms in on the image by a factor of 2. If you have a mask, OCR region, or difference region selected when you use the Zoom command, the zooming is centered on that region. If you do not have a region selected, the zoom is centered on the entire image. You can use the command repeatedly to keep zooming into the image.

- To zoom out from the image, click **View > Zoom > Zoom Out**.
- To zoom the normal display of the image by a percentage, click **View > Zoom > Zoom Special** and the percentage.
- To restore the image to its original size, click **View > Zoom > Normal Size**.
- To fit the image to the full size of the pane, click **View > Zoom > Fit To Window**.

Zoom factors always retain the image's aspect ratio to ensure that text and images appear without distortion. **Fit To Window** represents the largest zoom factor that can display the entire image in the window while maintaining the image's aspect ratio.

## Viewing Image Properties

To view the properties of an image:

- Click **File > Properties**.

The Image Properties dialog box shows information about the image, including its scale, color, size, and the creation date of the file.

## Working with Masks

You can create masks in the Image Comparator with the **Edit > New Mask** command. Masks are used to hide the underlying masked area from comparison when test scripts are played back. Any areas of the image that contain a mask are not compared when you play back a test script that contains an Image verification point.

Use masks to ensure that certain regions are not tested. For instance, if your application has a date field, you might want to mask it so that it will not produce a failure every time the test script is played back. You can also apply masks to hide differences that you determine were caused by intentional changes to the application, so that they do not cause failures in future tests.

Since you can only edit the Baseline file, you cannot perform the following procedures in the Actual file. However, when you select a mask in the Baseline file, the mask is also selected in the Actual file. You cannot modify the mask in the Actual file—it is shown there for convenience only.

You can do the following with masks:

- Display masks.
- Create masks.
- Move and resize masks.
- Cut, copy, and paste masks.
- Duplicate masks.
- Delete masks.
- Automatically mask differences.

For step-by-step instructions, search for each task in the Image Comparator Help.

## Working with OCR Regions

Robot uses Optical Character Recognition (OCR) regions to read the text within a designated region and compare it in subsequent playbacks of the test script.

You can use OCR regions to verify proper operation of an application that dynamically paints text in window areas or where the Actual text is difficult to obtain. OCR regions are also valuable in situations where a text string's font or weight may change unexpectedly but go undetected using traditional verification methods. To achieve the correct verification, you can define OCR regions on existing or newly-captured Image verification points.

You can do the following with OCR regions:



- Create an OCR region.
- Move and resize OCR regions.
- Cut, copy, and paste OCR regions.
- Duplicate OCR regions.
- Delete OCR regions.

For step-by-step instructions, search for each task in the Image Comparator Help.

## Saving the Baseline File

To save changes made to the Baseline file:

- Click **File > Save Baseline**.

This command is enabled only if you have made changes to the Baseline file.

## Viewing Unexpected Active Window

Robot is designed to respond to unexpected active windows (UAW) during test script playbacks. An unexpected active window is any unscripted window appearing during test script playback that interrupts the playback sequence and prevents the expected window from being made active. An example of a UAW is an error message generated by the application-under-test, or an e-mail notification message window.

You can view the unexpected window in the Image Comparator only if you have set the option in Robot. In the GUI Playback dialog box in Robot, click the **Unexpected Active Window** tab. Make sure that the **Detect unexpected active windows** and the **Capture screen image** options are both selected. (For more information about setting an unexpected active window option, see *Using the Rational Robot* manual.)

To open a UAW to view in the Comparator:

- 1 Start TestManager and open a log file containing a UAW.
- 2 Do one of the following in the **Event Type** column:
  - Double-click an unexpected active window event.
  - Select an unexpected active window event and click **UAW**.

The Image Comparator opens and that UAW appears.



# **Part 3: Performance Testing with Rational TestManager**



This chapter introduces performance testing using Rational TestManager. It includes the following topics:

- About performance testing
- Performance testing basics
- Rational TestManager and performance testing
- Planning performance tests
- Implementing performance tests
- Examples of performance tests
- Analyzing performance results

## About Performance Testing

---

Application developers, system testers, and system integrators use TestManager to test multi-user client/server systems. TestManager can be used to add maximum workload conditions to the network and server in a client/server environment, thereby reducing the total testing time. With TestManager, you can coordinate multiple computers as well as emulate multiple virtual testers from a single computer.

Performance tests that are run from TestManager play back test scripts with instructions on performing actions on the tested server. This test script can come from a number of sources:

- You can record a test script in Rational Robot.

When you use Rational Robot to record a script, Robot records activities in a session, and then automatically creates a test script that represents the user's interactions with the server, as well as all queries and responses.

For more information about recording a test script in Rational Robot, see the *Using Rational Robot* manual.

- You can write a test script using any scripting language.

When you supply a test script in this way, you must write an adapter so that TestManager recognizes the test script type. For more information, see the *Rational Test Extensibility Reference* manual.

- You can create a manual script that lists a virtual tester's tasks.

Before you begin planning and developing performance tests, you should be familiar with these TestManager concepts:

- Virtual testers
- Test scripts
- Test cases
- Test plans
- Scripting languages
- Local and Agent computers
- Suites

For more information about these topics, see Chapter 1, *Introducing Rational TestManager*.

## Performance Testing Basics

---

A *performance test* helps you determine whether a multi-client system is performing to defined standards under varying loads and configurations.

Performance testing is a class of tests implemented and executed to characterize and evaluate the performance-related characteristics of the tested server such as:

- Timing profiles
- Execution flow
- Response times
- Operational reliability and limits

Different types of performance tests, each focused on a different test objective, are implemented throughout the software development life cycle (SDLC).

Early in the development lifecycle—in the architecture iterations—performance tests focus on identifying and eliminating architecture-related performance bottlenecks. In the construction iterations, performance tests tune the software and environment

(optimizing response time and resources), and verify that the applications and system acceptably handle high load and stress conditions, such as a large number of transactions, clients, and/or volumes of data. Different types of performance tests are suited to each iteration.

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

In addition, one or more functional testers can be included in a performance test to observe what happens when many virtual testers are running against the same server. Because functional testers measure client response times, they better represent what a real user experiences when there is significant client processing or screen-painting time associated with the user activity that you are measuring. Also, a functional tester is a good cross-check for performance tester responses.

## Types of Tests

The term *performance testing* includes the following types of tests:

- Benchmark tests – Compares the performance of a new or unknown server to a known reference standard, such as existing software or measurements.
- Configuration tests/performance profiling – Verifies the acceptability of the server's performance behavior using varying configurations while the operational conditions remain constant.
- Load tests – Verifies the acceptability of the server's performance behavior using varying workloads while the operational conditions remain constant.
- Stress tests – Verifies the acceptability of the server's performance behavior when abnormal or extreme conditions are encountered, such as diminished resources or an extremely high number of users.
- Contention tests – Verifies that the server can handle multiple user demands on the same resource (that is, data records or memory).

Performance generally is evaluated in steps. For more information about this multi-level approach, see *Analyzing Performance Results* on page 237:

## Benchmark Tests

Benchmark tests can provide a baseline of information on how a server is performing under given conditions. An initial benchmark measurement can provide reference point from which other performance details are evaluated.

Benchmark tests can be as simple as determining what percentage of virtual testers complete an operation in a given amount of time, or as diverse as helping you to figure out why the remaining percentage of virtual testers did not complete the operation in that same amount of time.

If a benchmark of performance is established for a given application, then you can measure configuration, load, stress, and contention test results against this baseline to evaluate relative performance.

## **Configuration Tests/Performance Profiling**

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

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

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

While you typically perform configuration testing with functional testers only, you can also load your client/server system with performance testers to test the effects of changing configurations on workload.

## **Load Tests**

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

Load tests can involve performance testers only (for measuring server response times), or performance testers and functional testers (for measuring client response times).



## Stress Tests

Stress testing is the process of running your client application under extreme conditions to see if it or the server breaks under the strain. Examples of stress testing include:

- Continuously running a client application for many hours.
- Performing a large number of transactions.
- Having hundreds of virtual testers perform the same operation or a specific combination of operations simultaneously.

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

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

You can test the following types of stress conditions:

Stress condition	Type of test
A large number of users performing the same activity simultaneously.	performance
A few users continuously repeating the same action on the client application hundreds or thousands of times or over long periods of time.	functional
A few users continuously repeating the same client requests to a server hundreds or thousands of times or over long periods of time.	performance
Combinations of the preceding three conditions.	performance and functional

## Contention Tests

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

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

If you want to run a contention test under heavy virtual tester conditions, you can use performance testers to add to the load.

For more information on functional testing, see Chapter 6, *Planning Functional Tests*.

## Local and Agent Computers

You coordinate the activities of all your test scripts from a single Windows NT computer where TestManager is running. This is known as the *Local computer*. From the Local computer, you create, run, and monitor suites.

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

- *Adding workload to the server* – If you are running a test with a large number of virtual testers, you can use Agent computers to add load to the server.
- *Running test scripts on more than one computer* – If you are running a functional test, you can save time by running the test scripts on the next available Agent computer instead of having the Local computer run all the test scripts. For this situation, the test scripts must be modular and independent.
- *Testing hardware configurations* – If you are testing different hardware configurations, you can run test scripts on different Agent computers that are set up with these hardware configurations.

## Suites

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

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

# Rational TestManager and Performance Testing

---

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

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

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

## Why Use TestManager for Performance Testing?

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

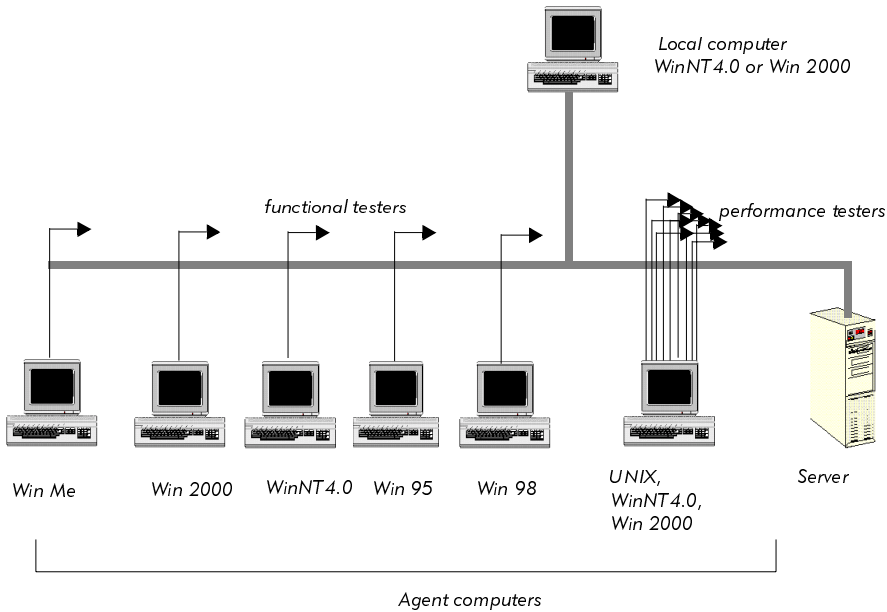
Problem	TestManager Solution
Does the server perform correctly under load?	Conduct stability and stress testing of the network and servers under maximum workload conditions.
Does the system meet scalability requirements?	Determine the number of virtual testers a server can support before the system is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

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

## The TestManager Environment

TestManager enables you to run a suite in a distributed environment. This environment consists of a single Local computer (on which you coordinate test execution and play back test scripts), and zero or more Agent computers (on which you play back test scripts).

The following figure illustrates a typical TestManager configuration:



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

## Planning Performance Tests

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

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

- How many virtual testers can the server support under normal conditions?
- Are there any situations where server performance degrades suddenly under normal conditions?
- How does the system perform when you exceed the normal conditions? In a worst-case scenario, does the system degrade gracefully or does it break down completely?
- How does the system perform under varying hardware configurations?

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

## Testing Response Times

TestManager lets you measure various indicators of performance. Whereas distributed functional testing measures correctness in terms of straightforward pass/fail responses, performance testing also measures time—for example:

- How long did it take for the action to complete?
- How quickly was the server able to respond under heavy load conditions?

You can measure the client response time or server response time, or both.

## Setting Pass and Fail Criteria for Performance Tests

Because performance can be subjective, it is essential that you identify the features to be tested and the criteria that will determine whether performance passes or fails. The pass or fail criteria often involves a range of acceptable response times. For example, you could define the following as an acceptable response time:

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

## Identifying Performance Testing Requirements

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

- Server computers: database servers, Web servers, other server systems
- Client computers: Windows 2000, NT, 98, 95, or Me computers; network computers; or Macintosh or UNIX workstations
- Databases that will be accessed
- Applications that will be running

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

- The size of the test databases and other test files to accurately represent the real workload
- The distribution of data across the server to prevent I/O bottlenecks
- If you are testing a database, the settings of key database parameters

## Designing a Realistic Workload

If you are testing performance, it is essential that your model accurately mirror the workload at your site. Therefore, you must determine the types of transactions that occur at your site.

For example, do users query the database and update it occasionally, or do they update it frequently? If they update the database frequently, are the updates complex and lengthy, or are they short?

When designing the workload, consider these issues:

- **The workload interval** – The period of time the workload model represents.

For example, the workload interval could be a peak hour, an average day, or an end-of-the-month billing cycle.

- **Test variables** – The factors you will change during the performance test.

For example, you could vary the number of virtual testers to understand how response time degrades as the workload increases.

It is best to change only one variable at a time. Then, if performance changes with the next test, you know that the change was caused by that one variable.

You set test variables when you set up a suite. For more information, see Chapter 4, *Implementing Tests*.

- **Virtual tester classifications** – You categorize the virtual testers into groups based on the types of activities they perform.

For each group, you identify the number of virtual testers or the percentage of overall virtual testers. For example, you could group 20% of the virtual testers into Accounting, 30% of the virtual testers into Data Entry, and 50% of the virtual testers into Sales.

You set up user groups within a suite. However, you should keep these user groups in mind as you plan the test scripts to be associated with the test. The test scripts should accurately reflect the actions of realistic user groups. For information about setting up user groups, see *Inserting User Groups into a Suite* on page 246.

- **User work profiles** – The set of activities that the virtual testers perform and the frequency with which they perform them. The virtual tester actions should mirror the mix of tasks that the users actually perform as closely as possible.

For example, if the Sales user group access the database 70% more than the other two groups, be sure that the workload reflects this.

- **User characteristics** – Determine how long a virtual tester pauses before executing a transaction, the rates at which the transaction is executed, and the number of times a transaction is executed consecutively. It is important to model the real user characteristics accurately because the values directly affect the overall performance of the system.

For example, a user who thinks for 5 seconds and types 30 words per minute puts a much smaller workload on the system than a user who thinks for 1 second and types 60 words per minute.

Use delays and think times to model the virtual tester characteristics. For more information about delays, see *Inserting a Delay* on page 263. For more information about think times, see the *Using Rational Robot* manual.

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

Taking the time to consider these issues will save you time in the long run. The more clearly defined your testing goals are, the more quickly you can achieve them.

## Implementing Performance Tests

---

Once you have chosen the pass and fail criteria, hardware and software requirements, and workload model, you are ready to create test scripts and set up the tests. Some issues to consider during this phase of the process are:

- **The termination conditions** – If one virtual tester fails, should the test stop or should it keep running?

If you are implementing a large number of virtual testers and a few fail, generally the test can continue. However, if a virtual tester that performs a fundamental task (such as setting up the database) fails, the test should stop.

You set termination conditions in the suite. For more information about setting termination conditions, see *Controlling How a Suite Terminates* on page 99.

- **The stable workload** – Should the test wait until all virtual testers are connected, or should the test begin running immediately?

If you are trying to measure the response time for virtual testers, you probably should wait until all testers are connected before the actual testing begins.

You define a stable workload for reporting purposes in the Performance report. For more information, see *Reporting on a Stable Load* on page 321.



- **The applications that you will test** – It is not cost-effective to test all of your applications. Spending time and resources testing an application that has little effect on overall performance takes away from the time spent testing critical applications. You must consider this balance when planning and designing tests.

In general, you should identify the 20% of the applications that generate 80% of the workload on your system. For example, you might not want to include an application that updates a database at the end of the year only.

- **The database on which you will run the test** – Decide whether you want to run the test on the production database or a test database. Running tests on production systems in current use may yield incorrect results as the effect of the regular user load is not included in the workload model.

## Examples of Performance Tests

---

This section summarizes some typical performance tests. Each test objective is accompanied by a table that lists the key elements to consider when defining such a test. The tables are intended only as a guide; they do not attempt to define all of the possible elements you can include in your performance tests.

### Number of Virtual Testers Supported Under Normal Conditions

Suppose you want to determine the number of virtual testers that a server can support, to ensure that the system can meet your scalability requirements. How many virtual testers can the system support before the response is unacceptable?

For example, you estimate that a database system supports 500 virtual testers. You could plan to run the test with 300, 400, 500, 600, and 700 virtual testers concurrently performing multiple tasks. The following table shows the key elements you might include when designing the test:

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

## Incrementally Increasing Virtual Testers

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

With TestManager, you could model this type of workload by incrementally loading virtual testers. You would start by developing a model of the workload that you want to test. For example, write down the frequency and volume of use of your applications. Then, when setting up your suite:

- 1 Schedule different user groups to start at different times over the life of the suite.
- 2 For each user group, set the number of virtual testers that run the test script and an iteration count (optional) as appropriate for your test.

By layering the start time and iteration count of your virtual testers, you build up load incrementally. You also can add spikes of load at specific times in your suite run.

The following describes a sample test that represents overlapping shifts:

- You start a suite with 100 virtual testers. This group of virtual testers represents the early shift of entry clerks repeating the same group of order entry transactions over and over, so you should set each virtual tester to run many iterations of the transaction; you should set enough iterations to keep this group of virtual testers running the test script until the suite ends. You may have to experiment to determine how many iterations you need.
- Through a suite, set a **Delay**. The **Delay type** might be from the start of the suite, or it might begin at a certain time of day. When the delay is over, 200 new virtual testers begin. This is the next shift of entry clerks which overlaps the first shift.
- During the combined shift, which represents peak load, 300 virtual testers perform transactions repeatedly.

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

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

## How a System Performs Under Stress Conditions

Stress testing can be understood as a relentless attempt to cause a breakdown in the server. Use a stress test when you suspect that there are some weak areas of the server, which may break down completely or diminish responsiveness when an operation is performed a high number of times or over a long period of time.

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

In a simple stress test, you could create a test where virtual testers perform the same operation continuously and repeatedly for hours on end. Your test might involve:

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

The following table summarizes a sample stress test:

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

## How Different System Configurations Affect Performance

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

- You want to test how your system performs with more (or less) memory.
- You want to test how your system performs with a different amount of disk space.
- You want to find the network card with which the system performs best.

The following table summarizes a sample configuration test for 100 virtual testers:

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

## Analyzing Performance Results

---

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

First, you should check that your data is statistically valid. To do this, run a Performance report and a Response vs. Time report on your data.

**Note:** At the end of a suite run, if the log information is appropriate and complete, TestManager runs the Performance and Response vs. Time reports automatically.

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

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

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

The following sections summarize the different levels of detail that you can use to analyze the results of your tests. For more information on performance testing reports, see Chapter 13, *Reporting Performance Testing Results*.

## **Comparing Results of Multiple Runs**

The first level of analysis involves evaluating the graphical summaries of results for individual suite runs and then comparing the results across multiple runs. For example, examine the distribution of response times for individual virtual testers or transactions during a single suite run. Then compare the mean response times across multiple runs with different numbers of virtual testers.

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

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

## **Comparing Specific Requests and Responses**

The second level of analysis involves examining summary statistical and actual data values for specific virtual tester requests and system responses. Summary statistics include standard deviations and percentile distributions for response times, which indicate how the system responses vary by individual virtual testers.

For example, if you are testing a SQL database, you could trace specific SQL requests and corresponding responses to analyze what is happening and the potential causes of performance degradation.

For second-level analysis, you could:

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

## Determining the Cause of Performance Problems

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

### Analyzing Results Statistically

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

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

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

Some of the measurements to consider during third-level analysis are:

- **Minimum** – The lowest response time.
- **Maximum** – The highest response time.

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

For example, System A and System B both have a mean response time of 12 milliseconds (ms). This does not necessarily mean that the system response is the same. Further evaluation of the results reveal that System A has response times of 11, 12, 13, and 12, and System B has response times of 1, 20, 25, and 2. Although the mean time is the same (12 ms), the minimum, maximum, and standard deviation are all quite different.

## Monitoring Computer Resources and Tuning Your System

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

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

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



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

For more information about running the Response vs. Time report, see *Mapping Computer Resource Usage onto Response Time* on page 323.



This chapter describes how to design performance testing suites. It includes the following topics:

- About suites
- Creating a suite
- Inserting user groups into a suite
- Inserting test scripts into a suite
- Inserting other items into a suite
- Using events and dependencies to coordinate execution
- Executing suites

## About Suites

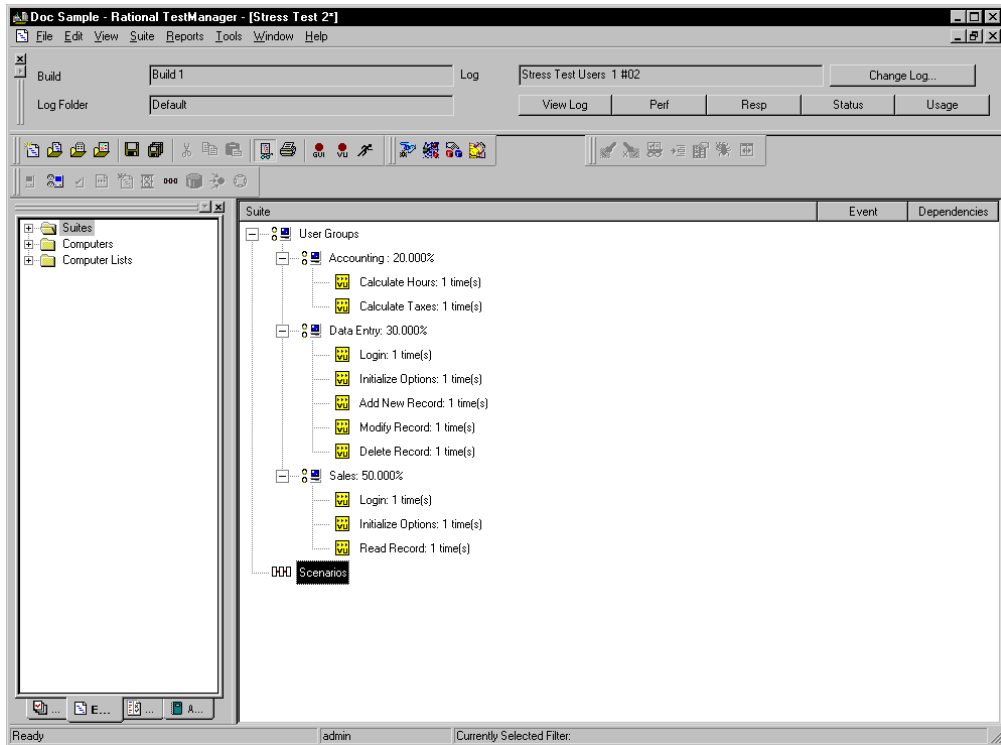
---

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

Through a suite, you can:

- Run test scripts and test cases.
- Group test scripts to emulate the actions of different types of virtual testers.
- Set the order in which test scripts run.
- Synchronize virtual testers.

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



In this suite:

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

The examples in this chapter show VU test scripts. A suite, however, can contain GUI scripts, VU scripts, VB scripts, Java scripts, or other user-defined test script types. For more information on defining other test script types and using them in TestManager, see the *Rational TestManager Extensibility Reference* manual.

## Creating a Suite

---

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

You can create a suite in several different ways. You can create a new suite:

- Using the performance testing suite wizard.
- Using the functional testing suite wizard.
- Based on an existing suite of any type.
- Based on an existing Robot session.
- Using a blank performance testing suite.
- Using a blank functional testing suite.

To create a new suite using any of these methods:

- Click **File > New Suite**.



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

### About Creating a Suite from a Wizard

If you are new to testing, using the suite wizards may be the easiest and fastest way to create a working test. Each wizard guides you through the process of creating a suite.

When you create a suite using the performance testing wizard, TestManager helps you choose the computer on which the test will run and helps you associate scripts that become the basis for the test.

When you create a new suite using the functional testing wizard, TestManager helps you choose test cases and scripts that become the basis for the test.

**Note:** When you create a new suite using the wizards, you must have valid test scripts available for use in the suite. For information on recording test scripts, see the *Using Rational Robot* manual. For information on other test script types, see the *Rational Test Extensibility Reference* manual.

## About Creating a Suite from a Session

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

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

Creating a suite from a session saves you from having to add individual test scripts to the suite. For example, suppose you record the test scripts Connect, Query, and Disconnect in a recording session named DBQuery. To run the test scripts in a suite, you could either add each test script to the suite in the order in which you recorded them, or you could create the suite from the session DBQuery.

Creating a suite from a session maintains the scripts in the order in which you recorded them originally. If you want to maintain this order, you should create the suite from the session. If you want more flexibility of placement of scripts in a suite, you should add the test scripts individually. For information about adding test scripts to suites, see *Inserting Test Scripts into a Suite* on page 249.

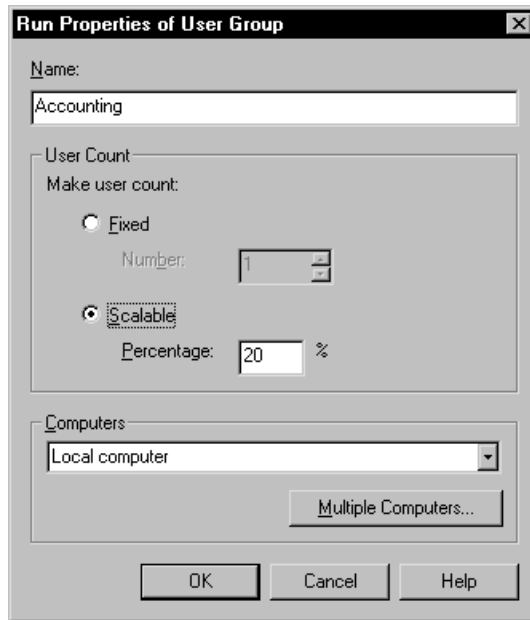
## Inserting User Groups into a Suite

---

A *user group* is the basic building block for all performance testing suites. A user group is a collection of virtual testers that perform the same activity. For example, the suite on page 244 contains three user groups: Accounting, Data Entry, and Sales.

To insert a user group into a suite:

- From an open suite, click **Suite > Insert > User Group**.



**Note:** The name of a user group cannot be identical to the name of a shared variable, a test script, or the following reserved words: MASTER, ALL, ASSIGN, TO, THRU, END, UNION, DELAY, delay, shared, SHARED, SYNC, DLB\_FREQ, DLB\_TIME, PERMUTE, TSIDX, CIDX, TC\_START, TC\_END.

When you add a user group to a suite, you must specify whether the group contains fixed or scalable virtual testers:

- **Fixed** – Specifies a static number of virtual testers. Enter the maximum number of virtual testers that you want to be able to run. For example, if you enter 50 virtual testers, you can run up to 50 virtual testers in the Sales group each time you run a suite.

Typically, you assign a fixed number of virtual testers to user groups that do not add a workload. For example, one virtual tester could run a Warmup test script to open a database for the virtual testers, and another virtual tester could run a Shutdown test script to restore and close the database.

- **Scalable** – Specifies a dynamic number of virtual testers. Type the percentage of the workload that the user group represents. For example, the Accounting group might represent 20 percent of the virtual testers, the Data Entry group might represent 30 percent of the virtual testers, and the Sales group might represent 50

percent of the virtual testers. Each time you run a suite, specify the total number of virtual testers that will run; TestManager distributes the virtual testers among the scalable user groups according to the percentages you specify.

When you define a user group, you must also specify the computer where the user group runs. The default computer is the TestManager Local computer, but you can specify that the user group runs on any defined Agent computer.

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

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

**Note:** Copy any custom-created external C libraries, Java class files, or COM components necessary for the test to the Agent computer.

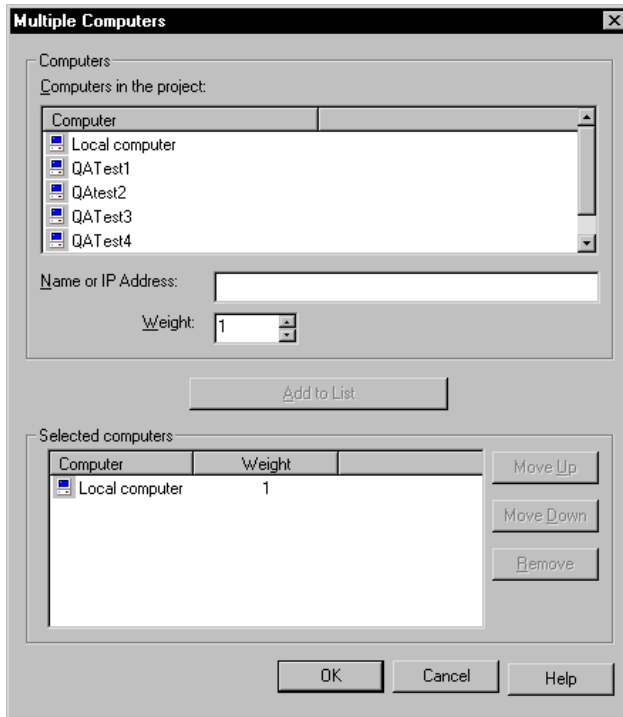
You can also distribute the virtual testers among multiple computers. Typically, you run a user group on multiple computers if you have:

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



To distribute the virtual testers in a user group among multiple computers:

- Click **Suite > Insert > User Group**, and then click **Multiple Computers**.



## Inserting Test Scripts into a Suite

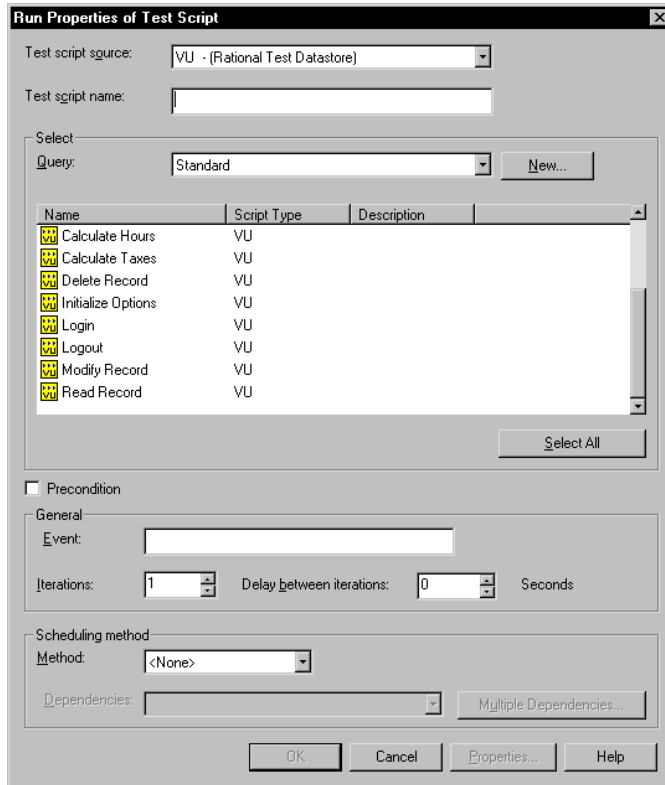
---

After you insert user groups into a suite, you add the test scripts that the user groups run. The suite on page 244 shows the test scripts associated with each user group. The Accounting group runs two test scripts, the Data Entry group runs five test scripts, and the Sales group runs three test scripts.

**Note:** You cannot mix GUI and VU test scripts in a user group. You can, however, mix other test script types.

To insert a test script into a suite:

- From an open suite, select the user group to run the test script, and then click **Suite > Insert > Script**.



## Preconditions

---

When you specify a test script, suite, or test case to be included in a suite, you can specify that the item is a *precondition* for the remainder of that suite sequence. This means that the test script, suite, or test case must complete successfully during the suite run for subordinate items in the suite sequence to run.

To set a precondition on a test script, suite, or test case:

- Right-click the test script, suite, or test case to which to apply the precondition and select **Run Properties**.

**Note:** Preconditions for test scripts, test cases and suites, within suites are different from preconditions and postconditions on test cases as discussed in Chapter 4, *Implementing Tests*.

Since suites can be complex and contain subordinate suites, test cases, and user groups, preconditions apply to their immediate sequence of events. For example, a suite includes two subordinate suites, each of which contain a setup type of script (logging on to a network, for example) and several test cases. If in the first suite a precondition is applied to a set up script and the script fails, TestManager skips all remaining actions (test cases) within that subordinate suite *only*. The suite resumes at the beginning of the next suite (or whatever is next in the larger suite).

Although preconditions are most commonly applied to test scripts, they also can be applied to test cases and suites within a suite. The precondition property applies only to the specific instance of the item in the suite. For example, if a test script is used multiple times within a suite, preconditions must be set for each instance of the test script individually.

Unlike events or dependencies, when a precondition is applied to a test script, suite, or test case, that item *must* pass for subsequent items in that section of the suite to continue.

Preconditions on test scripts, test cases, and suites can be used to ensure that the precondition of a test case is met correctly. For more information on test case preconditions and postconditions, see Chapter 3, *Designing Tests*.

## Inserting Other Items into a Suite

---

A suite requires only user groups and test scripts to run. However, a suite that realistically models the work that actual users perform is likely to be more complex and varied than this simple model. A realistic suite might also contain test cases, subordinate test suites, scenarios, selectors, delays, synchronization points, and transactors to represent a variety of virtual tester actions.

In addition to the items that you can add to a suite in TestManager, you can add certain features to a test script *only* through Rational Robot. These items—timers, blocks, and comments—are discussed in detail in the *Using Rational Robot* manual.

## Inserting a Test Case into a Suite

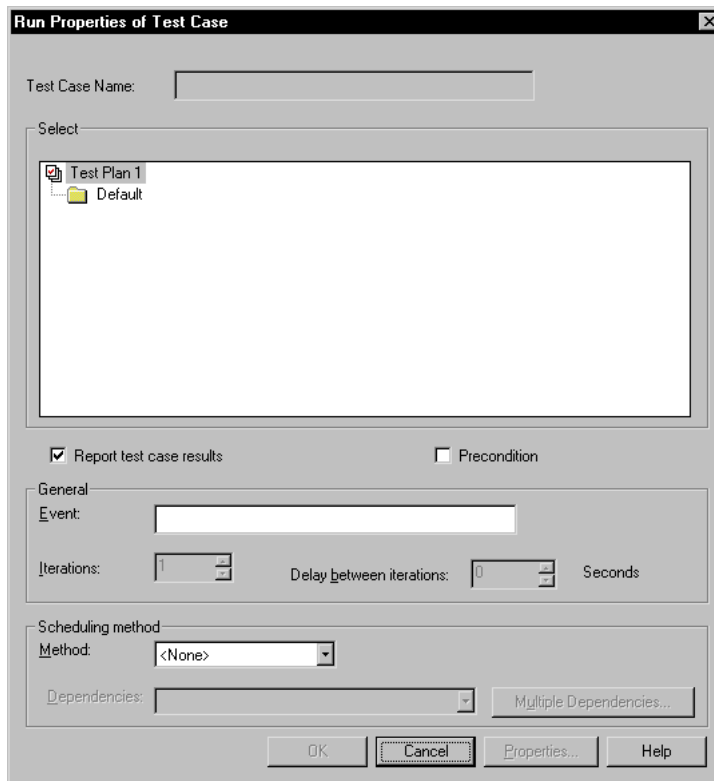
A test case, as discussed in Chapter 1, is a testable and verifiable behavior in system. It can include:

- Test inputs – the defined test requirement, possibly including Rational RequisitePro documents, Rational Rose models, or other kinds of items.
- Execution conditions – where, what, and how the input is tested, such as the operating system on the target computer.
- Expected results – the actual behavior to be verified.

The behavior can be as varied as a simple mouse click or a combination of server response times.

To insert a test case into a suite:

- From an open suite, click **Suite > Insert > Test Case**.



A test case can be considered *configured*, depending on its properties.

- Test cases define a behavior to be verified in the system. Test cases are not system dependent; they can be run on a system with any configuration.
- Configured test cases not only define a behavior to be verified in the system, but also specify the setup of the system on which the behavior will be verified. For the test criteria to be met and verified, the system on which the test case runs must exactly match the defined configuration.

Test cases can be included in suites for a number of reasons. Using a test case as a building block lets you create a test that can be used and applied in a variety of different ways depending on the resources specified at runtime. This can be useful for a set of test cases run on a regular basis. When you include configured test cases in suites, TestManager pairs available systems with matching configurations for you at runtime. Thus different configured test cases may run each time depending on system availability, simulating variations and randomness in system use.

Preconditions can be applied to test cases. For information on preconditions, see *Preconditions* on page 250.

To set a precondition on a test case:

- Right-click the test case, and then select **Run Properties**.

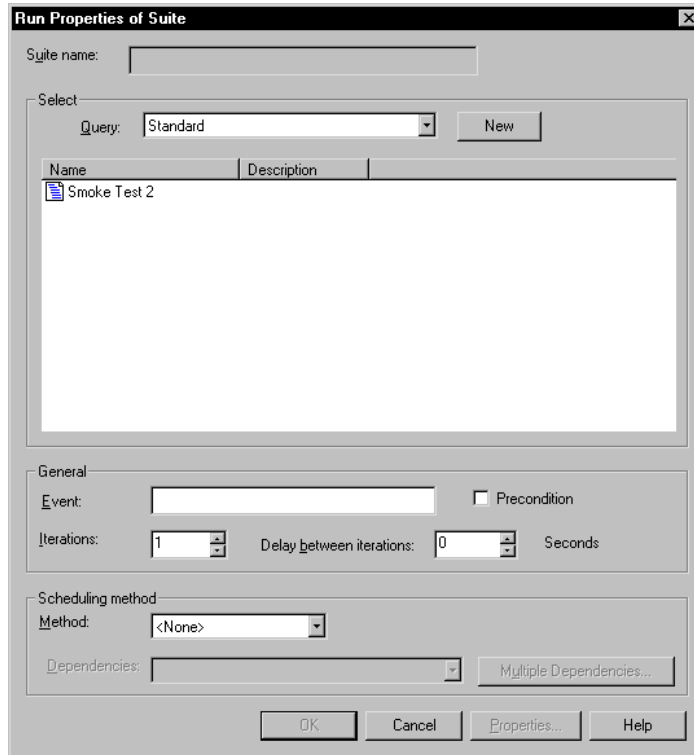
## Inserting a Suite

For maximum flexibility and power, TestManager allows you to insert a complete computer-based suite into another suite. This allows you to use suites as building blocks of tests just as you would any other suite item.

**Note:** You cannot place a user group-based performance suite into another suite. In addition, computer group-based functional suites placed into a suite must have been created with the **Prompt for resources** option selected for the computer group.

To insert a suite into a suite:

- From an open suite, click **Suite > Insert > Suite**.



Using suites as building blocks is particularly helpful when you are creating a large, complex tests, or when you are creating multiple tests that perform several duplicate functions. You can create and check a smaller suite, then insert it to any other suite. You save time by not having to redefine the same test assets in each separate suite. Any change made to a suite is replicated in every instance of that suite, thus making suites easier to maintain.

Preconditions can be applied to suites. For information on preconditions, see *Preconditions* on page 250.

To set a precondition on a suite:

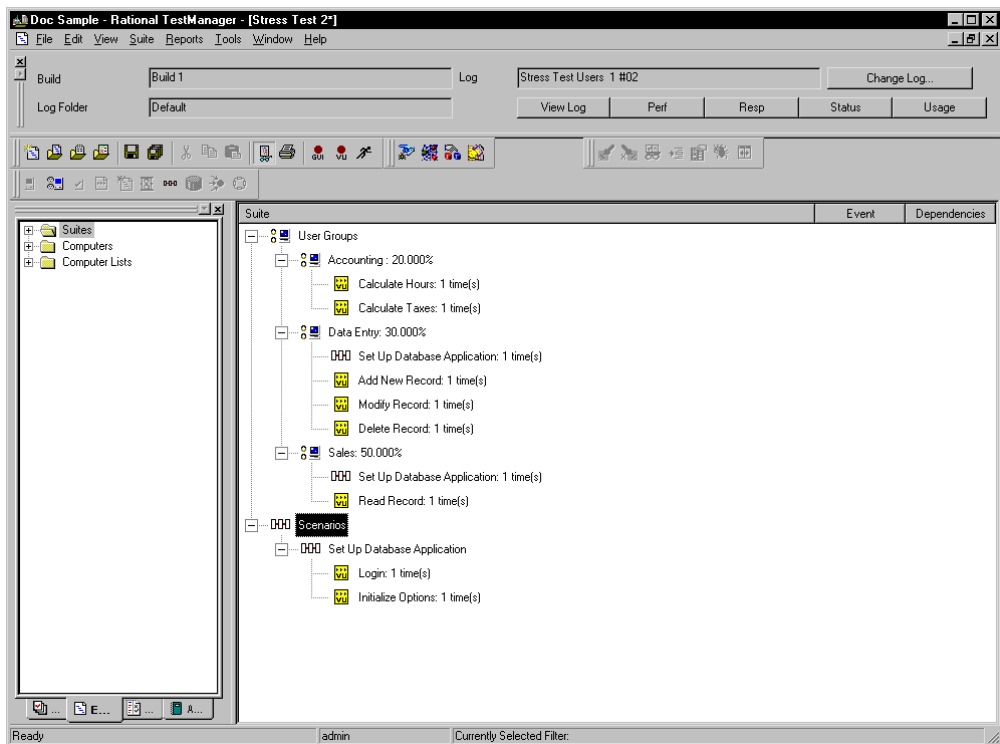
- Right-click the suite, and then select **Run Properties**.

## Inserting a Scenario

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

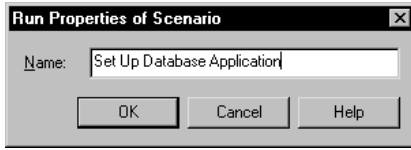
You define a scenario in the **Scenarios** section of the suite by creating a scenario and then inserting items within it. To make a user group execute a scenario, you insert the scenario name in a user group. Otherwise, the scenario is not executed.

In the suite on page 244, both the Data Entry and the Sales user groups run the test scripts Login and Initialize Options. You can simplify this suite by storing both test scripts in a scenario. The following suite shows the test scripts Login and Initialize Options grouped under the Set Up Database Application scenario:



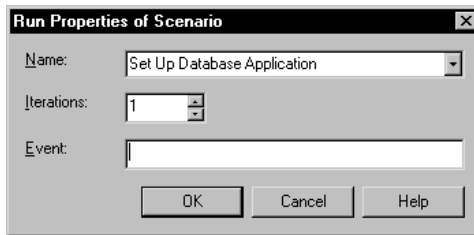
To create a new scenario:

- From the Scenarios section of the suite, click **Suite > Insert > Scenario**.



To insert a scenario into a suite:

- Click where you want to place the scenario, and then click **Suite > Insert > Scenario**.



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

## Suite or Scenario?

The results of inserting a suite or scenario into a suite are similar. But each has advantages and disadvantages.

- Use a suite when you want to reuse a series of events in a variety of suites, and you want to be sure that any change made to that series of events filters to all instances of the suite containing those events, across multiple suites. Suites are reusable among different suites.
- Use a scenario when you want to reuse a series of events within a suite, and you want to be sure that any change made to that scenario filters to all instances of it within a suite. Scenarios are not reusable among different suites.

For example, you could create three suites: one for testing accounting functions, one for testing data entry functions, and one for testing sales functions. Each suite needs to have virtual testers log in to a database to perform tasks. Yet within each suite,



tasks unique to the suite need to be repeated. You could use a suite for log in that could be inserted in each suite, but within each suite, use a scenario for the repeated functions unique to the suite.

## Inserting a Selector

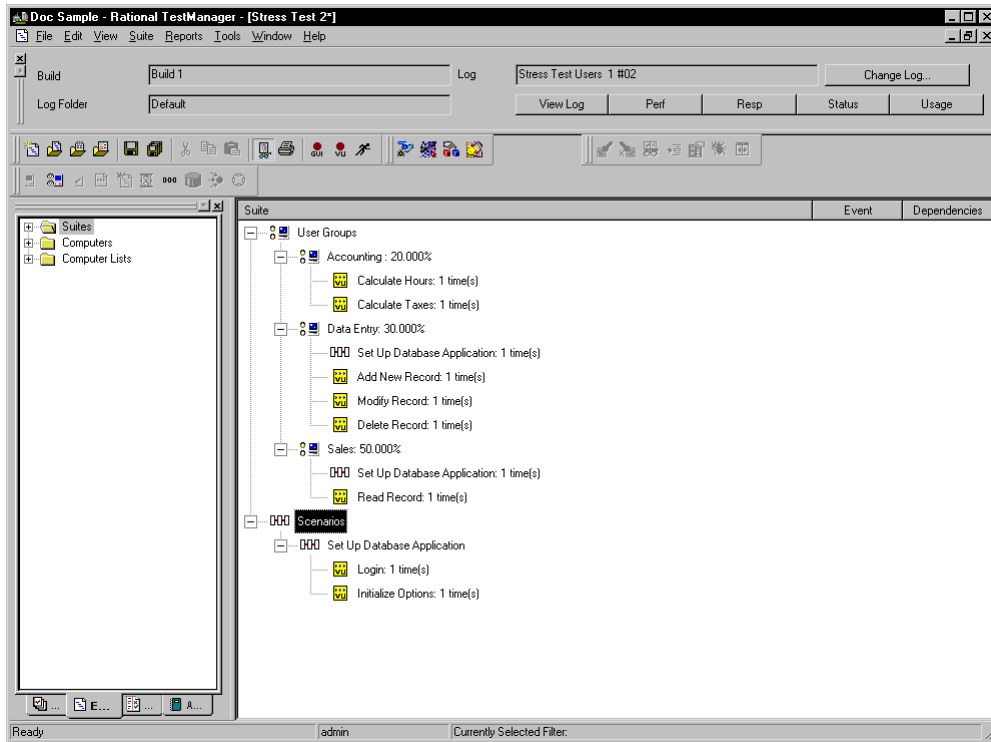
TestManager allows you to set suite items to run in different sequences by setting a *selector*. A selector provides more sophisticated control than running a simple sequence of consecutive items in a suite. A selector tells TestManager which items each virtual tester executes, and in what sequence. For example, you might want to repeatedly select a test script at random from a group of test scripts. A selector helps you to do this.

To insert a selector into a suite:

- Select the user group or a scenario that will contain the selector, and then click **Suite > Insert > Selector**.



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

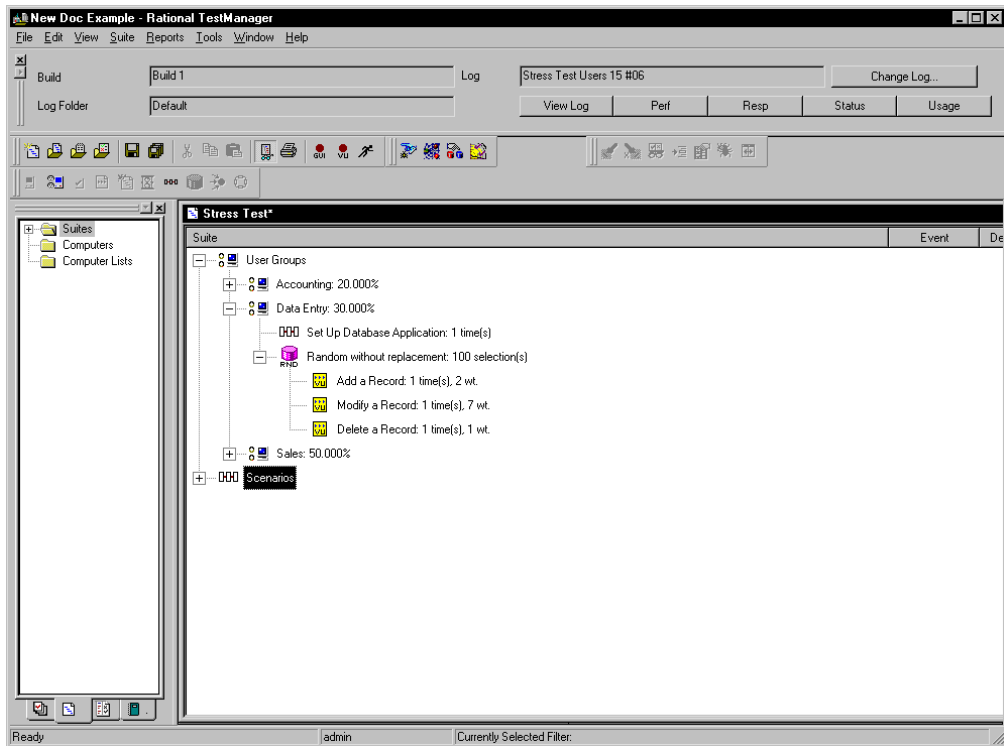


When you run the suite with 50 virtual testers, TestManager assigns 10 virtual testers to Accounting, 15 virtual testers to Data Entry, and 25 virtual testers to Sales (based on the specifications of the scalable user groups). All 50 virtual testers start executing test scripts at the same time.

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

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

To make your user group reflect this behavior, insert a *Random selector* into the Data Entry user group. The following suite shows the Data Entry user group set up to select test scripts randomly without replacement.



When you run the suite with 50 virtual testers, scaled according to user group specifications, each Data Entry virtual tester:

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

## Types of Selectors

TestManager provides the following types of selectors:

- **Sequential** – Runs each test script or scenario in the order in which it appears in the suite. This is the default.
- **Parallel** – Distributes its test scripts or scenarios to an available virtual tester (one virtual tester per computer). Typically, you use this selector in functional testing. The items are parceled out in order, based on which virtual testers are available to run another test script. Once an item runs, it does not run again.

A parallel selector distributes each test script without regard to its iterations.

- **Random with replacement** – The selector runs the items under it in random order, and each time an item is selected, the odds of it being selected again remain the same.

Think, for example, of a bucket that contains 10 red balls and 10 green balls. You have a 50% chance of picking a red ball and a 50% chance of picking a green ball. The first ball selected is red. The ball is then replaced in the bucket with another red ball. Every time you pick a ball, you have a 50% chance of getting a red ball.

Since the ball is replaced after each selection, the bucket always contains 10 red and 10 green balls. It is even possible (but unlikely) that you pick a red ball every time. Similarly, the Random with replacement selector is not guaranteed to run every item in it, particularly if you have set one test script to run more frequently than another. In other words, if your bucket contains 19 red balls and one green ball, the green ball might not be selected at all.

- **Random without replacement** – The selector runs the items under it in random order, but each time an item is selected, the odds change. For example, think of the same bucket that contains 10 red balls and 10 green balls. Again, the first ball selected is red. However, the ball is *not* replaced in the bucket. Therefore, the next time you have a slightly greater chance of picking a green ball. Each time you select a ball, your odds change.

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

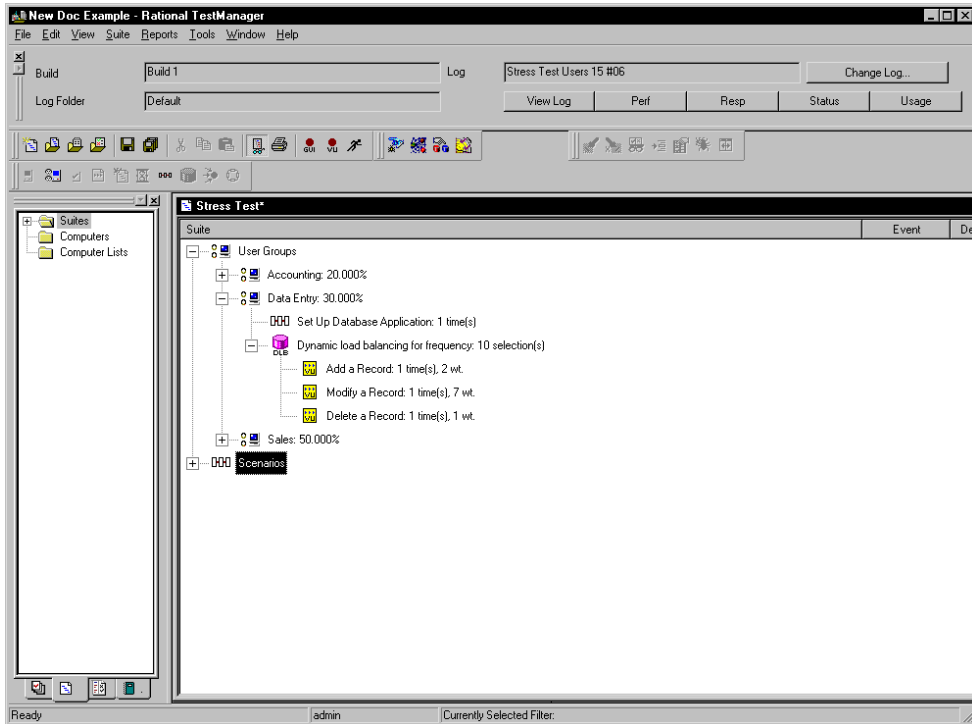
- **Dynamic load balancing** – With dynamic load balancing, items are not selected randomly. Think again of the bucket that contains red and green balls. You have assigned an equal “weight” to each ball. If the first ball that is selected is red, the second ball selected is always green. This is because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

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

If you choose to balance the load dynamically for time, the load is balanced by the runtime of each test script. Because ScriptA takes twice as long to run, it is actually selected only half as often as ScriptB.

If you choose to balance the load dynamically for frequency, both test scripts run an equal number of times. If ScriptA runs 500 times, ScriptB also runs 500 times. The fact that ScriptA takes longer to run is not factored into the balance.

Dynamic load balancing is done across all virtual testers in a user group. For example, the following figure shows the Data Entry user group with 15 virtual testers. Three test scripts, Add New Record, Modify Record, and Delete Record, are contained in a dynamic load balancing selector.



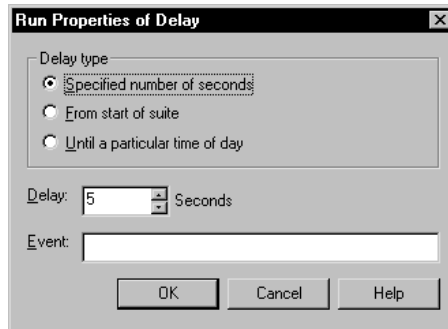
When you run the suite, the first Data Entry virtual tester selects the Modify Record script, because it has the largest weight. But because the workload is balanced across *all* Data Entry virtual testers, after the first virtual tester exits, TestManager recalculates the weights to reflect the fact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

## Inserting a Delay

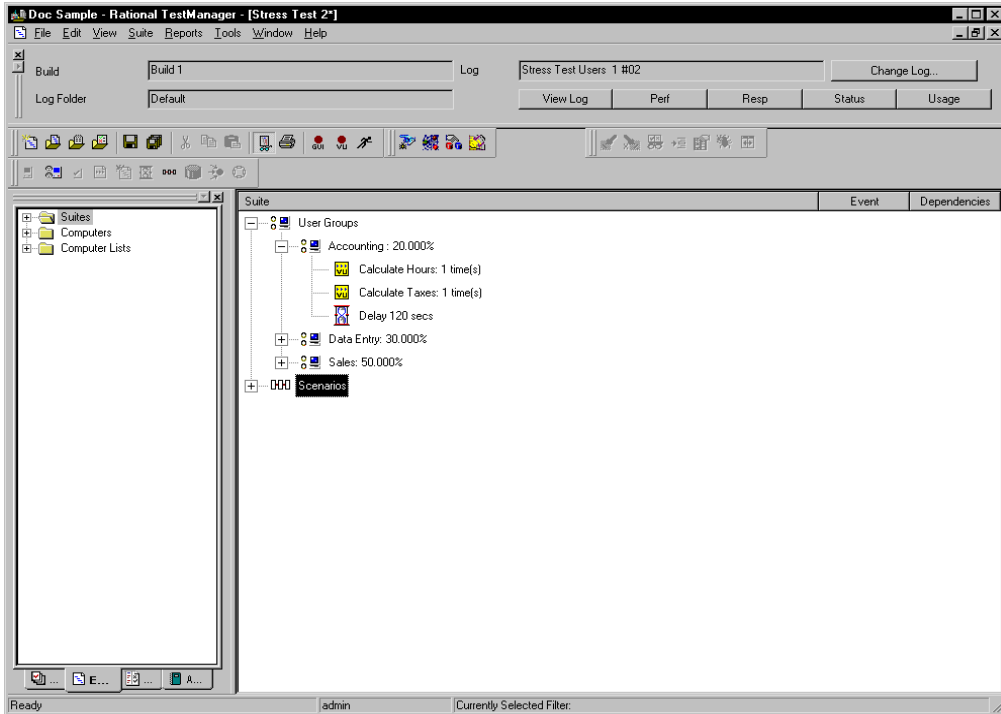
A *delay* tells TestManager how long to pause before it runs the next item in the suite.

To insert a delay into a suite:

- Click the user group, scenario, or selector to which to add a delay, and then click **Suite > Insert > Delay**.



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



You can insert a delay into a suite or a test script. The advantages of inserting a delay into a suite are that the delay is visible in the suite and the delay is easy to change without editing the test script.

The method that you use to insert a delay into a test script depends on the test script type. For VU test scripts, you insert a delay into a test script by editing the test script to include a `delay` routine or by modifying the think time environment variables. (For VB test scripts, the routine is `TSSDelay.Delay`; for Java test scripts, the routine is `TSSDelay.delay`.) Use this method to make the delay before test script execution different each time. For more information about `delay` library routines and the think time environment variables in the VU language, see the *VU Language Reference* manual or the Help. For delay routines other script types, such as VB and Java, see the appropriate Rational Test Script Services API documentation.

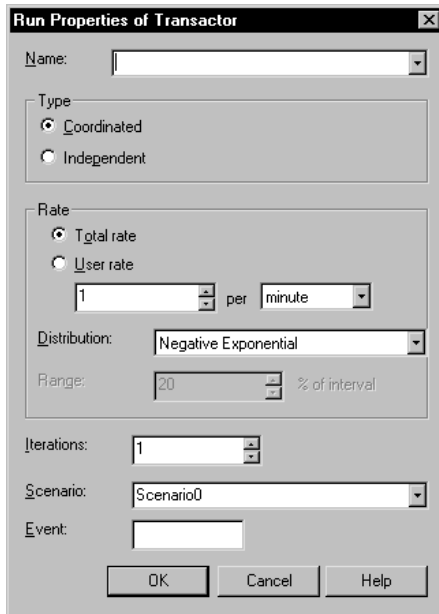


## Inserting a Transactor

A *transactor* tells TestManager the number of tasks that each virtual tester runs in a given time period. For example, you might be testing an Order Entry group that completes 10 forms per hour, or you might be testing a Web server that you want to be able to support 100 hits per minute. Use a transactor to model this time-based behavior.

To insert a transactor into a suite:

- Select the user group or selector to contain the transactor, and then click **Suite > Insert > Transactor**.



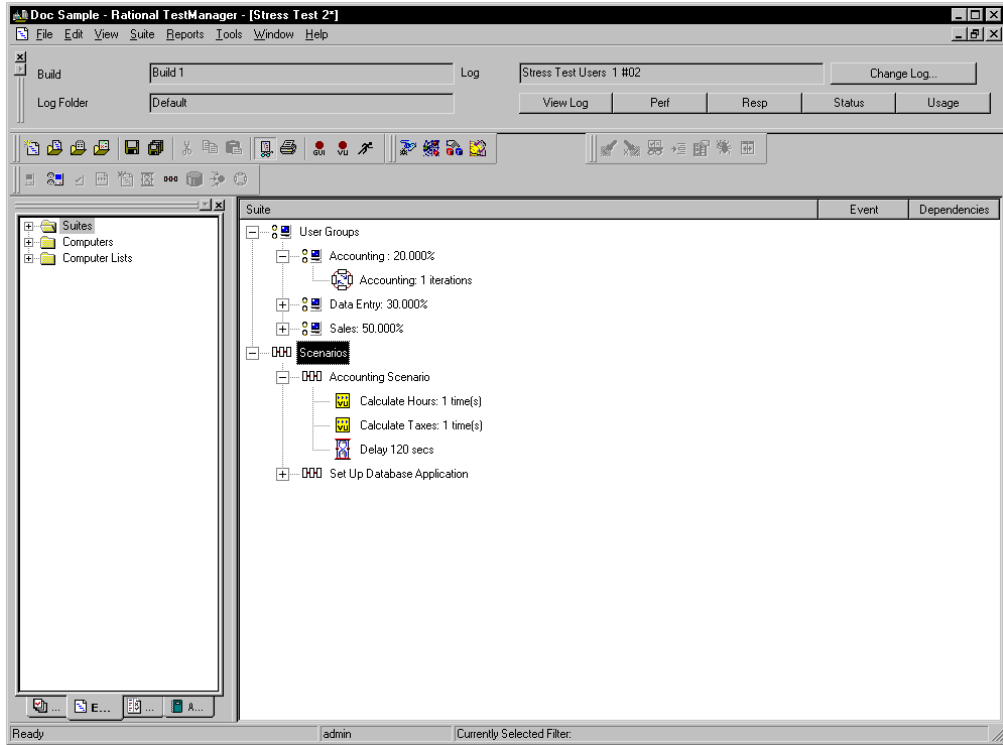
The screenshot shows the 'Run Properties of Transactor' dialog box. It has a title bar with a close button. The dialog contains several sections:

- Name:** A text input field.
- Type:** Two radio buttons: 'Coordinated' (selected) and 'Independent'.
- Rate:** Two radio buttons: 'Total rate' (selected) and 'User rate'. Below 'User rate' is a numeric input field with '1' and a unit dropdown menu with 'minute' selected.
- Distribution:** A dropdown menu with 'Negative Exponential' selected.
- Range:** A numeric input field with '20' and a unit dropdown menu with '% of interval' selected.
- Iterations:** A numeric input field with '1'.
- Scenario:** A dropdown menu with 'Scenario0' selected.
- Event:** A text input field.

At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

In the previous section, you added a delay to the Accounting user group. This delay made the virtual testers pause for two minutes after they calculated the hours and taxes, as shown in the suite on page 264.

However, suppose that the Accounting group instead calculates the hours and the taxes at the specific rate of 10 transactions per hour. You could edit the suite to reflect this by replacing the selector and delay with a transactor. The following suite shows the Accounting user group after you have added a transactor:



This suite is identical to the one on page 264, except that it contains:

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

A transactor can be one of two types:

- A *Coordinated* transactor, which has a built-in synchronization point, lets you specify the total rate that you want to achieve. The virtual testers work together to generate the workload. For example, if you run a suite with 10 virtual testers and then run the same suite with 20 virtual testers, the total transaction rate will stay the same.

Use a coordinated transactor when you are emulating the total transaction rate applied to a server, rather than the rate of specific times a virtual tester runs a task. For example, to emulate the number of hits per minute that a Web server can handle, use a coordinated transactor.

- An *Independent* transactor lets each virtual tester operate independently. It does *not* coordinate the virtual testers under it with a built-in synchronization point. For example, if you run a suite with 10 virtual testers and then run the same suite with 20 virtual testers, the total transaction rate will double—because the number of virtual testers have doubled.

Use an independent transactor if different user groups run the transaction at different times, or if you are emulating individual behavior rather than a group behavior. For example, to emulate an Accounting user group that performs 10 calculations per hour but not all at the same time, use an independent transactor.

Once you have defined the transactor type, you must then specify the transactor rate:

- **Total rate** – For a coordinated transactor, you generally select *Total rate*. This is because whether 100 virtual testers or 50 virtual testers are participating, it has no effect on the rate that TestManager submits transactions.
- **User rate** – For an independent transactor, you must select *User rate*.

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

Next, specify the distribution of the transactor:

- A *Constant* distribution means that each transaction occurs exactly at the rate you specify. For example, if the transaction rate is 4 per minute, a transaction starts at 15 seconds, 30 seconds, 45 seconds, and 60 seconds—exactly four per minute, evenly spaced, with a 15-second interval. Although this distribution is simple conceptually, it does not accurately emulate the randomness of virtual tester behavior.

A *Constant* distribution is useful for emulating an automated process. For example, you might want to emulate an environment where virtual testers are uploading data to a database every half hour.

- A *Uniform* distribution means that over time, the transactions average out to the rate you specify, although the time between each transaction is constant. The time between the start of each transaction is chosen randomly with a uniform distribution within the selected range. Think of this range as a “window” through which the transaction runs.

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

Therefore, the first transaction starts at 12–18 seconds (15 plus or minus 3). The second transaction starts 15 seconds plus or minus 3 seconds after the first transaction starts. If the first transaction starts at 12 seconds, the second transaction starts at 24 to 30 seconds. However, if the first transaction starts at 18 seconds, the second transaction starts at 30 to 36 seconds.

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

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

- A *Negative Exponential* distribution, in contrast, changes the *probability* of when a transaction starts. This distribution most closely emulates the bursts of activity followed by a tapering off of activity that is typical of virtual tester behavior. Using

the same example of 4 transactions per minute, the probability that a transaction starts immediately is high, but decreases over time. TestManager maintains the desired average rate.

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

Transactors can be inserted in a user group or independently in a sequential or random selector. If you are inserting an independent transactor *within* a random selector, you must specify the weight of the selector. For information about selectors, see *Types of Selectors* on page 260.

A transactor can set an event. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 276.

## Inserting a Synchronization Point

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

- All virtual testers associated with the synchronization point arrive at the synchronization point.

When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. When the specified number of virtual testers reaches the synchronization point, TestManager releases the virtual testers and allows them to continue executing the suite.

- A timeout period is reached before all virtual testers arrive at the synchronization point.

When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. Other testers arrive at the synchronization point and wait. However, before all virtual testers arrive at the synchronization point, the timeout period expires and TestManager releases the virtual testers and allows them to continue executing the suite. Virtual testers that did not make it to the synchronization point before the timeout expired do not stop at the synchronization point. They also continue executing the suite.

- You manually release the virtual testers while monitoring the suite.

When one virtual tester encounters a synchronization point, the virtual tester stops and waits for other virtual testers to arrive. Other testers arrive at the synchronization point and wait. However, this time you decide to release virtual testers from the synchronization point and continue executing the suite. All virtual testers may or may not have arrived at the synchronization point. Virtual testers that did not make it to the synchronization point before you released them manually do not stop at the synchronization point. They also continue executing the suite.

You can insert a synchronization point:

- Into a test script – Insert a synchronization point into a test script using Rational Robot in one of the following ways:
  - During recording, using the toolbar button or the Insert menu.
  - During test script editing, by manually typing the synchronization point command into the test script.

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

Use this method if the synchronization point depends upon some logic that you add to the test script during editing.

For information on inserting a synchronization point into a test script during recording, see the *Using Rational Robot* manual.

- Into a suite – Insert a synchronization point into a suite through TestManager.

Insert a synchronization point into a suite in TestManager to pause execution before or between test scripts rather than within a test script. Inserting a synchronization point into a suite offers these advantages:

- You can easily move the location of the synchronization point without having to edit a test script.
- The synchronization point is visible within the suite rather than hidden within a test script.

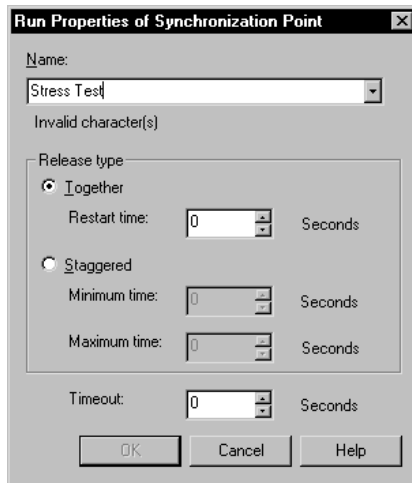
When you insert a synchronization point into a suite in TestManager, you can do more than assign a synchronization point name to a test script. For example:

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

- If the virtual testers are to be released at different times (staggered), you can specify the minimum and maximum times within which all virtual testers are released.
- You can specify a timeout period.

To insert a synchronization point into a suite:

- Click **Suite > Insert > Synchronization Point**.



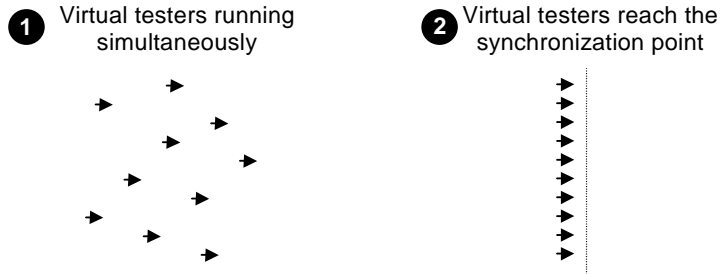
For example, when you run a stress test (an attempt to run your applications under extreme conditions to see if they or the server “break”), your suite might contain virtual testers that perform the certain operations continuously and repeatedly for hours on end. To most effectively run a stress test, you could synchronize the virtual testers so that they perform the operations at the same time to stress the system. You could do this by inserting a synchronization point to coordinate these virtual testers to perform certain functions simultaneously.

### How Synchronization Points Work

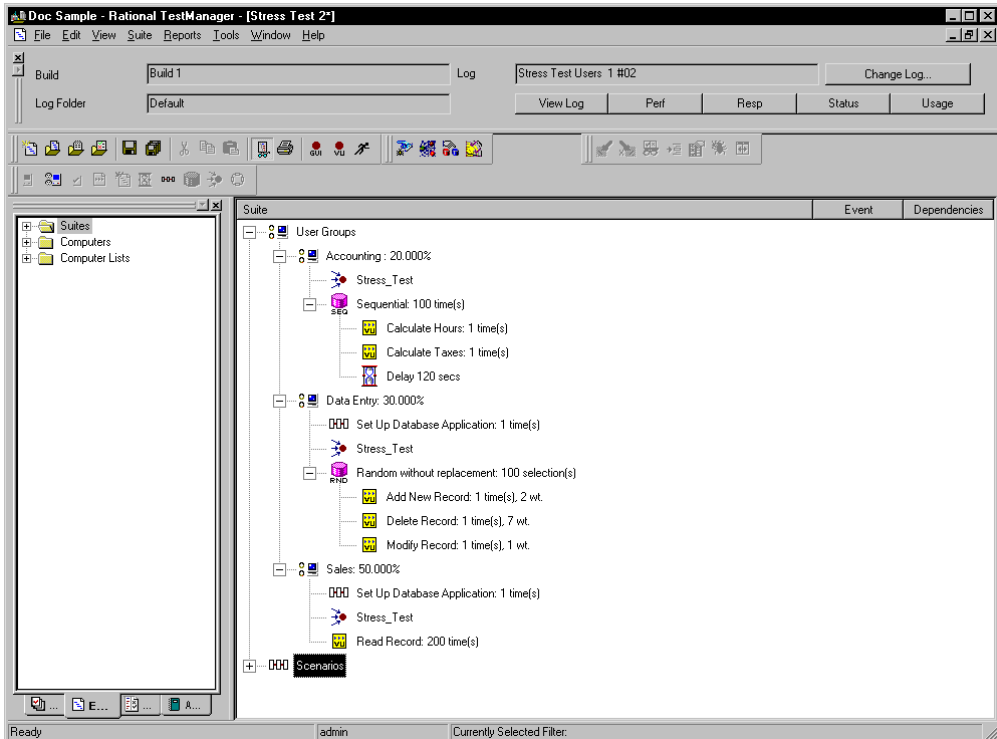
At the start of a test, all virtual testers begin executing their assigned test scripts. They continue to run until they reach the synchronization point. When specified in a test script, a synchronization point is a programmatic command (`sync_point` in a VU test script, `SQASyncPointWait` in a SQABasic test script, `TSSSync.SyncPoint` in a VB test script, or `TSSSync.syncPoint` in a Java test script). When specified in a suite, a synchronization point is placed similarly to other suite elements (delays, transactors, and so on).

The following figure illustrates a synchronization point:

The virtual testers pause at the synchronization point until TestManager releases them.



The following suite shows synchronization points called Stress Test:





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

If you run the test with 10,000 virtual testers, when all the virtual testers reach the Stress Test synchronization point, they are released. In this example:

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

When setting synchronization points, you must specify how virtual testers are released from the synchronization point:

- *Together* – Releases all virtual testers at once.

Specify a *restart time* to delay the virtual testers. For example, if you set the Restart time to 4 seconds, after the virtual testers all reach the synchronization point (or the timeout occurs), they wait 4 seconds, and then they are all released.

The default restart time is 0, which means that when the last virtual tester reaches the synchronization point, all virtual testers are released immediately.

- *Staggered* – Releases the virtual testers one by one.

The amount of time that each virtual tester waits to be released is chosen at random and is uniformly distributed within the range of the specified *minimum time* and *maximum time*. For example, if the minimum time is 1 second and the maximum time is 4 seconds, after the virtual testers reach the synchronization point (or the timeout occurs) each virtual tester waits between 1 and 4 seconds before being released. All virtual testers are distributed randomly between 1 and 4 seconds.

The *timeout* period for a synchronization point specifies the total time that TestManager waits for virtual testers to reach the synchronization point. If all the virtual testers associated with a synchronization point do not reach the synchronization point when the timeout period ends, TestManager releases any virtual testers waiting there. The timeout period begins when the first virtual tester arrives at the synchronization point.

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

The default timeout is 0, which means that there is no timeout. Setting a timeout is useful because one virtual tester might encounter a problem and might never reach the synchronization point. When you set a time you, you do not hold up other virtual testers because of a problem with one virtual tester.

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

### **Why Use Synchronization Points?**

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

Synchronization points inserted into test scripts are used in conjunction with timers to determine the effect of varying virtual tester load on the timed activity. For example, to simulate the effect of virtual tester load on data retrieval:

- 1 While recording the test script (named VU1 in this example) that submits the query and displays the result, perform the following actions:
  - a Insert a synchronization point named `TestQuery` into the test script.
  - b Start a block.

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

- 4 Run the suite a number of times, each time applying a different number of virtual testers to the VU2 test script. However, you need only one virtual tester running the VU1 test script in each test.

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

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

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

### **Release Times and Timeouts for Synchronization Points in Test Scripts**

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

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

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

### **Scope of a Synchronization Point**

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

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

- A Data Entry user group of 75 virtual testers. This user group runs a test script containing the synchronization point `Before Query`.
- An Engineering user group of 10 virtual testers. This user group runs a different test script than the Data Entry groups runs. But this test script also contains a synchronization point named `Before Query`.

- A Customer Service user group of 25 virtual testers. This user group runs a test script that contains no synchronization points. However, the user group does have a synchronization point defined for it. This synchronization point is also named Before Query.

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

## Using Events and Dependencies to Coordinate Execution

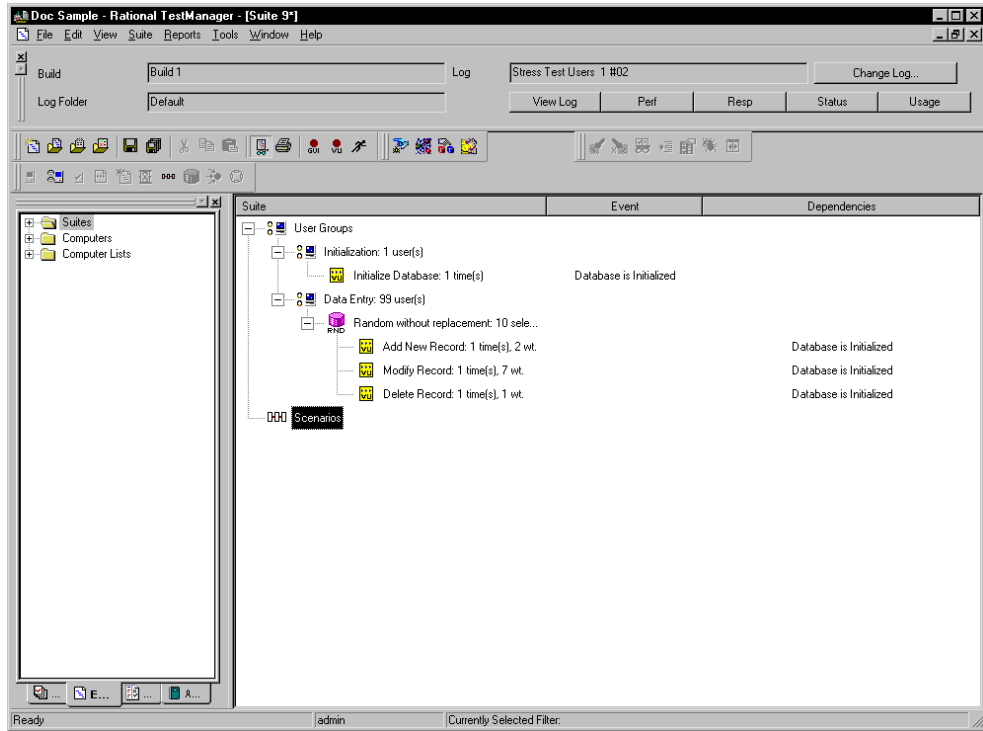
---

An *event* is a mechanism that coordinates the way items are run in a suite. For example, you are running a suite that contains 100 virtual testers that access a database. You want the first virtual tester to initialize the database, and the other 99 virtual testers to wait until the initialization is complete. To do this, you could set a *dependency* on the initialization event, which blocks the 99 virtual testers until the *event* (the first virtual tester initializes the database) occurs.

You can have multiple events in a suite. While only one item in a suite can *set* an event, many items can *depend* on the event.

**Note:** Events and dependencies require only that actions occur—not necessarily that they complete successfully. If parts of your suite require that actions not only occur, but also complete successfully, then use a precondition on a test case, test script, or suite. For more information on preconditions set on items within suites, see *Preconditions* on page 250.

The following suite shows 99 virtual testers waiting until the first virtual tester initializes a database:



The second column in the suite lists the events, and the third column lists the dependencies. In this suite, as soon as the Initialize Database test script completes, it sets the event Database Is Initialized. The Add New Record, Modify Record, and Delete Record test scripts depend on this event and can run only after it is set.

**Note:** In the previous example, the virtual testers in the Data Entry user group ran test scripts randomly. In this case, you must add a dependency to each test script in the selector, because you do not know which test script will run first. However, if the Data Entry user group runs the test scripts sequentially, add a dependency to the first test script only.

To add a test script that sets or depends on an event:

- Click **Suite > Insert > Test Script**.

**Note:** The previous example shows how to add a test script that sets an event and another test script that depends upon an event. However, scenarios, transactors, and delays can also set events, and executables can be dependent on an event.

## Executing Suites

---

After you have created and saved your suite, and before you actually run it, you can:

- Check the suite for errors.
- Check the status of Agent computers.
- Control the runtime information of the suite.
- Control how the suite terminates.
- Run the suite.

Finally, while the suite is running, you can monitor the progress of the suite while it is running.

For information on all these topics, see Chapter 5, *Executing Tests*.

This chapter describes how to create and manage datapools. It includes the following topics:

- What is a datapool
- Planning and creating a datapool
- Data types
- Managing datapools
- Managing user-defined data types
- Generating and retrieving unique datapool rows
- Creating a datapool outside Rational Test
- Creating a column of values outside Rational Test

You should familiarize yourself with the concepts and procedures in this chapter before you begin to work with datapools.

**Note:** This chapter describes datapool access from VU and GUI test scripts played back in a TestManager suite. Additional information on datapools can be found in a number of different Rational documents:

- For datapool procedures, see the Rational TestManager Help.
- For information on using datapools in VB or Java test scripts see the appropriate Rational Test Script Services API documentation.
- For information on datapools in custom test script types, see the *Rational TestManager Extensibility Reference* manual.
- For more information on creating datapools during test script recording, see the *Using Rational Robot* manual and Robot Help.
- For information about datapools and GUI test scripts see the *SQABasic Reference* manual.

## What Is a Datapool?

---

A *datapool* is a test dataset. It supplies data values to the variables in a test script during playback.

Datapools let you automatically supply variable test data to virtual testers under high-volume conditions that potentially involve hundreds of virtual testers performing thousands of transactions.

Typically, you use a datapool so that:

- Each virtual tester that runs the test script can send realistic data (which can include unique data) to the server.
- A single virtual tester that performs the same transaction multiple times can send realistic (usually different) data to the server in each transaction.

If you do not use a datapool during playback, each virtual tester sends the same literal values to the server—the values defined in the test script.

For example, suppose you record a VU test script that sends order number 53328 to a database server. If 100 virtual testers run this test script, order number 53328 is sent to the server 100 times. If you use a datapool, each virtual tester can send a different order number to the server.

## Datapool Tools

You create and manage datapools with either Robot or TestManager, as follows:

Activity	Robot	TestManager
Automatically generate datapool commands in a test script.	•	
Create a datapool and automatically generate datapool values.	•	•
Edit the DATAPOOL_CONFIG section of a VU test script.	•	
Edit datapool column definitions and datapool values.	•	•
Create and edit datapool data types.		•



Activity	Robot	TestManager
Perform datapool management activities such as copying and renaming datapools.		•
Import and export datapools.		•
Import data types.		•

This chapter discusses datapools and explains how to perform activities related to datapools in TestManager. For information on activities associated with datapools performed in Rational Robot, see the *Using Rational Robot* manual.

## Managing Datapool Files

A datapool consists of two files:

- Datapool values are stored in a text file with a .csv extension.
- Datapool column names are stored in a specification(.spc) file. Robot or TestManager is responsible for creating and maintaining this file. You should never edit this file directly.

.csv and .spc files are stored in the TMS\_Datapool directory of your project.

Unless you import a datapool, Robot or TestManager automatically creates and manages the .csv and .spc files based on instructions that you provide through the user interface.

If you import a datapool, you are responsible for creating the .csv file and populating it with data. However, the Rational Test software is still responsible for creating and managing the .spc file for the imported datapool.

For information about importing datapools, see *Importing a Datapool* on page 300 and *Creating a Datapool Outside Rational Test* on page 306.

**Note:** TestManager automatically copies a .csv file to each Agent computer that needs it. If an Agent's .csv file becomes out-of-date, TestManager automatically updates it.

## Datapool Cursor

The datapool **cursor**, or row-pointer, can be shared among all virtual testers that access the datapool, or it can be unique for each virtual tester.

Sharing a datapool cursor among all virtual testers allows for a coordinated test. Because each row in the datapool is unique, each virtual tester can share the same cursor and still send unique records to the database.

In addition, a shared cursor can be persistent across suite runs. For example, suppose that the last datapool row accessed in a suite run is row 100:

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

**Note:** Virtual testers running SQABasic test scripts can share a cursor when playback occurs in a TestManager suite, but not when playback occurs in Robot.

For information about defining the scope of a cursor, see the description of the **Cursor** argument in the *Using Rational Robot* manual.

## Row Access Order

Row access order is the sequence in which datapool rows are accessed at test runtime.

With GUI test scripts, you control the row access order of the datapool cursor through the *sequence* argument of the SQABasic SQADatapoolOpen command.

With VU test scripts, you control row access order through the **Access Order** setting in the Robot Configure Datapool in Script dialog box. (See the *Using Rational Robot* manual.)

For other types of test scripts, refer to the appropriate API documentation.

## Datapool Limits

A datapool can have up to 150 columns if Robot or TestManger automatically generates the data for the datapool, or 32,768 columns if you import the datapool from a database or other source. Also, a datapool can have up to 2,147,483,647 rows.

## What Kinds of Problems Does a Datapool Solve?

If you play back a test script just once during a test run, that test script probably does not need to access a datapool.

But often during a test run, and especially during performance testing, you need to run the same test script multiple times. For example:

- During performance testing, you probably want to run multiple instances of a test script so that the test script is executed many times simultaneously. (Remember, a virtual tester is one runtime instance of a test script.)
- During functional and performance testing, you often run multiple iterations of a test script so that the test script is executed many times consecutively (simulating a virtual tester performing the same task over and over).

If the values used in each test script instance and each test script iteration are the same literal values—the values you provided during recording or hand-coded into the test script—you might encounter problems at suite runtime.

Here are some examples of problems that datapools solve:

- *Problem:* During recording, you create a personnel file for a new employee using the employee's unique social security number. Each time the test script is played back, there is an attempt to create the same personnel file and supply the same social security number. The application rejects the duplicate requests.

*Solution:* Use a datapool to send different employee data, including unique social security numbers to the server each time the test script is played back.

- *Problem:* You delete a record during recording. During playback, each instance and iteration of the test script attempts to delete the same record, and "Record Not Found" errors result.

*Solution:* Use a datapool to reference a different record in the deletion request each time the test script is played back.

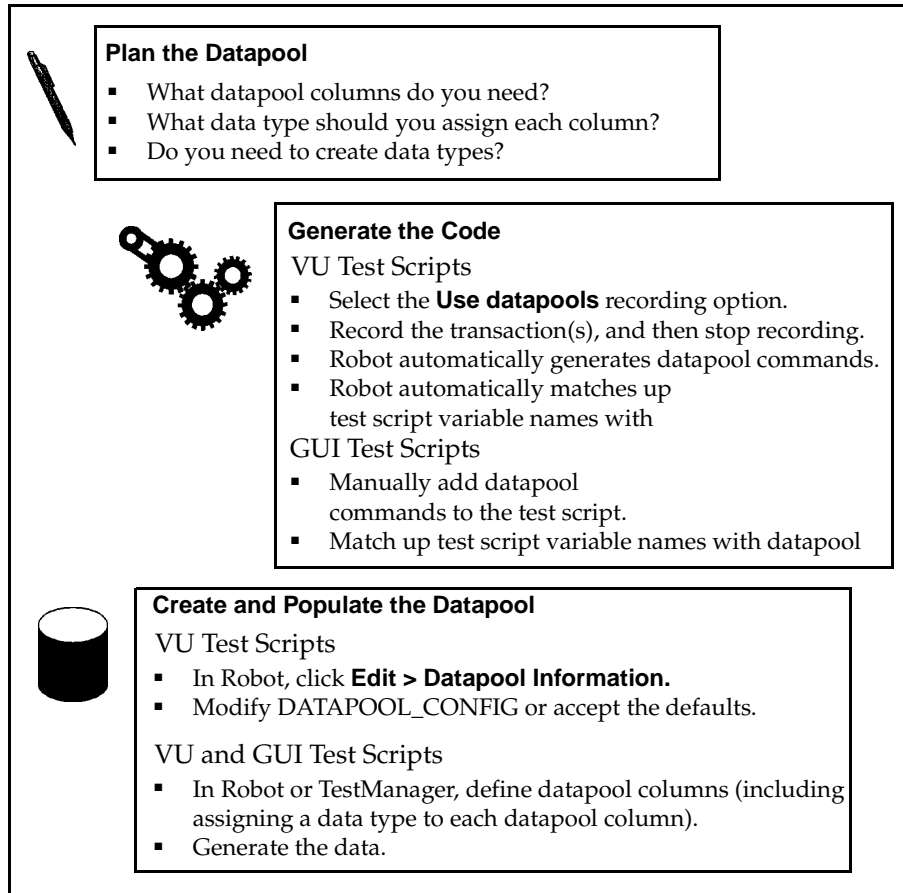
- *Problem:* The client application reads a database record while you record a test script for a performance test. During playback, that same record is read hundreds of times. Because the client application is well designed, it puts the record in cache memory, making retrieval deceptively fast in subsequent fetches. The response times that the performance test yields are inaccurate.

*Solution:* Use a datapool to request a different record each time the test script is played back.

# Planning and Creating a Datapool

---

A summary of the stages involved in preparing a datapool for use in testing is illustrated in the following figure. The order shown is the typical order for planning and creating a datapool for test scripts:



## 1 Plan the datapool.

Determine the datapool columns you need. In other words, what kinds of values (names, addresses, dates, and so on) do you want to retrieve from the datapool and send to the server?

Typically, you need a datapool column for each test script variable to which you plan to assign datapool values during recording.

For example, suppose your client application has a field called **Order Number**. During recording, you type in a value for that field. With VU test scripts, the value is assigned to a test script variable automatically. During playback, that variable can be assigned unique order numbers from a datapool column.

This stage requires some knowledge of the client application and the kinds of data that it processes.

To help you determine the datapool columns you need, record a preliminary test script. Rational Robot automatically captures all of the values supplied to the client application during recording and lists them in the `DATAPOOL_CONFIG` section at the end of the test script. For more information, see *Finding Out Which Data Types You Need* on page 288.

## 2 Generate datapool code.

To access a datapool at runtime, a test script must contain datapool commands, such as commands for opening the datapool and fetching a row of data. With VU test scripts, a `DATAPOOL_CONFIG` section must also be present. This section contains information about how the datapool is created and accessed.

Datapool code is generated in either of the following ways:

- With *VU test scripts*, Robot generates datapool code automatically when you finish recording a session. Robot is aware of all the variables in the session that are assigned values during recording, and it matches up each of these variables with a datapool column in the test script.

To have Robot generate datapool commands automatically during recording, make sure **Use datapools** is selected in the **Generator** tab of the Session Record Options dialog box before you record the test script.

**Note:** You must still supply data to the datapool and enable it.

- With *SQABasic test scripts*, you manually insert the datapool commands and match up test script variables with datapool columns. For information about coding datapool commands, see the *Using Rational Robot* manual.
- For information on using datapools with other script types, see the appropriate Rational Test Script Services API documentation.

## 3 Create and populate the datapool.

After the datapool commands are in the test script, you can create and populate the datapool.

To start creating and populating a datapool for a VU test script you are editing in Robot, click **Edit > Datapool Information**.

Creating and populating a datapool for a test script involves the following general steps:

- Editing the `DATAPOOL_CONFIG` section of the test script. For example, you might want to change the default datapool access flags, or exclude a datapool column from being created for a particular test script variable. Or, you can accept all of the default settings that Robot specifies when it creates this section in a VU test script.

For information about editing the `DATAPOOL_CONFIG` section of a test script, see the *Using Rational Robot* manual.

- Defining the datapool columns that you determined you needed during the planning stage. For example, for an Order Number column, you can specify the maximum number of characters that an order number can have, and whether the Order Number column must contain unique values.

For information about defining datapool columns, see the *Using Rational Robot* manual.

You also assign a data type to each datapool column. Data types supply a datapool column with its values. For information about data types, see *Data Types* on page 286.

- Generating the data. Once you configure the datapool and define its columns, you populate the datapool by clicking **Generate Data**.

**Note:** You can also create and populate a datapool file manually and import it into the datastore. For more information, see *Creating a Datapool Outside Rational Test* on page 306.

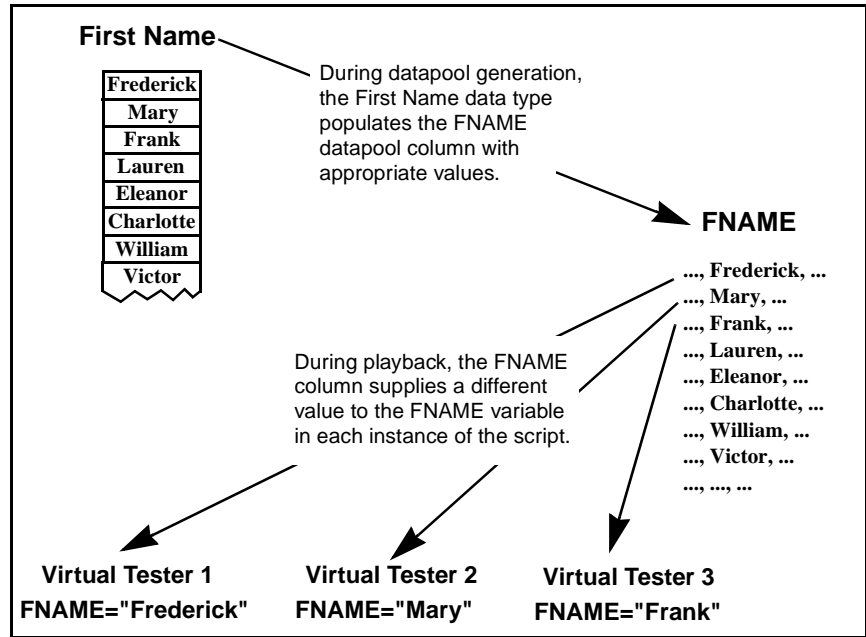
## Data Types

---

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

For example, the Names - First data type (shipped with Rational Test as a standard data type) contains a list of common English first names. Suppose you assign this data type to the datapool column `FNAME`. When you generate the datapool, TestManager populates the `FNAME` column with all of the values in the Names - First data type.

The relationship between data types, datapool columns, and the values assigned to test script variables during playback is shown in the following figure:



## Standard and User-Defined Data Types

There are two kinds of datapool data types, as follows:

- **Standard data types** that are included with Rational Test. These data types include commonly used, realistic sets of data in categories such as first and last names, company names, cities, and numbers.

For a list of the standard data types, see Appendix B, *Standard Datapool Data Types*.

- **User-defined data types** that you create. You must create a data type if none of the standard data types contains the kind of values you want to supply to a test script variable.

User-defined data types are useful in situations such as:

- When a field accepts a limited number of valid values. For example, suppose a datapool column supplies data to a test script variable named *color*. This variable provides the server with the color of a product being ordered. If the product only comes in the colors red, green, blue, yellow, and brown, these are the only values that can be assigned to *color*. No standard data type contains these exact values.

To ensure that the variable is assigned a valid value from the datapool:

- i Before you create the datapool, create a data type named Colors that contains the five supported values (Red, Green, Blue, Yellow, Brown).
  - ii When you create the datapool, assign the Colors data type to the datapool column COLOR. The COLOR column will supply values to the test script's `color` variable.
- When you need to generate data that contains multi-byte characters, such as those used in some foreign-language character sets. For more information, see *Generating Multi-Byte Characters* on page 290.

Before you create a datapool, find out which standard data types you can use as sources of data and which user-defined data types you need to create. Although it is possible to create a data type while you are creating the datapool itself, the process of creating a datapool will be smoother if you create the user-defined data types first.

## Finding Out Which Data Types You Need

To decide whether to assign a standard data type or a user-defined data type to each datapool column, you need to know the kinds of values that will be supplied to test script variables during playback—for example, whether the variable contains names, dates, order numbers, and so on.

To find the kind of values that are supplied to a variable:

- With VU test scripts, view the `DATAPOOL_CONFIG` section of the test script. (Robot adds this information automatically during recording to a VU test script when you select **Use datapools** in the **Generator** tab of the Session Record Options dialog box.)

The `DATAPOOL_CONFIG` section contains a line for each value assigned to a test script variable during recording. In the following example, the value 329781 is assigned to the variable `CUSTID`:

```
INCLUDE, "CUSTID", "string", "329781"
```

For more information about the `DATAPOOL_CONFIG` section of a test script, see the *Using Rational Robot* manual.

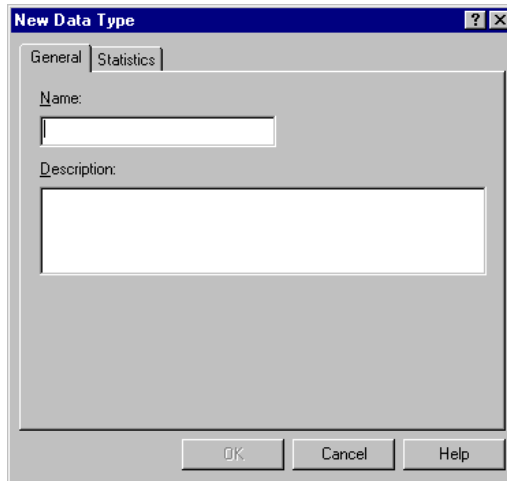
## Creating User-Defined Data Types

If none of the standard data types can provide the correct kind of values to a test script variable, create a user-defined data type.



To create a user-defined data type:

- Click **Tools > Manage > Data Types**, and then click **New**.



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

**Note:** You can assign data from a standard data type to a user-defined data type. For information, see *Editing User-Defined Data Type Definitions* on page 302.

## Generating Unique Values from User-Defined Data Types

You may want a user-defined data type to supply unique values to a test script variable during playback. To do so, the user-defined data type must contain unique values.

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

- Set **Sequence** to Sequential.
- Set **Repeat** to 1.
- Make sure the **No. of records to generate** value does not exceed the number of unique values in your user-defined data type.

For information about the values you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 293.

## Generating Multi-Byte Characters

If you want to include multi-byte characters in your datapool (for example, to support Japanese and other languages that include multi-byte characters), you can do so in either of these ways:

- Through a user-defined data type. For information, see *Creating User-Defined Data Types* on page 288.

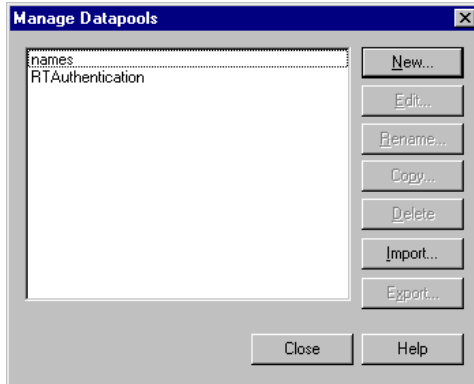
The editor provided for you to supply the user-defined data fully supports Input Method Editor (IME) operation. An IME lets you type multi-byte characters, such as Kanji and Katakana characters as well as multi-byte ASCII, from a standard keyboard. It is included in the Japanese version of Microsoft Windows.

- Through the Read From File data type. For information, see *Creating a Column of Values Outside Rational Test* on page 310.

## Managing Datapools

---

Manage datapools in the Manage Datapools dialog box. The activities you perform in this dialog box affect datapools stored in the current datastore. For information about where datapools are stored, see *Datapool Location* on page 301.



## Creating a Datapool

When you create a datapool using TestManager, you must specify the following:

- Name and description of the datapool.

Choose a name of up to 40 characters. While a description is optional, entering one can help you identify the purpose of the datapool. Datapool descriptions are limited to 255 characters.

- Column names.

Datapool columns are also called *fields*. With VU test scripts, datapool column names must match the names of the test script variables that they supply values to. Names are case-sensitive.

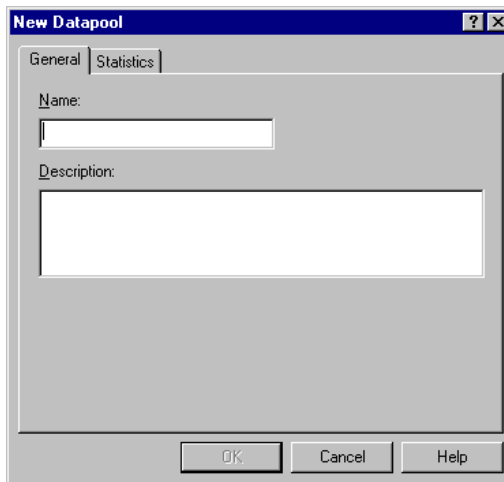
- Data type and properties for each datapool column.

For information about the properties you can define for a datapool column, see *Defining Datapool Columns* on page 293.

- Number of records to generate.

To create and automatically populate a datapool:

- Click **Tools > Manage > Datapools**, and then click **New**.



## If There Are Errors

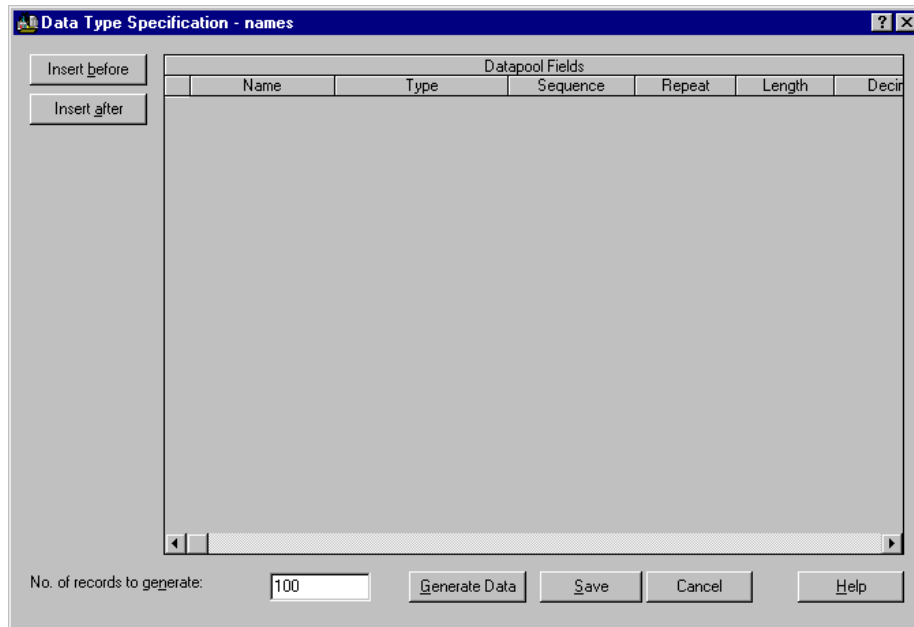
If the datapool values are not successfully generated, TestManager asks if you want to see an error report rather than a summary of the generated data. Viewing this report can help you identify where to make corrections in the datapool fields. To view an error report, click **Yes** when TestManager asks whether you want to see an error report or summary data.

## Viewing Datapool Values

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

To see the generated values:

- In the Manage Datapools dialog box, select the datapool you just created, click **Edit**, and then click **Edit Datapool Data**.



## Making the Datapool Available to a Test Script

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

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

- For information about automatically adding datapool commands and `DATAPOOL_CONFIG` to a test script during recording, see the *Using Rational Robot* manual.

## Defining Datapool Columns

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

Grid column	Description
<p><b>Name</b></p>	<p>The name of a datapool column (and its corresponding test script variable).</p> <p>If you change the name of a datapool column, be sure the new name matches all instances of its corresponding test script variable.</p> <p>If you create a datapool outside of the Rational Test environment and then import it, TestManager automatically assigns default names to the datapool columns. Use <b>Name</b> to match the imported datapool column names with their corresponding test script variables. Names are case-sensitive.</p> <p>You can use an IME to type multi-byte characters in datapool field names.</p>
<p><b>Type</b></p>	<p>The standard or user-defined data type that supplies values to the datapool column in <b>Name</b>. User-defined data types are marked with an asterisk (*).</p> <p>Specify the data type to assign to the datapool column, as follows:</p> <ul style="list-style-type: none"> <li>▪ To select a standard data type or an existing user-defined data type, click the currently displayed data type name, and then select the new data type from the drop-down list:           <div style="border: 1px solid black; padding: 2px; display: inline-block; margin: 5px 0;">Random alphanumeric stri ▾</div> </li> <li>▪ See Appendix B for a description of the standard data types.</li> <li>▪ If you type rather than select the name of a user-defined data type, enter an asterisk before the user-defined data type name. For example, to specify the user-defined data type MyData, type:           <p style="margin-left: 20px;">*MyData</p> </li> <li>▪ To create a new user-defined data type, enter the data type name (without the asterisk) in the field, and then press RETURN. After you click <b>Yes</b> to confirm that you want to create a user-defined data type, the Data Type Properties - Edit dialog box appears.</li> <li>▪ For information about creating a data type, see <i>Creating User-Defined Data Types</i> on page 288.</li> </ul>

Grid column	Description
<b>Sequence</b>	<p>The order in which the values in the data type specified in <b>Type</b> are written to the datapool column. Select one of these options from the drop-down list:</p> <ul style="list-style-type: none"> <li>▪ <b>Random</b> – Writes numeric and alphanumeric values to the datapool column in any order.</li> <li>▪ <b>Sequential</b> – Writes numeric values sequentially (for example, 0, 1, 2...). With decimal numbers, the sequence is based on the lowest possible decimal increment (for example, with a <b>Decimals</b> value of 2, the sequential values are 0.00, 0.01, 0.02, ...).</li> <li>▪ <b>Sequential</b> is only supported for numeric values (including date and time values) and values generated from user-defined data types.</li> </ul> <p>When you select Sequential with numeric data types, and you specify a <b>Minimum</b> and <b>Maximum</b> range, <b>Interval</b> must be greater than 0.</p> <ul style="list-style-type: none"> <li>▪ <b>Unique</b> – With data type Integers - Signed, ensures that numbers written to the datapool column are unique. Also, set <b>Repeat</b> to 1, and define a <b>Minimum</b> and <b>Maximum</b> range.</li> </ul> <p>Do not confuse the <b>Random</b> and <b>Sequential</b> settings in this grid with <b>Random</b> and <b>Sequential</b> access order in the Configure Datapool in Script dialog box. The <b>Random</b> and <b>Sequential</b> settings in this grid determine the order in which values are written to an individual datapool column at datapool creation time. <b>Random</b> and <b>Sequential</b> access order determine the order in which virtual testers access datapool rows at suite runtime.</p>
<b>Repeat</b>	<p>The number of times a given value can appear in a datapool column. <b>Repeat</b> cannot be set to 0.</p> <p>To make values unique with Integers - Signed data types and user-defined data types, set <b>Repeat</b> to 1. For unique Integers - Signed values, also set <b>Sequence</b> to either Sequential or Unique.</p> <p>When defining unique values, make sure the number of rows you are generating is not higher than the range of possible unique values.</p>
<b>Length</b>	<p>The maximum number of characters that a value in the datapool column can have. If the datapool column contains numeric values, <b>Length</b> specifies the maximum number of characters a number can have, <i>including</i> a decimal point and minus sign, if any.</p> <p>For example, for decimal numbers as high as 999.99, set <b>Length</b> to 6. For decimal numbers as low as -999.99, set <b>Length</b> to 7.</p> <p><b>Length</b> cannot be 0.</p>
<b>Decimals</b>	<p>Specifies the maximum number of decimal places that floating point values can have. Maximum setting is 6 decimal places.</p>

Grid column	Description
<b>Interval</b>	<p>Writes a sequence of numeric values to the datapool column. The sequence increments by the interval you set. For example, if <b>Interval</b> is 10, the datapool column contains 0, 10, 20, and so on. If <b>Interval</b> is 10 and <b>Decimal</b> is 2, the datapool column contains 0.00, 0.10, 0.20, and so on.</p> <p>Minimum interval is 1. Maximum interval is 999999.</p> <p>With numeric data types (including dates and times), when <b>Sequence</b> is set to Sequential and you specify a <b>Minimum</b> and <b>Maximum</b> range, <b>Interval</b> must be greater than 0.</p> <p>Use <b>Interval</b> only with numeric values (including dates and times).</p>
<b>Minimum</b>	<p>Specifies the lowest in a range of numeric values. For example, if the datapool column supplies order number values, and the lowest possible order number is 10000, set <b>Type</b> to Integer - Signed, <b>Minimum</b> to 10000, and <b>Maximum</b> to the highest possible order number.</p> <p>Use <b>Minimum</b> only with numeric values (including dates and times).</p>
<b>Maximum</b>	<p>Specifies the highest in a range of numeric values. For example, if the datapool column supplies values to a variable named <i>ounces</i>, set <b>Type</b> to Integer - Signed, <b>Minimum</b> to 0, and <b>Maximum</b> to 16.</p> <p>Use <b>Maximum</b> only with numeric values (including dates and times).</p>
<b>Seed</b>	<p>The number that Rational Test uses to compute random values. The same seed number always results in the same random sequence. To change the random sequence, change the seed number.</p>
<b>Data File</b>	<p>The path to the user-defined data type file. The path is automatically inserted for you. This field is not modifiable.</p> <p>Data type files are stored in the Datatype directory of your project. You never have to modify these files directly.</p>

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

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

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

## Example of Datapool Column Definition

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

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

After you record the test script, create the datapool. Define the datapool's columns in the Datapool Specification dialog box, as illustrated below:

Float data type with 0 decimals is used for credit card numbers

Customer ID is unique

Date range

Datapool column 1

		Datapool Fields							
Name	Type	Sequence	Repeat	Length	Decimals	Interval	Minimum	Maximum	
fName	Names - First	Random	1	20	0	0	0	0	
lName	Names - Last	Random	1	20	0	0	0	0	
custID	Integer - Signed	Unique	1	7	0	0	1000000	9999999	
ccNum	Float - XXXX	Random	1	16	0	0	1000000000000000	9999999999999999	
ccType	*Credit Card Type	Random	1	15	0	0	0	0	
ccExpDate	Date - MM/DD/YYYY	Random	1	15	0	0	07011998	12312002	

No. of records to generate: 1000

Generate Data

Save Cancel Help

Generate 1000 datapool rows

The only user-defined data type needed

Note the following datapool column definition highlights:

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

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



- **ccNum** column. The Integer - Signed data type generates numbers up to nine digits.  
**Note:** Because credit card numbers contain more than nine digits, the standard data type Float X.XXX is used to supply credit card numbers to this datapool column.
- **Decimals** is set to 0 so that only whole numbers are generated. **Sequence** is set to Random to generate random card numbers. To generate unique numbers, **Repeat** is set to 1.
- **ccType** column. This is the only datapool column that needs to have values supplied from a user-defined data type. The user-defined data type Credit Card Type contains just four values—American Express, Discover, MasterCard, and Visa.
- **ccExpDate** column. The standard data type Date - MM/DD/YYYY supplies credit card expiration dates to this datapool column. The range of valid expiration dates is set from July 1, 1998 through December 31, 2002. **Sequence** is set to Random to generate random dates.

## Example of Datapool Value Generation

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

To see the values you generated:

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

This is what you see:

Drag this vertical bar to change column width.

fName	lName	custID	ccNum	ccType	ccExpDate
Karen	Conway	2445267	5068543149543072	Visa	09/13/1998
Koganti	Pistora	7327429	8996616484463400	Discover	01/18/2000
Clarence	Kaffen	4950796	3680866544363417	Discover	07/28/1999
Pam	Elders	3211256	63678330561131982	Visa	07/29/2000
Izzy	Nies	6821857	6517053874186043	Visa	08/04/2001
Gene	McByaer	6198389	9892851646473812	MasterCard	09/22/2001
Linsey	Randolph	7593933	3889780335422292	Discover	10/12/2001
Napoleon	Seibert	8266508	4900680213960734	Discover	04/20/2000
Lester	Quit	7541706	7510525399071469	Discover	03/07/2001
Norman	Sicoli	8419493	2649627877687276	Visa	02/07/2000
Mike	Sandy	8039891	4251911486691095	Discover	08/23/2001
Geniet	McDonagh	6261296	8630209049474339	Visa	10/28/2001
Toni	Garavalia	3743098	4335144215955984	MasterCard	02/11/1998
Rex	Sullins	8874076	2648600742281909	Visa	11/30/1998
Linda	Dalton	2650643	1162743126430160	Visa	03/24/2000
Emanuele	Lioce	5753551	5849409703724003	MasterCard	01/24/2001
Ginger	Bucchi	1946446	4251965524822971	Visa	03/28/1999
Murry	Scragg	8241330	9779549341305014	Visa	04/13/2001
Israel	Nichlen	6797118	2787256474102288	Discover	07/31/1999
Lyle	Reutz	7710609	4895042872558646	MasterCard	11/12/1999
Claudette	Beattie	1421162	9696045256296100	American Express	11/03/2000
Hennietta	Byers	2052623	8939704872609540	Visa	06/16/2001
Cathi	Barbieri	1327082	7245010991034065	Visa	08/23/2000
Jackson	Noack	6833313	677863533843530	Visa	01/07/1998
Michele	Dughman	3073764	5981411390202104	Discover	09/20/1998

## Editing Datapool Column Definitions

The Datapool Specification dialog box allows you to define and edit the columns in the datapool file. Datapool column definitions are listed as rows in this dialog box. Datapool columns are also called *fields*.

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

- Click **Tools > Manage > Datapools**, select the datapool to edit, and then click **Edit**.

When you finish editing datapool column definitions, choose whether to generate data for the datapool.

To see the generated values:

- In the Datapool Properties - Edit dialog box, click **Edit Datapool Data**.

If the datapool values are not successfully generated, TestManager asks if you want to see an error report rather than a summary of the generated data. Viewing this error report can help you identify where to make corrections in the datapool fields. To view an error report, click **Yes** when TestManager asks whether you want to see an error report or summary data.

## Deleting a Datapool Column

Datapool column definitions are listed as rows in the Datapool Specification dialog box. To delete a datapool column definition from the list, select the row to be deleted and press the `DELETE` key.

## Editing Datapool Values

To view or edit the values in an existing datapool:

- Click **Tools > Manage > Datapools**, select the datapool to edit, and then click **Edit**.

When modifying the values in an existing datapool, note that:

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

fName	lName	custID
Karen	Conway	2445267
Koganti	Pistora	7327429

For an example of the datapool values that TestManager generates, see *Example of Datapool Value Generation* on page 297.

## Renaming or Copying a Datapool

When you rename or copy a datapool, you must specify a new name for the datapool, up to a maximum of 40 characters.

To rename or copy a datapool:

- Click **Tools > Manage > Datapools**.

## Deleting a Datapool

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

To delete a datapool:

- Click **Tools > Manage > Datapools**.

## Importing a Datapool

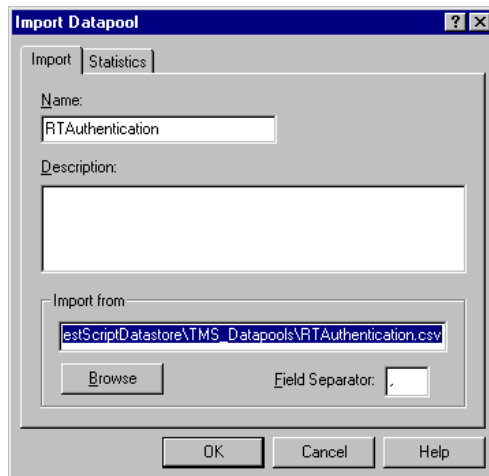
It is possible for you to create and populate a datapool yourself, using a tool such as Microsoft Excel. For example, you might want to export data from your database into a .csv file, and use that file as your datapool.

If you create a datapool yourself, you need to import it into the same datastore as the test scripts that will access it. You can use TestManager to import a datapool .csv file.

When you import a datapool, you often have to change the names of the datapool columns to match the names of the corresponding test script variables. For more information, see *Matching Datapool Columns with Test Script Variables* on page 309.

To import a datapool .csv file:

- Click **Tools > Manage > Datapools**, and then click **Import**.



## Datapool Location

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

For example, if the current project is MyProject, and the current datastore directory is MyDatastore, then the datapool is stored in the following directory:

```
C:\MyProject\MyDatastore\DefaultTestScriptDatastore\TMS_Datapool
```

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

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

## Importing a Datapool from Another Project

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

To import a datapool into a new project:

- Click **Tools > Manage > Datapools**, and then click **Import**.

**Note:** If the datapool that you are importing includes user-defined data types, import the data types *before* you import the datapool. For information, see *Importing a User-Defined Data Type* on page 303.

## Exporting a Datapool

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

Do not attempt to export a datapool to another Rational Test project. Instead, use the import feature to import the datapool into the new project. For more information, see *Importing a Datapool* on page 300.

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

- Click **Tools > Manage > Datapools**.

## Managing User-Defined Data Types

---

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

For information about creating user-defined data types, see *Data Types* on page 286.

### Editing User-Defined Data Type Values

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

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

To edit a user-defined data type:

- Click **Tools > Manage > Data Types**.

### Editing User-Defined Data Type Definitions

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

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

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

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

- Click **Tools > Manage > Data Types**.

You can also add values to a user-defined data type by supplying it with values from a standard data type. This automatic generation of values by TestManager can reduce the typing that you need to perform when adding values to the user-defined data type.

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

Rather than typing in all 100 numbers, with dashes, you can have TestManager generate the numbers and assign them to a user-defined data type. Then, you can edit the data type values and each ID.

When you choose to automatically generate values, you must specify guideline values for TestManager to use during generation. These values include:

- **Type** = Integers - Signed
- **Sequence** = Sequential
- **Repeat** = 1
- **Length** = 7
- **Interval** = 1
- **Minimum** = 1000001
- **Maximum** = 1000100
- **No. of records to generate**

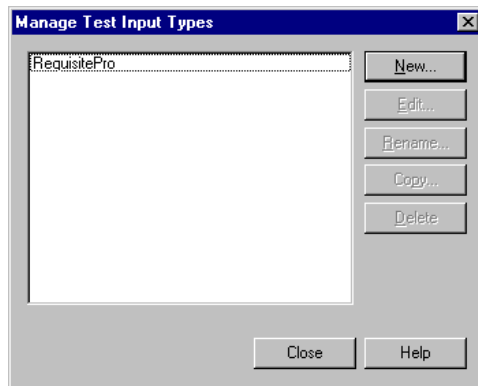
**Note:** You can also assign the standard data type Read From File to a user-defined data type. For information about using the Read From File data type, see *Creating a Column of Values Outside Rational Test* on page 310.

## Importing a User-Defined Data Type

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

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

- Click **Tools > Manage > Test Input Types**.



## Renaming or Copying a User-Defined Data Type

When you rename or copy a user-defined data type, you must specify a new name for the data type, up to a maximum of 40 characters.

To rename or copy a user-defined data type:

- Click **Tools > Manage > Data Types**.

## Deleting a User-Defined Data Type

To delete a user-defined data type in TestManager:

- Click **Tools > Manage > Data Types**.

## Generating and Retrieving Unique Datapool Rows

---

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

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

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

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

Use the following guidelines when the datapool is being accessed by either a single test script or by multiple test scripts, including both VU and GUI test scripts.



## What You Can Do to Guarantee Unique Row Retrieval

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

What to do	How to do it
Specify at least one column of unique data.	<p>In the Datapool Specification dialog box, specify that at least one datapool column should contain unique data. Unique data can be supplied through the Integers - Signed data type, through the Read From File data type, and through user-defined data types.</p> <p>With the Integers - Signed data type, take all of these actions:</p> <ul style="list-style-type: none"> <li>▪ Set <b>Sequence</b> to Unique or Sequential.</li> <li>▪ Set <b>Repeat</b> to 1.</li> <li>▪ If <b>Sequence</b>=Unique, set an appropriate range in <b>Minimum</b> and <b>Maximum</b>.</li> <li>▪ Make sure that the values of <b>Length</b> and <b>No. of records to generate</b> are appropriate for the set of numbers to generate.</li> </ul> <p>With the Read From File data type, see <i>Generating Unique Values</i> on page 311 for information.</p> <p>With user-defined data types, see <i>Generating Unique Values from User-Defined Data Types</i> on page 289 for information.</p>
Generate enough datapool rows.	<p>Generate at least as many unique datapool rows as the number of times the datapool will be accessed during a test.</p> <p>For example, if 50 virtual testers will access a datapool during a test, and each virtual tester is set for 3 iterations each, the datapool must contain at least 150 rows.</p> <p>You specify the number of rows to generate in the <b>No. of records to generate</b> field of the Datapool Specification dialog box.</p>
Disable cursor wrapping.	<p>If the datapool cursor wraps after the last row in the datapool has been accessed, previously fetched rows are fetched again.</p> <p>Disable cursor wrapping in any of these ways:</p> <ul style="list-style-type: none"> <li>▪ When editing the DATAPOOL_CONFIG section of a VU test script in the Configure Datapool in Script dialog box, set <b>Wrap at end of file?</b> to <b>No</b>.</li> <li>▪ When editing a VU test script in Robot, add DP_NOWRAP to the list of flags in the <i>flags</i> argument of the DATAPOOL_CONFIG statement or the datapool_open function.</li> <li>▪ When editing a GUI test script in Robot, set the <i>wrap</i> argument of the SQADatapoolOpen command to <b>False</b>.</li> </ul>

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

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

## Creating a Datapool Outside Rational Test

---

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

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

After you create and populate a datapool, you can use TestManager to import the datapool into the datastore. For information about importing a datapool, see *Importing a Datapool* on page 300.

## Datapool Structure

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

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

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

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

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

The quotes are used only when the value is stored in the datapool file. The quotes are not part of the value that is supplied to your application.

- A single value can contain embedded strings. For example, "Jones, Robert "Bob"" is a single value in a record, not two.
- Each record ends with a line feed.
- Datapool column names are stored in a .spc file. (Robot and TestManager edit the .spc file. Never edit the .spc file directly.)
- The datapool name that is stored in the datastore is the same as the root datapool file name (without the .csv extension). The maximum length of a datapool name is 40 characters.

## Example Datapool

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

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

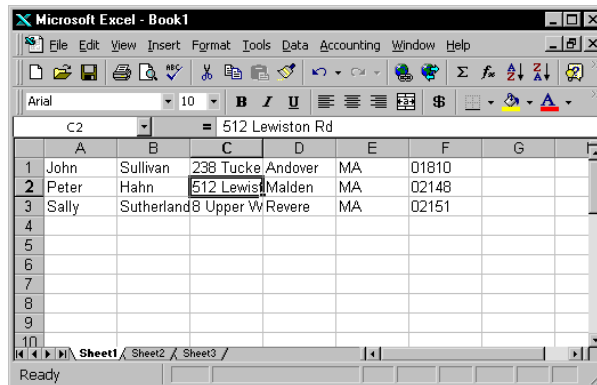
## Using Microsoft Excel to Create Datapool Data

When you are using Microsoft Excel to populate a datapool, do not separate values with the Windows separator character (see page 307). Excel automatically inserts the separator character when you save the datapool in .csv format.

To create and populate a datapool using Microsoft Excel:

- Click **File > New** to create a new Excel workbook.

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



Note that:

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

## Saving the Datapool in Excel

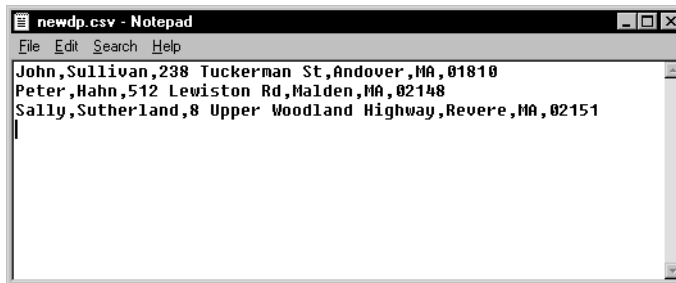
When you finish adding rows of values to the datapool, save the datapool to .csv format.

To save a datapool file using Microsoft Excel:

- Click **File > Save As**.

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

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



## Matching Datapool Columns with Test Script Variables

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

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

For information about how to open the Datapool Specification dialog box during datapool editing, see *Editing Datapool Column Definitions* on page 298.

## Maximum Number of Imported Columns

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

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

## Creating a Column of Values Outside Rational Test

---

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

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

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

### Step 1. Create the File

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

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

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

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

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

To do so, from the Datapool Specification dialog box, in the **Type** column, select the data type Read From File for the datapool column being supplied the values from the external text file.

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

## Generating Unique Values

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

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

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

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

For information about the values you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 293.





This chapter discusses performance testing reports and suggests ways to evaluate the data provided in them. It includes the following topics:

- About reports
- Running a report
- Customizing reports
- Changing report defaults
- Types of reports

## About Reports

---

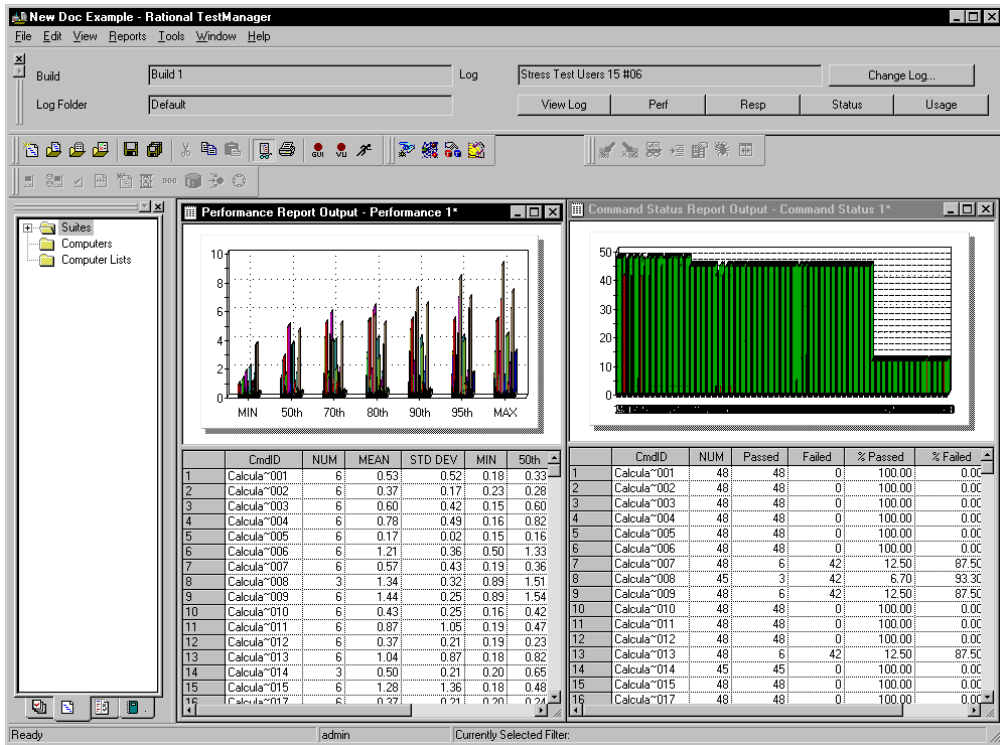
TestManager provides several types of reports that help you analyze the success or failure of a given suite run, and the performance of the server under specified conditions. For example, you can determine how long it took for a virtual tester to execute a command and how response times varied with different suite runs.

You can define new reports based on standard report types. Custom reports can help you zoom in on a given application element and further refine tests to show exactly the data you need as determined by your test plan or test case.

TestManager does not differentiate between the report definition and the processed report data. Most actions that you can perform on a report definition you can also perform on the processed report data.

By default, if a test completes successfully and the test generates appropriate data, TestManager automatically runs Command Status and Performance reports against the data in the log and displays the processed results.

The following figure shows a sample Command Status and a sample Performance report:



After you examine report data, you can save or delete the report. If you save the report, TestManager gives it a default name based on the type of report and the number of existing reports of that type. (For example, Performance 1.) TestManager saves the report under the logs in the project. To view the report again, you can open the saved report. If you delete the report, you can re-create it by running the same type of report against the same log.

The following table summarizes the different types of reports:

Report	Function	Information
Performance	<p>Display the response times, and calculate the mean, standard deviation, and percentiles for each command in the suite run.</p> <p>The report groups responses by command ID and shows only responses that passed. In contrast, Response vs. Time reports show each command ID individually and show passed and failed responses.</p>	<i>Performance Reports</i> on page 332
Compare Performance	Compare the response times measured by Performance reports. After you have generated several Performance reports, use the Compare Performance report to compare specific data.	<i>Compare Performance Reports</i> on page 335
Response vs. Time	<p>Display individual response times and whether a response has passed or failed. This report is useful for looking at data points for individual responses as well as trends in the data.</p> <p>The report shows each command ID individually and the status of the response. In contrast, the Performance reports group responses by command ID and they show only passed responses.</p> <p>You can right-click on the report, select a computer that was in the run, and graph the resource monitoring statistics for that computer. These are the same statistics that you display when you choose to monitor resources during a suite run.</p>	<i>Response vs. Time Reports</i> on page 341
Command Status	Obtain a quick summary of which and how many commands passed or failed. The report displays the status of all emulation commands and SQABasic timer commands.	<i>Command Status Reports</i> on page 344
Command Usage	View cumulative response time and summary statistics, as well as throughput information for emulation commands for all scripts and for the suite run as a whole.	<i>Command Usage Reports</i> on page 346

**Note:** Users who upgraded from Rational Suite PerformanceStudio or Rational LoadTest may want to access information that previously was available in the Analog and Trace reports. The information in those reports is now available through the Test Log window. For information on viewing and using test logs, see Chapter 6, *Evaluating Tests*.

## Running a Report

---

TestManager automatically runs the default Performance and Command Status reports at the end of a suite run (unless the suite was aborted and log data was not generated). While these reports offer a significant amount of information about your test run, you might want to run other reports and/or vary the information displayed in any given report. This section describes how to run different reports.

You run reports from the Report bar or from TestManager menus.

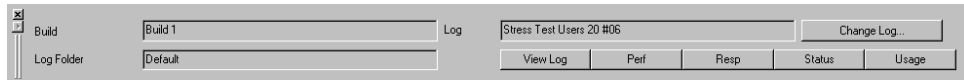
**Note:** You might also want to view the log files—the “raw” result files—before you run reports against them. For information about viewing log information in the Test Log window, see Chapter 6, *Evaluating Tests*.

### Running a Report from the Report Bar

The quickest way to run a report is to click a button on the Report bar. On the Report bar, TestManager lists the log created by the last suite you ran. Unless you specify another log, TestManager runs the report using the information in this log.

To run a report from the Report bar:

- Click **View > Report Bar**, and then click any one of the report buttons.



**Note:** You can customize the Report bar by populating it with your own reports. For more information, see *Changing the Reports that Run from the Report Bar* on page 331.

### Running a Report from the Menu Bar

Although TestManager lets you run reports quickly from the Report bar, you can run only one report of each type against a log in this way. However, you might want to run a number of reports from a series of logs. For example, if you have defined some new Performance reports, you might want to run each report against the same log. You can run these reports from the menu bar.

To run a report from the menu bar:

- Click **Reports > Run**, and select the type of report to run.

# Customizing Reports

---

TestManager lets you customize reports for your particular testing requirements.

You can customize a report by:

- Filtering the data.

For example, you can filter the report so that it contains only one virtual tester group, only certain test scripts, and only certain command IDs.

- Changing a report's advanced options.

For example, you can modify a Response vs. Time report so that extremely long responses are not included in the report.

- Changing a graph's type and appearance.

For example, you can display a graph as a line graph or a bar graph.

After you have customized a report and saved it, you can use it repeatedly to quickly analyze your data.

## Filtering Report Data

TestManager provides a set of default reports with predefined settings and options. You can, however, customize reports to filter only certain data.

For example, the Performance report on page 314 contains information from many command IDs, and the graph is complex. To see fewer command IDs, zoom in on the graph, as explained on page 326. Alternately, right-click the report, click **Settings**, and then click **Select Command IDs**.

However, instead of filtering the processed report, it is much easier to filter the report definition beforehand so that the resulting report contains only the information you are interested in. You can filter a report so it includes only certain virtual testers, only certain test scripts, or only certain commands.

When you choose to set up filtering in a report, you must specify the following information, depending on the type of report:

- Build and log information

The build and log folder for which to look for appropriate log files, and the specific log file on which to filter data in the report.

- Virtual Testers

The virtual tester and/or groups (computer or user) associated with the specified log on which to filter data.

- **Scripts**

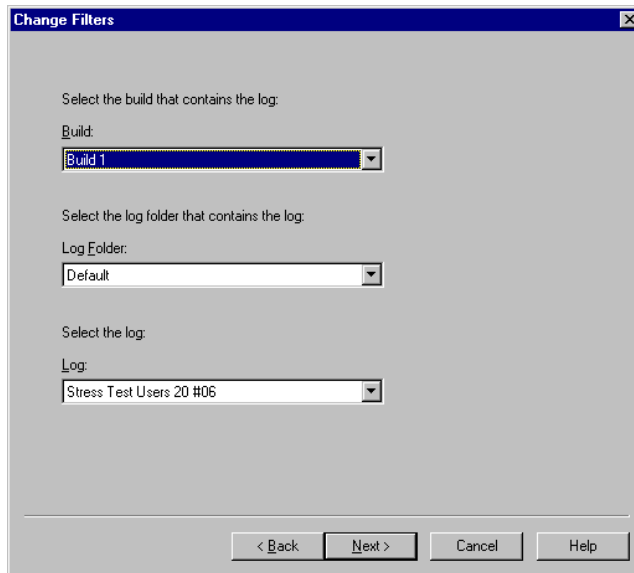
The test scripts associated with the selected virtual testers on which to filter data.

- **Command IDs**

The command IDs specified in the selected test scripts on which to filter data.

To set up filtering in Performance, Response vs. Time, Command Status, and Command Usage Reports:

- Open or create a new report of that type, and then click **Change Filters**.



**Note:** If you are filtering virtual testers, you usually select the log with the largest number of virtual testers. This ensures that your report filters all of the virtual testers.

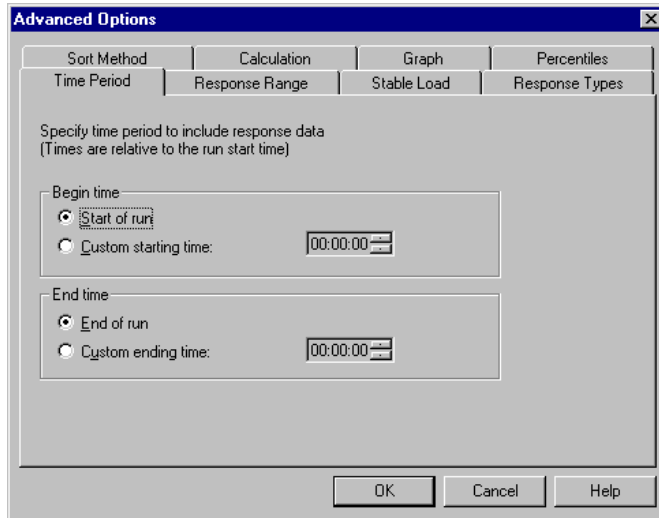
You can also filter the report *after* you run it. For more information, see *Filtering Command IDs that Appear in a Graph* on page 327.

## Setting Advanced Options

All TestManager reports have advanced options, which determine how the report data is calculated and displayed. The specific advanced options are different for each report. To fine tune a report, change the advanced options.

To see the advanced options for a report:

- Open or create a new report, and then click the **Change Options** button.



**Note:** For more information about advanced options, see the TestManager Help.

The following table summarizes each advanced option, and lists the reports that use the option:

Option	Description	Reports
<b>Graph</b>	Display the report as a graph, a table, or both, change the type of graph displayed, change the labels for the graph axes, and add headers and footers.	Command Status, Performance, Response vs. Time, Compare Performance
<b>Response Range</b>	Include only responses that fall between a maximum and minimum time. The default includes all response times.  you might want to set a maximum response time to eliminate outliers. If you change this option for one report, change the other reports, too, so that the reports reflect the same information. For more information, see <i>Eliminating Outliers</i> on page 321.	Command Status, Performance, Response vs. Time, Compare Performance
<b>Response Types</b>	Include only HTTP responses or responses with timers. The default includes all responses. The Command Status and Response vs. Time reports also let you filter responses that contain verification points.	Command Status, Performance, Response vs. Time

Option	Description	Reports
<b>Sort Method</b>	Sort command IDs numerically or in the order in which they were run. The default is to sort command IDs alphabetically.	Command Status, Performance, Response Vs. Time
<b>Stable Load</b>	<p>Specify a number of virtual testers that must be logged on before results are reported. The default is to report results when any number of virtual testers are logged on. you might want to change this option so that a certain number of virtual testers, or all virtual testers, must be logged on. For more information, see <i>Reporting on a Stable Load</i> on page 321.</p> <p>If you dynamically added virtual testers when you monitored the suite, you might want to change this option to correspond with the numbers of virtual testers you added. For more information, see <i>Reporting on a Dynamic Number of Virtual Testers</i> on page 322.</p> <p>If you change this option for one report, change the other reports, too, so that the reports reflect the same information.</p>	Command Status, Performance, Response vs. Time
<b>Time Period</b>	Report on a specific portion of the suite run. The default is to report on the entire run.	Command Status, Performance, Response vs. Time
<b>Calculation</b>	Change how response times are calculated. The default measures the time from the end of the last send command until the last byte of the response is received. If you change this option for one report, change the other reports, too, so that the reports reflect the same information.	Performance, Response vs. Time
<b>Response Status</b>	Include only passed responses, or only failed responses. The default is to include all responses.	Response vs. Time
<b>Summary</b>	Summarize data by virtual tester, test script, command ID (Command Status), or run (Command Usage). The default for the Command Status report is detailed by command ID; the default for the Command Usage report is by run.	Command Status, Command Usage
<b>Percentiles</b>	Change how the response times are grouped. Generally, the defaults of 50, 70, 80, 90, and 95 are adequate.	Performance

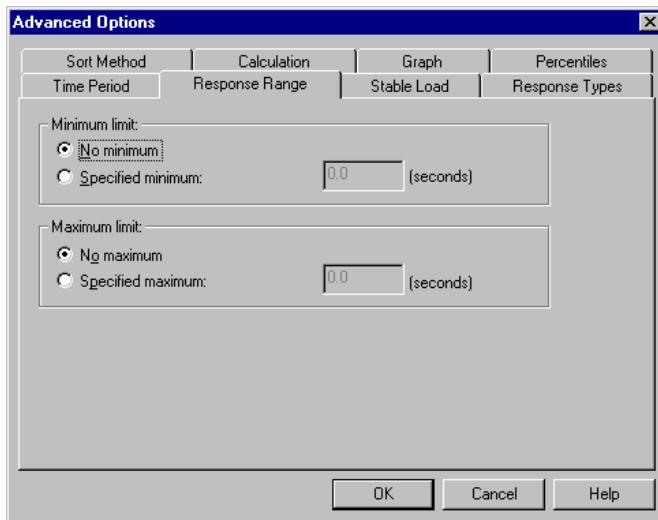


## Eliminating Outliers

Reports may contain some values, called *outliers*, that are completely out of the normal range. For example, you run a Performance report on 1000 virtual testers and most response times range from 2 to 7 seconds. The response for one command ID is 30 seconds—far more than normal. Since this occurs only once it may be a nonrepresentative time. In some cases you might want to eliminate such data points from the report because they may inaccurately skew cumulative data.

To eliminate outliers:

- In the **Response Range** tab in the Advanced Options dialog box, specify a maximum limit for a response time.



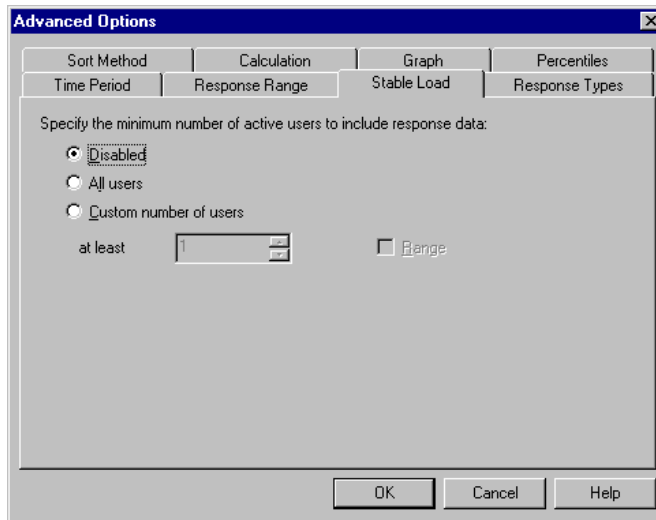
**Note:** Consider carefully whether to remove data points from graphs. While at times outlying data may non-representative, in other cases outliers could be indicative of other performance issues.

## Reporting on a Stable Load

It is useful to limit your report so that it includes only times when you have a stable virtual tester load. For example, you probably are not interested in response times when only a few virtual testers have logged on to the system, or when most of the virtual testers have logged off.

To specify a stable load:

- On the **Stable Load** tab in the Advanced Options dialog box, specify the minimum number of virtual testers that you consider to be a stable load.



## Reporting on a Dynamic Number of Virtual Testers

If you add virtual testers dynamically when running a suite, you will want to know how this addition affects your results. For example, if you start a suite with 50 virtual testers, and then dynamically add three more groups of 50, your reports should show ranges of 50–100, 101–150, 151–200.

To report on a dynamic number of virtual testers:

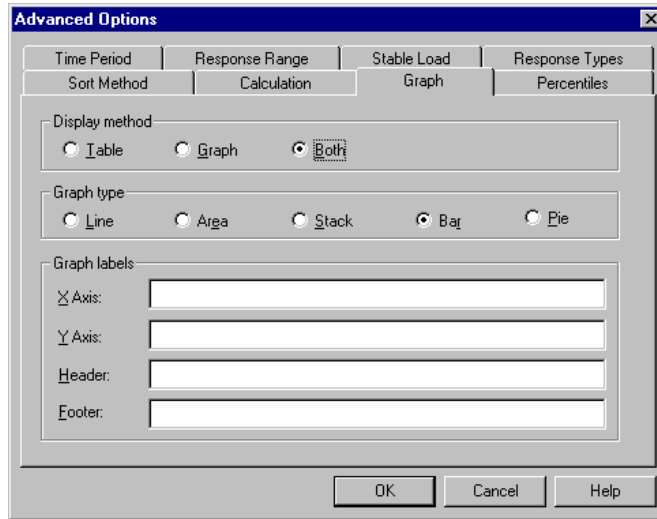
- On the **Stable Load** tab in the Advanced Options dialog box, specify the number of virtual testers.
- On the **Response Range** tab in the Advanced Options dialog box, specify a range for a dynamic number of virtual testers.

## Reporting on a Particular Command ID

The default Response vs. Time report can look confusing because it contains information about every command ID. This information is useful for assessing trends in the data. However, you might want to report on a particular command ID or a small group of command IDs, and display the report in a line histogram, which is easier to read.

To report on a particular command ID and then display it as a line histogram:

- On the **Graph** tab of the Advanced Options dialog box, specify the graph type after filtering the report on the command ID.



## Mapping Computer Resource Usage onto Response Time

Monitoring computer resources is essential in performance testing. If you have a performance problem, you need to determine whether it is caused by a large number of virtual testers or by a hardware bottleneck. The Response vs. Time report lets you overlay computer resource statistics over response time. If your response time increases, you can determine whether this was caused by a computer resource problem.

**Note:** TestManager needs to be set up to collect the information on computer resources before you can report on computer resources. Therefore, when you run a suite, select the **Monitor resources** check box. For information on running suites, see Chapter 11, *Creating Performance Testing Suites*.

To map computer resources onto response time:

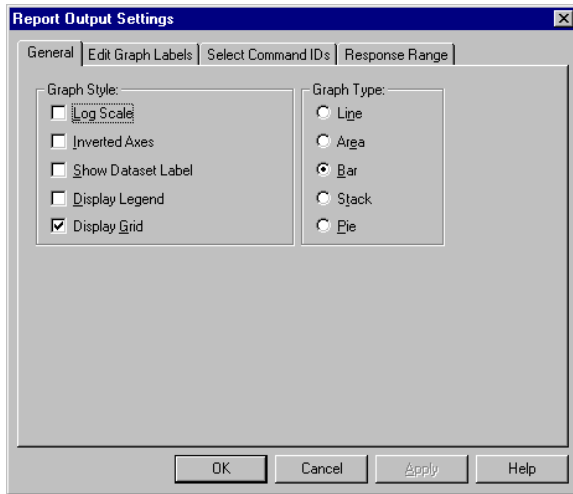
- Right-click on the graph of the Response vs. Time report, and then click **Show Resources**.

## Changing a Graph's Appearance or Type

TestManager can display the Compare Performance, Performance, Response vs. Time, and Command Status reports as both graphs and table-style reports. The Report Output Settings dialog box lets you change the type of graph that appears and lets you enhance its display.

To change the type or appearance of a graph:

- From an open report, click **View > Settings**.



**Note:** Available options for changing the type or appearance of a graph vary according to the type of report you selected.

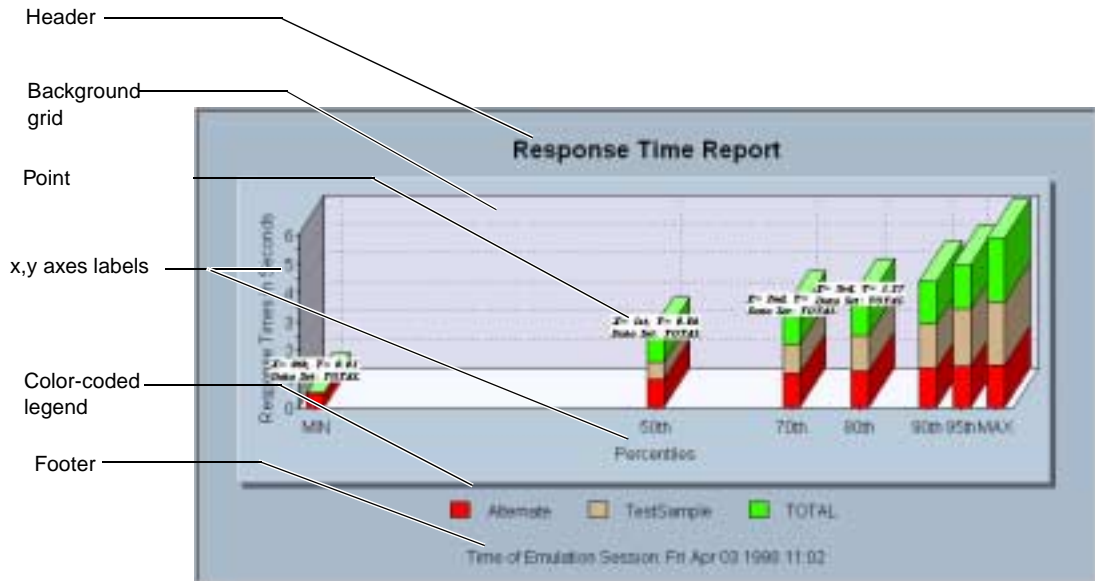
From this dialog box, you can:

- Change the appearance of a graph.
- Change the labels of a graph.
- Filter information such as the command IDs.

In a Performance report, you can also change the response range that appears in the graph.

## Changing a Graph's Appearance

TestManager lets you control a graph's format and appearance. You can display or clear information about selected points and datasets without affecting the graph's cumulative data. The following figure shows a stack graph with a header, background grid, and various other options:



Graph options that you can change include:

- **Log Scale** – Scales any graphical display type to its logarithmic equivalent.
- **Inverted Axes** – Switches the relative positions of the graph’s axes.
- **Show Dataset Label** – Applies the data set labels to the graph.
- **Display Legend** – Displays a color-coded legend for all displayed graphical components (not available on the Response vs. Time report).
- **Display Grid** – Displays a grid that is useful for visual comparisons (not available on the Pie graph).

## Displaying and Clearing Data Point Information

When working with graphs, you may want to display the value of a specific point in a graph.

To display information about a data point:

- Move the mouse over the desired area of the graph, and click CTRL-SHIFT-BUTTON1.

To clear data point information:

- Right-click the graph, and then click **Clear Point Information**.

## Changing a Graph's Type

When working with graphs, you can change the type of graph that TestManager displays.

To change a graph's type:

- In the graph that you want to change, click **View > Settings**, and then choose a graph type.

## Enlarging and Rotating a Graph

By clicking combinations of **SHIFT/CONTROL** keys and mouse buttons, you can further manipulate a graph's appearance. The following table lists some of the ways you can do this:

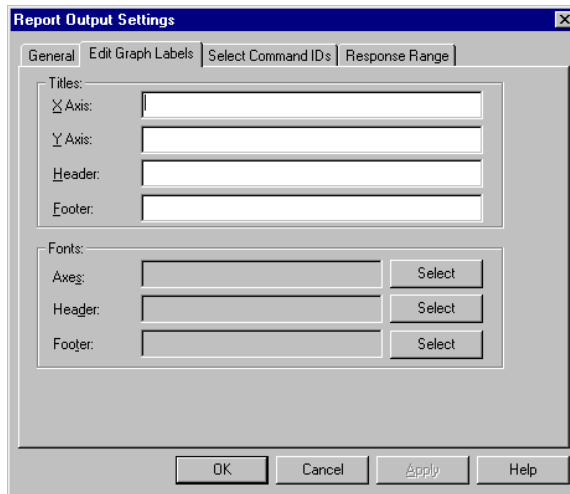
Action	Mouse/Key sequence	Other required action
Enlarge a graph's size.	CTRL-BUTTON1 BUTTON2	Drag the mouse toward the bottom of the graph.
Change a graph's position.	SHIFT-BUTTON1 BUTTON2	Move the mouse to reposition the graph.
Zoom in on a graph's axes.	SHIFT-BUTTON1	Draw a box around the area to zoom, and then release BUTTON1.
Zoom in on a graph's data.	CTRL-BUTTON1	Draw a box around the area to zoom, and then release BUTTON1.
Rotate the view of a graph (stack and pie graphs only).	BUTTON1 BUTTON2	Move the mouse up and down to change the inclination angle. Move the mouse left and right to rotate the graph (stack only).
Reset a graph to its original size.	The lowercase letter "r"	None.

## Changing a Graph's Labels

When working with a Command Status, Performance, or Compare Performance graph, you can change the labels of the graph, including text, font, style, and size of a label.

To change a graph's labels:

- In the graph that you want to change, click **View > Settings**.

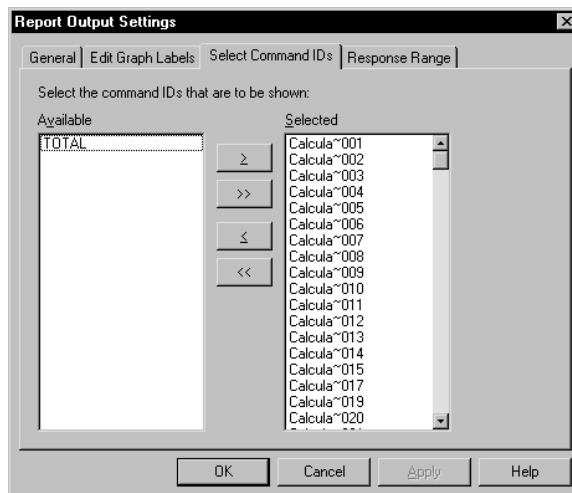


## Filtering Command IDs that Appear in a Graph

TestManager lets you filter command ID data before or after you process the report. Filtering command ID data after running the report is useful if your report results in a graph that is complex, and you want to examine portions of it in more detail.

To filter the command IDs in a graph:

- In the graph in which you want to filter IDs, click **View > Settings**.



## Editing the Properties of a Report

A report's properties are stored in the current project. These properties include:

- The name of the report.
- A description of the report.
- Who created the report.
- When the report was created.

Most of these properties are added automatically. However, you may want to add a description of the report definition, or the results you were trying to achieve with the report. To do this, edit the report properties.

To edit these properties:

- Choose a report and click **File > Properties**.

## Managing Reports

---

When working with reports, TestManager allows you to perform common management tasks with them. You can print, copy, rename, delete, and export reports as necessary for test administration.

### Printing a Report

TestManager allows you to print out the properties of a report and the processed results of any given report

To print a report:

- From an open report, click **File > Print**.

**Note:** When printing a processed report, you can add a header to the printout. For more information, see *Changing a Graph's Labels* on page 326.

### Copying a Report

TestManager allows you to copy reports to the Clipboard for use by other applications and within TestManager.

TestManager displays Compare Performance, Performance, Response Vs. Time, and Command Status reports in graphs and tables. You can copy the table portion onto the Clipboard, and then paste it into another application such as Microsoft Excel, Microsoft Word, or Microsoft Paint.



To copy a table into another application:

- From an open report, select the rows that you want to copy, and then click **Edit > Copy**.

Copying a report within TestManager is useful if, for example, you have defined a rather complex report and you want to modify one option. Although you can define another report from scratch, it is much easier to copy the report and then change that one option. In this way, you can be sure that you have modified only that option.

Copying graphical report data within TestManager is useful if you want to change the report settings. For example, you might want to change the format of a graph from bar to stack, or you might want to filter the report data. By copying the report and then changing the settings of the copy, you can preserve both the original and the changed report.

To copy a report:

- From the **Analysis** tab of the Test Asset Workspace, select the report to copy and click **Edit > Copy**.

**Note:** When you copy a report, you must specify a new name for the report.

## Renaming a Report

After you have defined a number of reports, you might want to rename one or several—for example, if you adopt a new naming convention for reports, or if you want to standardize naming conventions.

To rename a report:

- From the **Analysis** tab of the Test Asset Workspace, right-click the report to rename and select **Rename**.

## Deleting a Report

After you have defined and processed a number of reports, you may find that some of them are no longer useful. You can delete both reports that you have defined and the default reports that come with TestManager.

**Note:** Before you delete a report—default or otherwise—make sure that other people using the current Rational Test project do not use the report.

Deleting processed reports is useful if you run reports frequently and accumulate reports that you no longer need.

To delete a report:

- From the **Results** tab of the Test Asset Workspace, right-click the report and select **Delete**.

**Note:** To restore a default report that you have deleted, click **Tools > Options**, click the **Reports** tab, and then click the **Restore Defaults** button.

## Exporting Reports

The Performance, Command Status, Compare Performance, and Response vs. Time reports display data graphically. You can export this graphic data to a .csv file for further processing.

To export reports:

- Open the report and click **File > Export to File**.

## Changing Report Defaults

---

TestManager automatically generates Performance and Command Status reports at the end of a suite run. In addition, you can click a report name on the Report bar, and TestManager runs the report that you click.

You can specify the reports that TestManager generates at the end of a run. For example, TestManager can automatically display a Command Usage report in addition to the Performance and Command Status reports. Or TestManager can generate a Performance report based on a report that you have defined instead of the default Performance report.

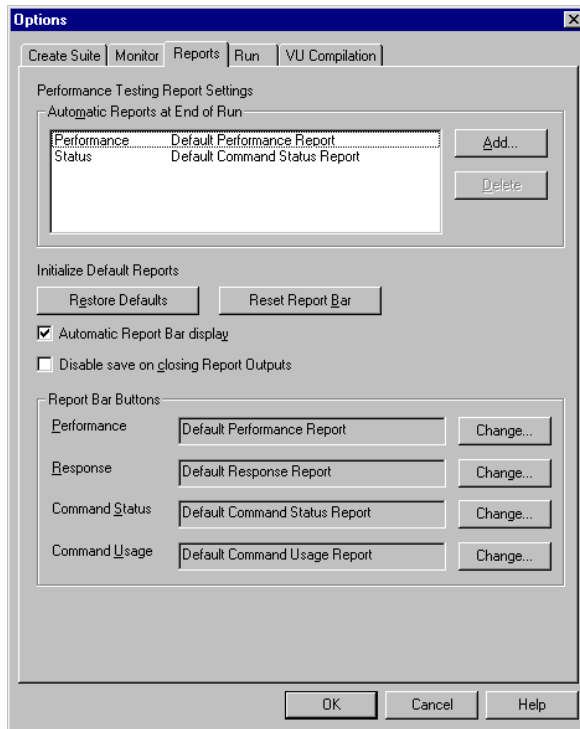
You also can change the reports that TestManager runs when you click a Report bar button. For example, instead of TestManager running the default Performance report, you can have it run a Performance report that you have defined.

## Changing the Reports that Run Automatically

TestManager automatically displays Performance and Command Status reports at the end of the suite run. However, you can change the reports that TestManager automatically displays.

To change the reports that TestManager automatically displays at the end of a suite run:

- Click **Tools > Options**, and then click the **Reports** tab.



## Changing the Reports that Run from the Report Bar

The Report bar lets you run reports by clicking a button. TestManager automatically runs the default reports unless you specify otherwise. For example, you may have defined a new report that you want to run from the Report bar instead of a default report.

To specify the reports that TestManager runs from the Report bar:

- Click **Tools > Options**, and then click the **Reports** tab.

**Note:** To reset the Report bar so that it generates the default reports, click **Tools > Options**, click the **Reports** tab, and then click the **Reset Report Bar** button.

## Types of Reports

---

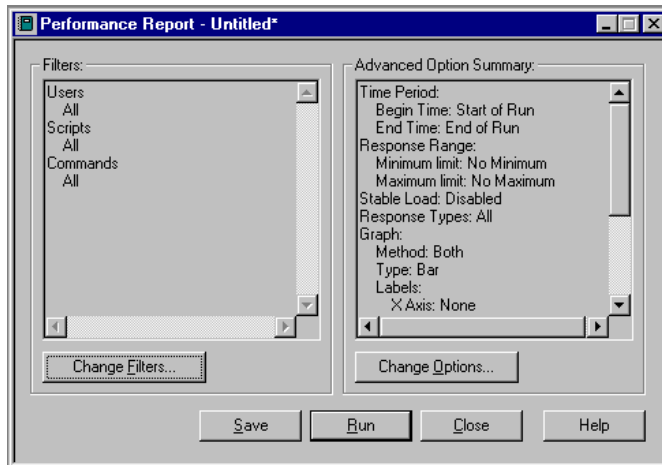
This section discusses the five kinds of performance reports available in TestManager.

## Performance Reports

You can use Performance reports to display the response times recorded during the suite run for selected commands. Performance reports also provide the mean, standard deviation, and percentiles for response times.

To define a new Performance report:

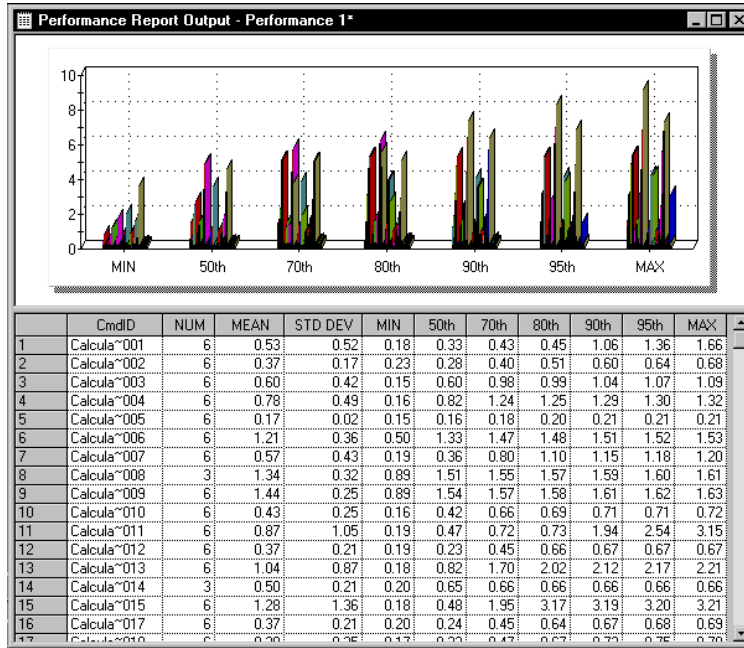
- Click **Reports > New > Performance**.



Performance reports are the foundation of reporting performance-related results in TestManager. They can show whether an application meets base criteria as defined in the test plan and/or the test case. For example, a Performance report can tell you whether 95% of virtual testers received responses from the test system in eight seconds or less—or what percentage of virtual testers did not receive responses from the system in that time.

Performance reports use the same input data as Response vs. Time reports, and they sort and filter data similarly. However, Performance reports group responses with the same command ID, while Response vs. Time reports show each command ID individually.

The following figure shows an example of a Performance report. This graph shows 15 bars for each percentile category, because 15 commands are graphed.



- The graph plots the seconds of response time against preset percentiles.
- The MIN category shows the minimum response time for each command ID.
- The 50th category shows the 50th percentile of time for each command ID.  
Half of the command IDs had a shorter response time and half had a longer response time.
- The MAX category shows the maximum response time for each command ID.

## What's in Performance Reports?

Performance reports contain the following information:

- **Cmd ID** – The command ID associated with the response.
- **NUM** – The number of responses for each command ID.
- **MEAN** – The arithmetic mean of the response times of all responses for each command ID.
- **STD DEV** – The standard deviation of the response times of all responses for each command ID.

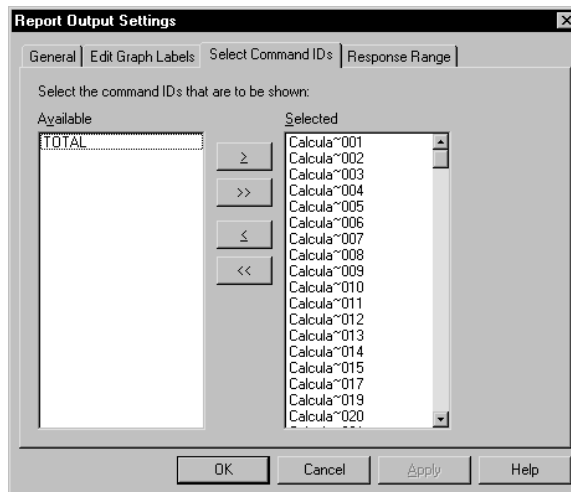
- **MIN** – The minimum response time of all responses for each command ID.
- **50th, 70th, 80th, 90th, 95th** – The percentiles of the response times of all responses for each command ID.

For example, if the 95th percentile of Add Ne002 is 0.53, then 95% of the responses are less than 0.53 seconds.

- **MAX** – The maximum response time of all responses for each command ID.

For example, to display the total response time in the graph and in the last line of the report:

- 1 From an processed Performance report, right-click on the graph and select **Settings**.



- 2 Click the **Select Commands IDs** tab, and select **TOTAL**.

## About Percentiles in Performance Reports

A percentile in a Performance report represents the longest amount of time it takes a defined percentage of the total number of virtual testers to complete a test script. Percentiles are given for each command ID in a test script and for the total time it took for all virtual testers to complete the entire list of commands.

For example, assume that you have a total of 100 virtual testers each executing a command once. The time in the 50th percentile column indicates that 50% of the virtual testers completed the command within that amount of time. It took some of those virtual testers two seconds to complete a command in a test script, some three seconds, and some five seconds. The 50th percentile time that would appear on the

Performance report would be three seconds, meaning that it took all of the virtual testers in the 50th percentile less than or equal to three seconds to complete the command.

The Performance report displays the minimum and maximum amounts of time it takes to complete one command. The percentile times range between the minimum and maximum times.

When you run a performance test, it is common for some virtual tester runs to take an abnormally long time to complete a command. For example, a server could be downloading an application while you are performing the test, so that the time it takes for the virtual testers to complete their tasks during this download will be longer than for the other virtual testers. This creates outliers—data that is extreme at one end of the scale and that does not accurately represent the trend of all the virtual testers. You can use percentiles to evaluate the results of the test in a way that eliminates outliers, and thus gives you a more accurate picture of the true response times.

For example, assume that you have a server that contains an internal web site that has support information used by 80 technical support staff. You do the following:

- Decide that the time it takes to gain access to the Web site should never exceed 8 seconds.
- Create a suite with a test script that gains accesses to the internal support web site.
- Consider that the anticipated load is 80 users, and decide to test for 100 users to eliminate outliers.
- Define a user group of 100 virtual testers, run the suite, and then run a Performance report.

The percentiles shown on the report are 50, 60, 70, 80, 90, 95. Although you ran the suite with 100 virtual testers, because you are only required to have results for 80 users, you can discount the 90th and 95th percentiles to eliminate outlier data and to get accurate results for 80 users. Along with eliminating outliers, testing for more virtual testers enables you to determine whether the server can handle more than the anticipated load

## Compare Performance Reports

The Compare Performance report compares response times measured by Performance reports. After you have generated several Performance reports, you can use a Compare Performance report to compare the values of a specific field in each of those reports. You also can compare reports that show different numbers of virtual testers or compare reports from runs on different system configurations.

Compare Performance reports allow you to see how benchmark information on a particular application differs for various load configurations. This can help you identify, for example, needed protocol improvements in the tested application. For example, you could run a test on an application with various virtual tester loads, and then compare the Performance reports of the various test runs to see how the application manages under an ever-increasing virtual tester load.

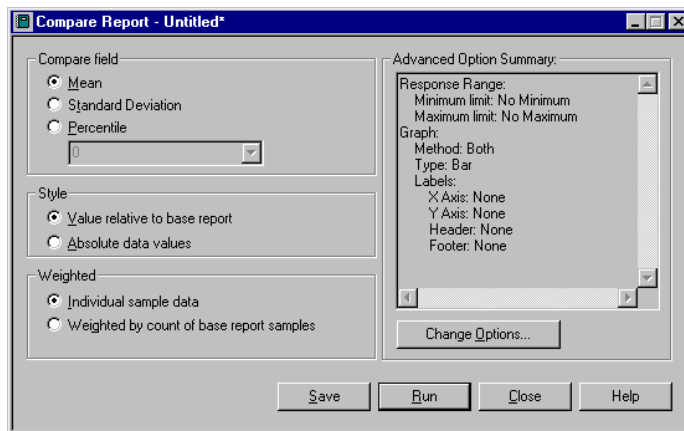
When you run a Compare Performance report, you specify the base Performance report and up to six other Performance reports.

## Defining a Compare Performance Report

Defining a Compare Performance report is similar to defining other reports.

To define a new Compare Performance report:

- Click **Reports > New > Compare Performance**.



When you define a Compare Performance report, you must define the following:

- The fields to compare in the selected Performance reports:
  - **Mean** – Compares the mean value of the response times.
  - **Standard Deviation** – Compares the standard deviation for the response times.
  - **Percentile** – Compares the response times based on the percentile that you select. The percentile must be in the Performance report. For example, if the Performance reports calculate the 50, 70, 80, 90, and 95 percentiles, the Compare Performance report must use one of these percentiles.
- The style of the comparison relative to the base Performance report:



- **Value relative to base report** – Compares the response times relative to the base Performance report. With this option, the first column in the report (the base Performance report) is always 1, and the other columns are relative to it. For example, if the base report lists a response time as 2.5, and another report lists the response time as 5, then the Compare Performance report lists them as 1 and 2.
- **Absolute data values** – The indicated response times appear in the report. For example, if the base report lists a response time as 2.5, and another report lists the response time as 5, then the Compare Performance report lists them as 2.5 and 5.
- The weight of the response times that occur most frequently:
  - **Individual sample data** – The response times are not weighted. A command ID that occurs ten times and a command ID that occurs 100 times have an equal influence on the response time statistics.
  - **Weighted by count of base report samples** – The response times are weighted to reflect the frequency of occurrence of the command ID to which they correspond. Command IDs that occur more frequently have more influence on the response time statistics, and command IDs that occur less frequently have less influence on the statistics.

**Note:** For more information about advanced options, see the TestManager Help.

## What's in Compare Performance Reports?

A Compare Performance report can compare reports in a number of ways. It can compare reports absolutely, or it can compare reports relative to a base report. In addition, the response times can be weighted so that command IDs that occur frequently have more influence, or they can be unweighted so that each command ID has equal influence.

There are four versions of the Compare Performance report:

- Absolute
- Weighted absolute
- Relative
- Weighted relative

Examples of each version use the same Performance reports as input.

## Absolute Compare Performance Reports

Absolute reports display the actual values of the response times, in seconds. The final line of the report gives the arithmetic sum of the response times.

To define an absolute report:

- Follow the steps in *Defining a Compare Performance Report* on page 336, and be sure to choose the **Absolute data values** option.

The following figure shows the last few lines of an absolute Compare Performance report:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	0.01	0.04	0.01	0.01
150	READ R018	0.01	0.02	0.01	0.01
151	READ R019	0.00	0.01	0.00	0.00
152	READ R020	0.00	0.01	0.00	0.01
153	READ R021	0.00	0.01	0.00	0.00
154	READ R022	0.00	0.00	0.00	0.00
155	SUM	2.73	4.79	5.11	12.05

## Weighted Absolute Compare Performance Reports

Weighted absolute reports weigh response times by their frequency of occurrence and are useful for comparing total response times.

To define a weighted absolute report:

- Follow the steps in *Defining a Compare Performance Report* on page 336, and choose the **Absolute data values** and the **Weighted by count of base report samples** options.

The weight applied is equal to the number of valid responses for that command ID in each report. If the command IDs in the reports have a different number of responses, TestManager uses the smallest non-zero number as the weight.

The weighted absolute value is the product of this weight and the absolute value for the response time. The final line of the weighted absolute Compare Performance report gives the arithmetic sum of the weighted response times for each report.

The following figure shows the last few lines of a weighted absolute Compare Performance report:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	0.03	0.12	0.03	0.03
150	READ R018	0.03	0.06	0.03	0.03
151	READ R019	0.00	0.03	0.00	0.00
152	READ R020	0.00	0.03	0.00	0.03
153	READ R021	0.00	0.03	0.00	0.00
154	READ R022	0.00	0.00	0.00	0.00
155	WEIGHTED SUM	9.11	14.61	17.45	41.53

### Relative Compare Performance Reports

Relative reports list the base response time as 1.00 and the other response times relative to that base.

To define a relative report:

- Follow the steps in *Defining a Compare Performance Report* on page 336, and choose the **Value relative to base report** option.

The final line of the report gives the geometric mean of the relative response times for each report. To determine the geometric mean, TestManager multiplies the response times, and then takes a root of the product that is equal to the number of response times.

For example, if there are five response times, TestManager multiplies them together and takes the fifth root of the product.

Mathematically, the geometric mean of a set of values  $x_1, x_2, \dots, x_k$  is expressed as:

$$(x_1 x_2 \dots x_k)^{1/k}$$

The following figure shows the last few lines of a relative Compare Performance report:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	1.00	4.00	1.00	1.00
150	READ R018	1.00	2.00	1.00	1.00
151	READ R019	1.00	10.00	1.00	1.00
152	READ R020	1.00	10.00	1.00	10.00
153	READ R021	1.00	10.00	1.00	1.00
154	READ R022	1.00	1.00	1.00	1.00
155	GEO MEAN	1.00	1.51	1.07	1.44

### Weighted Relative Compare Performance Reports

This report is the same as the relative report, except that it also lists the weighted geometric mean.

To define a weighted relative report:

- Follow the steps in *Defining a Compare Performance Report* on page 336, and choose the **Value relative to base report** and the **Weighted by count of base report samples** options.

The weighted geometric mean differs from the geometric mean in that it takes into account the frequency with which the different command IDs occur. Frequently used command IDs have a greater influence on the weighted geometric mean than infrequently used ones—in contrast to the geometric mean, where all command IDs have equal influence.

The weight applied when calculating the weighted geometric mean for each command ID equals the number of valid responses for that ID in each report being compared. If the number of valid responses for a command ID differs among the reports, the smallest non-zero count is used as its weight.

Mathematically, the weighted geometric mean of a set of values  $x_1, x_2, \dots, x_k$  with frequencies (weights) of  $f_1, f_2, \dots, f_k$ , where  $f_1 + f_2 + \dots + f_k = N$ , is expressed as:

$$(x_1^{f_1} x_2^{f_2} \dots x_k^{f_k})^{1/N}$$

The following figure shows the last few lines of a weighted relative Compare Performance report:

	CmdID	Build 1	Build 1	Build 1	Build 1
		sample schedule	sample schedule	sample schedule	sample schedule
		Users 5 #02	Users 10 #01	Users 15 #03	Users 20 #09
		Performance 1	Performance 1	Performance 1	Performance 1
149	READ R017	1.00	4.00	1.00	1.00
150	READ R018	1.00	2.00	1.00	1.00
151	READ R019	1.00	10.00	1.00	1.00
152	READ R020	1.00	10.00	1.00	10.00
153	READ R021	1.00	10.00	1.00	1.00
154	READ R022	1.00	1.00	1.00	1.00
155	GEO MEAN	1.00	1.51	1.07	1.44
156	WGHT GMEA	1.00	1.30	1.08	1.47

### N/A and Undefined Responses

Occasionally, you might see the strings `n/a` and `undefn` in a Compare Performance report. The following table describes when TestManager displays these strings:

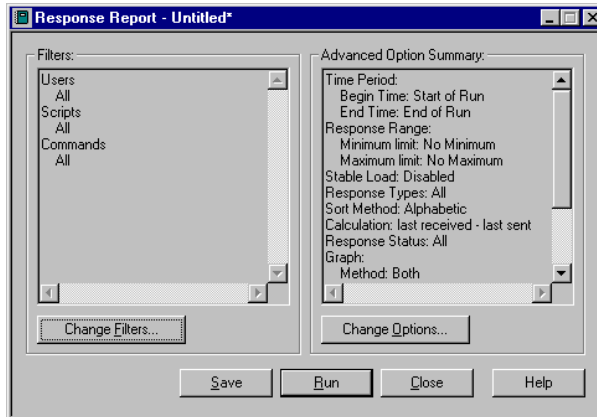
If ...	Then the Compare Performance report ...
A command ID is in the base report but does not exist in the other reports.	Lists <b>n/a</b> for that command ID in the table and does not include information for that command ID in the report graph.
A command ID is in the report but does not occur in the base report.	Ignores that command ID.
You are producing a relative report, and some command IDs have a response time of 0.	Lists the response time as <b>0</b> in the base report, and lists the other results corresponding to that command ID as <b>Undefn</b> .
All the response times for a report are listed as <b>n/a</b> or <b>Undefn</b> .	Lists the geometric mean or sum as <b>n/a</b> .

### Response vs. Time Reports

Response vs. Time reports display individual response times. Response vs. Time reports use the same input data as Performance reports, and sort and filter data similarly. However, Response vs. Time reports show each command ID individually, while Performance reports group responses with the same command ID.

To define a new Response vs. Time report:

- Click **Reports > New > Response vs. Time.**



Response vs. Time reports are useful for the following tasks:

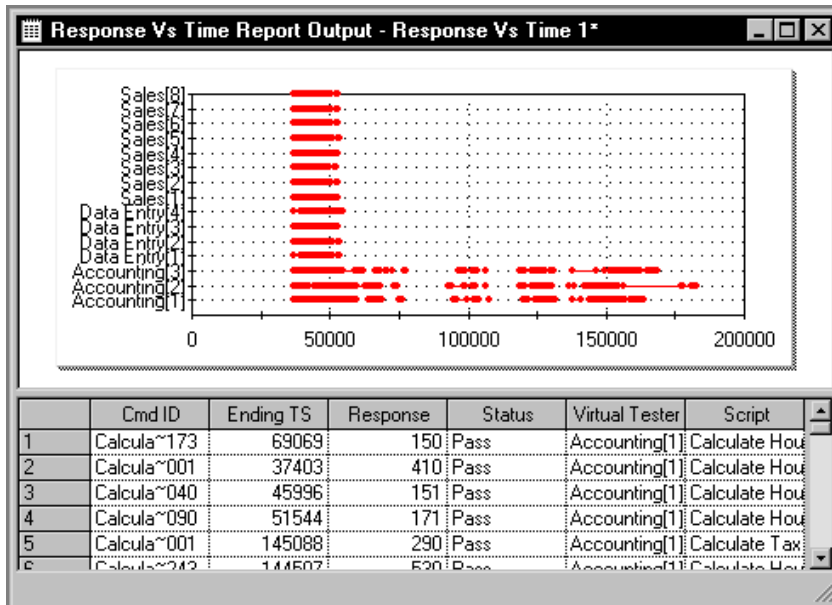
- Checking the trend in the response time. The Response vs. Time report shows the response time versus the elapsed time of the suite run.

The response time should be clustered around one point rather than getting progressively longer or shorter. If the trend changes, check that you have excluded login and setup time in your results. The worst case is that you might need to change your test design.

- Checking any spikes in the response time. If the response time is relatively flat except for one or two spikes, you might want to investigate the cause of the spikes.
- Filtering the data so that it contains only one command ID, and then graphing that command ID as a histogram.
- Checking the resources used by a computer in the run (optional).

To see the resources used, right-click on the Response vs. Time report and select a computer.

The following figure shows a Response vs. Time report. This graph shows that the first virtual tester in the accounting group (Accounting 1) executed two commands.



The graph plots each virtual tester versus the response time in milliseconds. The graph contains many short lines that resemble dots. They indicate that the response times for all the virtual testers are quite short. The longer a line is on the X axis, the longer the response time, because the X axis graphs the response time.

## What's in Response vs. Time Reports?

Typically, Response vs. Time reports contain two sections, one for expected responses and one for unexpected responses. The responses within each section are sorted by command ID. Within each command ID, responses are sorted by the ending timestamp.

Response vs. Time reports contain the following information:

- **Cmd ID** – The command ID associated with the response.
- **Ending TS** – The ending timestamp of the response. This timestamp corresponds to the value of the read-only timestamp variable for the response. The timestamp is the interval ending timestamp as defined by the Time Period report option.
- **Response** – The response time in milliseconds.
- **Status** – Displays **P** or **F** to indicate whether the response passed or failed.
- **Virtual Tester** – The virtual tester corresponding to the response.
- **Script** – The name of the test script corresponding to the response.

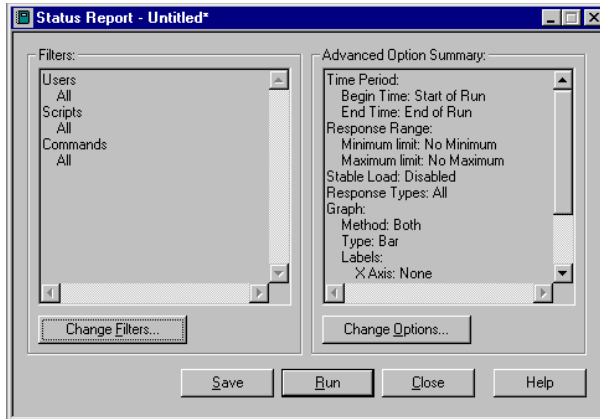
## Command Status Reports

Command Status reports show how well actual responses correspond with the expected responses. If the response that you received is the same or is expected, TestManager considers that it has passed; otherwise, TestManager considers it failed.



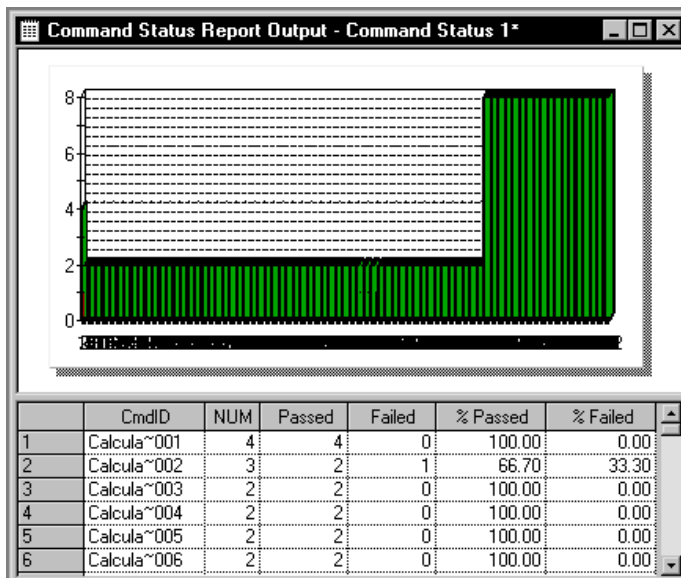
To define a new Command Status report:

- Click **Reports > New > Command Status**.



Command Status reports reflect the overall health of a suite run. They are similar to Performance reports, but they focus on the quantity of commands run in the suite. Command Status reports are excellent tools for debugging the testing process, as you can see easily which commands fail repeatedly and that address that test script accordingly.

The following figure shows an example of Command Status report. This graph shows that command 1 (command ID Add Ne01) ran 24 times and did not fail, and command 6 (command ID Calcul001) ran 8 times and did not fail.



The graph plots the command number against the number of times the test script ran. It displays commands that passed in green, and displays commands that failed in red.

## What's in Command Status Reports?

Command Status reports contain the following information:

- **Cmd ID** – The command ID associated with the response.
- **NUM** – The number of responses corresponding to each command ID. This number is the sum of the numbers in the **Passed** and **Failed** columns.
- **Passed** – The number of passed responses for each command ID (that is, those that did not time out).
- **Failed** – The number of failed responses for each command ID that timed out (that is, the expected response was not received).
- **% Passed** – The percentage of responses that passed for that command ID.
- **% Failed** – The percentage of responses that failed for that command ID.

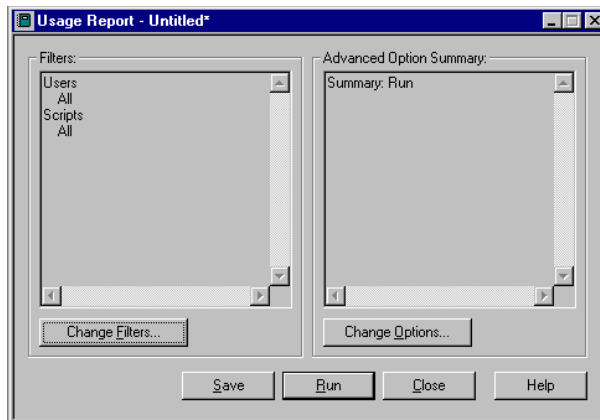
The last line of the report lists the totals for each column.

## Command Usage Reports

Command Usage reports display data on all emulation commands and responses. The report describes throughput and virtual tester characteristics during the suite run.

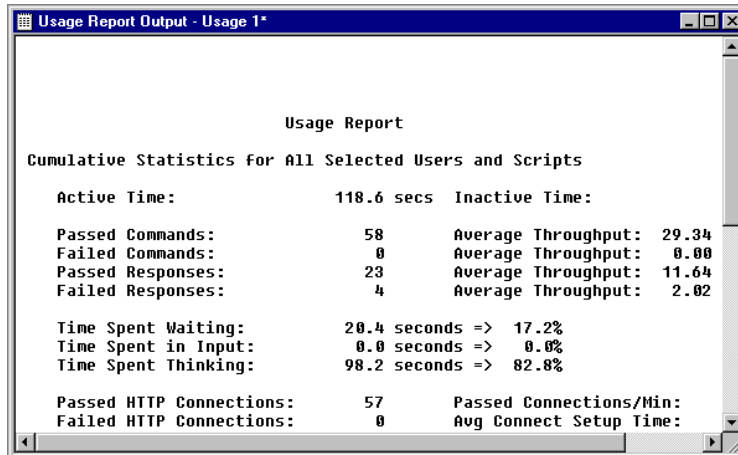
To define a new Command Usage report:

- Click **Reports > New > Command Usage**.



The summary information in the Command Usage report gives a high-level view of the division of activity in a test run. The cumulative time spent by virtual testers executing commands, thinking, or waiting for a response can tell you quickly where there are bottlenecks in the test application. The Command Usage report also can provide summary information for protocols.

The following figure shows an example of a Command Usage report:



## What's in Command Usage Reports?

Command Usage reports contain a section on cumulative statistics and a section on summary statistics.

### Cumulative Statistics

- **Active Time** – The sum of the active time of all virtual testers. The active time of a virtual tester is the time that the virtual tester spent thinking (including delays after the virtual tester's first recorded command), executing commands, and waiting for responses.
- **Inactive Time** – The sum of the inactive time of all virtual testers and test scripts. The inactive time of a virtual tester is the time before the virtual tester's first emulation command (including overhead time needed to set up and initialize the run), and possibly inter-script delay (the time between the last emulation command of the previous test script and the beginning of the current test script).
- **Passed Commands** – The total number of passed `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, `http_request`, `sock_send`, `emulate`, and `DCOM` method call commands executed.

- **Failed Commands** – The total number of failed `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, `http_request`, `sock_send`, `emulate`, and `DCOM` method call commands executed.
- **Passed Responses** – The total number of responses to input commands that were matched by passing receive commands (`sqlnrecv`, `sqllongrecv`, `http_nrecv`, `http_recv`, `sock_nrecv`, and `sock_recv`). This is not the same as the total number of expected receive commands, since a response may be matched by an arbitrary number of receive commands. A response is considered expected if all receive commands used to match it have an expected status.
- **Failed Responses** – The total number of responses that were matched by failing receive emulation commands. This is not the same as the total number of unexpected receive commands, since a response may be received by an arbitrary number of receive commands. A response is considered unexpected if any receive commands used to match it have an unexpected status.
- **Average Throughput** – Four measurements of average throughput are provided: passed command throughput, failed command throughput, passed response throughput, and failed response throughput. This represents the throughput of an average virtual tester.
- **Time Spent Waiting** – The total time spent waiting for responses, given both in seconds and as a percentage of active time. The time spent waiting is the elapsed time from when the input command is submitted to the server until the server receives the complete response. The time that an `http_request` spends waiting for a connection to be established is counted as time spent waiting.
- **Time Executing Commands** – The total time spent in executing `sqlexec`, `sqlprepare`, `sql*_cursor`, `TUXEDO`, `emulate`, and `DCOM` method call commands. This measurement is provided both in seconds and as a percentage of active time. The time spent executing SQL commands is defined as the elapsed time from when the SQL statements are submitted to the server until these statements have completed. The time spent executing `TUXEDO` commands is defined as the time to execute the specific `ATMI` primitive until it succeeds or fails.
- **Time Spent in Input** – The total time spent sending virtual tester input to the server. This measurement is provided both in seconds and as a percentage of active time. The time spent by `http_request` and `sock_send` commands in sending input to the server is reported as time spent in input.
- **Time Spent Thinking** – The total time spent thinking, both in seconds and as a percentage of active time. The time spent thinking for a given command is the elapsed time from the end of the preceding emulation command until the current emulation command is submitted to the server. This definition of think time

corresponds to that used during the run only if the environment variable `Think_def` in the test script has the default LR (last received), which assumes that think time starts at the last received data timestamp of the previous response.

If any SQL emulation commands were executed, the Command Usage report includes:

- **Rows Received** – The number of rows received by all reported `sqlnrecv` commands.
- **Received Rows/Sec** – Average number of rows received per second. Derived by dividing the number of rows received by the active time
- **Average Rows/Response** – Average number of rows in the passed and failed responses. Derived by dividing the number of rows received by the number of passed and failed responses.
- **Average Think Time** – Average think time in seconds for `sqlexec` and `sqlprepare` statements only.
- **SQL Execution Commands** – The number of `sqlexec` commands reported.
- **Preparation Commands** – The number of `sqlprepare` commands reported.
- **Rows Processed** – The number of rows processed by all reported `sqlexec` commands.
- **Processed Rows/Sec** – Average number of rows processed per second. Derived by dividing the number of rows processed by the active time.
- **Avg Rows/Execute Cmd** – Average number of rows processed by each `sqlexec` command. Derived by dividing the number of rows processed by the number of `sqlexec` commands reported.
- **Avg Row Process Time** – Average time in milliseconds for processing a row by an `sqlexec` command. Derived by dividing the time spent on `sqlexec` commands by the number of rows processed.
- **Avg Execution Time** – Average time in milliseconds to execute an `sqlexec` or DCOM method call command. Derived by dividing the time spent on `sqlexec` commands by the number of `sqlexec` commands.
- **Avg Preparation Time** – Average time in milliseconds to execute an `sqlprepare` command. Derived by dividing the time spent on `sqlprepare` commands by the number of `sqlprepare` commands.

If any HTTP emulation commands were executed, the Command Usage report includes:

- **Passed HTTP Connections** – The number of successful HTTP connections established by all reported `http_request` commands.
- **Failed HTTP Connections** – The number of HTTP connection attempts that failed to establish a connection for all reported `http_request` commands.
- **HTTP Sent Kbytes** – Kilobytes of data sent by reported `http_request` commands.
- **HTTP Received Kbytes** – Kilobytes of data received by reported `http_nrecv` and `http_rcv` commands.
- **Sent Kbytes/Connection** – Kilobytes of data sent by reported `http_request` commands per connection. Derived by dividing the kilobytes of data sent by the number of successfully established HTTP connections.
- **Passed Connections/Min** – The number of successful HTTP connections established per minute. Derived by dividing the number of successful HTTP connections by the active time.
- **Avg Connect Setup Time** – Average time, in milliseconds, required to establish a successful HTTP connection. Derived by dividing the total connection time for all recorded `http_request` commands by the number of successful connections.
- **HTTP Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `http_request` commands by the active time.
- **HTTP Rcv Kbytes/Sec** – Kilobytes of data received per second. Derived by dividing the kilobytes of data received by all recorded `http_nrecv` and `http_rcv` commands by the active time.
- **Rcv Kbytes/Connection** – Kilobytes of data received by reported `http_nrecv` and `http_rcv` commands per connection. Derived by dividing the kilobytes of data received by the number of successfully established HTTP connections.

If any socket emulation commands were executed, the Command Usage report includes:

- **Passed Socket Connections** – The number of successful socket connections established by all reported `sock_connect` functions.
- **Socket Sent Kbytes** – Kilobytes of data sent by reported `sock_send` commands.
- **Socket Received Kbytes** – Kilobytes of data received by reported `sock_nrecv` and `sock_rcv` commands.
- **Passed Connections/Min** – The number of successful socket connections established per minute. Derived by dividing the number of successful socket connections by the active time.

- **Socket Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `sock_send` commands by the active time.
- **Socket Recv Kbytes/Sec** – Kilobytes of data received per second. Derived by dividing the kilobytes of data received by all recorded `sock_nrecv` and `sock_recv` commands by the active time.

If any TUXEDO emulation commands were executed, the Command Usage report includes:

- **Tuxedo Execution Commands** – The number of TUXEDO commands reported.
- **Avg Execution Time** – Average time in milliseconds to execute a TUXEDO command. Derived by dividing the time spent on TUXEDO commands by the number of TUXEDO commands.

If any `start_time` emulation commands were executed, the Command Usage report includes:

- **stop\_time Commands** – The number of `stop_time` commands reported.
- **stop\_time Cmds/Min** – The number of `stop_time` commands per minute. Derived by dividing the number of `stop_time` commands by the active time.
- **start\_time Commands** – The number of `start_time` commands reported.
- **Avg Block Time** – Average response time in seconds for reported `stop_time` commands. Derived by dividing the sum of the response times for all `stop_time` commands by the number of `stop_time` commands. The response time of a `stop_time` command is the elapsed time between it and its associated `start_time` command.

If any `emulate` emulation commands were executed, the Command Usage report includes:

- **Passed emulate Commands** – The number of `emulate` commands that report a passed status.
- **Passed emulate Time Spent** – The total time spent, from when the passed `emulate` commands start to where they end.
- **Failed emulate Commands** – The number of `emulate` commands that report a failed status.
- **Failed emulate Time Spent** – The total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Command Usage report includes:

- **Passed testcase Commands** – The number of `testcase` commands that report a passed status.
- **Failed testcase Commands** – The number of `testcase` commands that report a failed status.

### Summary Statistics

- **Duration of Run** – Elapsed time from the beginning to the end of the run. The beginning of the run is the time of the first emulation activity among all virtual testers and test scripts, not just the ones you have filtered for this report. Similarly, the end of the run is the time of the last emulation activity among all virtual testers and test scripts.
- **Passed Commands, Failed Commands, Passed Responses, Failed Responses** – Identical to their counterparts in *Cumulative Statistics* on page 347.
- **Total Throughput** – Four measurements of total throughput are provided: passed command throughput, failed command throughput, passed response throughput, and failed response throughput. The total throughput of passed commands is obtained by dividing the number of passed commands by the run's duration, with the appropriate conversion of seconds into minutes. Thus, it represents the total passed command throughput by all selected virtual testers at the applied workload, as opposed to the throughput of the average virtual tester. The total failed command, and the total passed and failed response throughputs are calculated analogously.

These throughput measurements, as well as the test script throughput, depend upon the virtual tester and test script selections. For example, if only three virtual testers from a ten-virtual tester run are selected, the throughput would not represent the server throughput at a ten-virtual tester workload, but rather the throughput of three selected virtual testers as part of a ten-virtual tester workload. As a guideline, the summary throughput measurements are most meaningful when all virtual testers and test scripts are selected.

- **Number of Users** – The number of virtual testers in the suite run.
- **Number of stop\_time Cmds** – The number of `stop_time` commands in the suite run.
- **Number of Completed Scripts** – Test scripts are considered complete if all activities associated with the test script are completed before the run ends.
- **Number of Uncompleted Scripts** – The number of test scripts that have not finished executing when a run is halted. Test scripts can be incomplete if you halt the run or set the suite to terminate after a certain number of virtual testers or test scripts.



- **Average Number of Scripts Completed per User** – Calculated by dividing the number of completed test scripts by the number of virtual testers.
- **Average Script Duration for Completed Scripts** – Average elapsed time of a completed test script. Calculated by dividing the cumulative active time of all virtual testers and test scripts by the number of completed test scripts.
- **Script Throughput for Completed Scripts** – The number of test scripts-per-hour completed by the server during the run. Calculated by dividing the number of completed test scripts by the duration of the run, with the conversion of seconds into hours. This value changes if you have filtered virtual testers and test scripts.

If any `stop_time` emulation commands were executed, the Command Usage report includes:

- **Avg Number of stop\_time Commands** – Calculated by dividing the number of `stop_time` commands by the number of virtual testers.
- **Average start\_time/stop\_time Duration** – Average response time in seconds for reported `stop_time` commands. Derived by dividing the sum of the response times for all `stop_time` commands by the number of `stop_time` commands. The response time of a `stop_time` command is the elapsed time between it and its associated `start_time` command.
- **stop\_time Command Throughput for all Users** – The number of `stop_time` commands executed per minute during the suite run. Derived by dividing the number of `stop_time` commands by the duration of the run.

If any `emulate` emulation commands were executed, the Command Usage report includes:

- **Passed emulate Commands** – The number of `emulate` commands that report a passed status.
- **Passed emulate Time Spent** – The total time spent, from when the passed `emulate` commands start to where they end.
- **Failed emulate Commands** – The number of `emulate` commands that report a failed status.
- **Failed emulate Time Spent** – The total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Command Usage report includes:

- **Passed testcase Commands** – The number of `testcase` commands that report a passed status.

- **Failed testcase Commands** – The number of `testcase` commands that report a failed status.

# Configuring Local and Agent Computers

# A

If your suite runs a large number of virtual testers, you must set certain system environment variables for the run to complete successfully.

## Running More Than 245 Virtual Testers

---

If your suite runs more than 245 virtual testers total, you must change two settings in the NuTCRACKER operating environment on the Local computer. To run more than 245 virtual testers on an NT Agent computer, you must make the same changes on that Agent.

To change these settings:

- 1 Click **Start > Settings > Control Panel > Nutcracker**.
- 2 Click the **NuTC 4 Options** tab.
- 3 Select **Semaphore Settings** from the Category list.
- 4 Change the **Max Number of Semaphores** to  $N + S + 10$ , where  $N$  is the number of virtual testers you want to run and  $S$  is the number of shared variables used by scripts in the suite.
- 5 Repeat for **Max Number of Semaphores Per ID**.
- 6 Click **OK**.
- 7 Click **Restart Later**.
- 8 Restart NT.

## Running More Than 1000 Virtual Testers

---

If your suite runs more than 1000 virtual testers total, you must create an environment variable that sets the minimum shared memory size on the Local computer. To run more than 1000 virtual testers on an NT Agent computer, you must make the same changes on that Agent.

To create and set this environment variable:

- 1 Click **Start > Settings > Control Panel > System**.
- 2 Click the **Environment** tab.
- 3 Create an environment variable named `RT_MASTER_SHM_MINSZ`, and set its value to  $700 * N$ , where  $N$  is the number of virtual testers that you want to run.  
  
On the Local computer,  $N$  is the total number of virtual testers for the entire run.  
On the Agent computer,  $N$  is the number of virtual testers that run on that Agent.
- 4 Click **Set**, and then click **OK**.
- 5 Restart NT.

## **Running More Than 1000 Virtual Testers on One NT Computer**

---

If your suite runs more than 1000 virtual testers on an NT computer, you must create and set a system environment variable on each NT computer running more than 1000 virtual testers.

To create and set this environment variable:

- 1 Click **Start > Settings > Control Panel > System**.
- 2 Click the **Environment** tab.
- 3 Create an environment variable named `RT_MASTER_NTUSERLIMIT`, and set its value to the number of virtual testers you want to run.
- 4 Click **Set**, and then click **OK**.
- 5 Restart TestManger (on the Local computer) or the test Agent (on the Agent computer) for the new setting to take effect on that computer.

## Running More Than 24 Virtual Testers on a UNIX Agent

---

If your suite runs more than 24 virtual testers on a UNIX Agent computer, you must set the following system environment variables:

System Environment Variable	Value
Total TestManager processes (NPROC, MAXUP)	The number of virtual testers on the Agent + 5.
Total open files (NFILE, NINODE)	$(6 * N) + (open\_files * N) + (connections * N)$ <i>N</i> is the number of virtual testers on the Agent. <i>open_files</i> is the number of files explicitly opened within test scripts. <i>connections</i> is the number of connections open concurrently.
Total system-wide shared memory (SHMALL/SHMMAX)	$724 + 5609N + 16S + 13G + group\_names$ bytes <i>N</i> is the number of virtual testers on the Agent. <i>S</i> is the total number of shared variables in all the test scripts in the suite. <i>G</i> is the total number of user groups in the suite. <i>group_names</i> is the total length of all user group names in the suite.
Semaphore set IDs (SEMMNI, SEMMAP)	1
Total semaphores (SEMMNS)	The number of virtual testers on the Agent.
Semaphores per set (SEMMSL)	The number of virtual testers on the Agent.

**Note:** These values are in addition to the requirements of other system processes or applications. The current system values should *not* be decreased. For example, if other system processes require SEMMNI=10, then do not decrease the value to 1.

For example, for a Solaris Agent running 2000-4000 virtual testers, set system environment variables as follows:

```
set semsys:seminfo_semmmap=1024
set semsys:seminfo_semmni=4096
set semsys:seminfo_semmns=4096
set semsys:seminfo_semmnu=4096
set semsys:seminfo_semmsl=1024
set semsys:seminfo_semopm=50
set semsys:seminfo_semume=64
set semsys:seminfo_shmmni=1024
```

```
set semsys:seminfo_shmmax=100072000
set semsys:seminfo_shmseg=100
set semsys:seminfo_shmmin=1
```

## Controlling TCP Port Numbers

---

The `rtmstr_v` and `rtmstr_s` network services have been added to control the ports on the Local computer to which the Agent communication software connects. These network services allow tests to be run with Local and Agent computers on different networks separated by a firewall, by controlling the ports to which the listening Local server processes bind.

In a test run involving Agents, there are multiple socket connections between the Local and each Agent.

Connections made from the Local to the Agent computer are always made to a single well-known port on which the test Agent is listening. This port defaults to 8800.

There are two connections made from each Agent to the Local, one to a Local server process named `rtvsrv` and another to a Local server process named `rtssrv`. These two server processes each listen on a separate port. They do not bind to a specific port, but instead the Local computer's operating system chooses a port dynamically. The Local computer then communicates these port values to the Agent during run initialization. (Note that all Agents connect to the same two ports on the Local computer.) It is these two dynamically chosen ports on the Local computer that cause firewall administration problems because the two ports that will be used cannot be determined in advance.

You can control this problem by using the optional presence of network services (the traditional TCP/UDP network services defined in an `/etc/services` file, not to be confused with an NT service). On NT, the services file is found in *Drive*\WINNT\system32\drivers\etc\services. There is one entry per line, which lists the service name, the port number, and the protocol (TCP or UDP).

Specifically, control over the ports is provided as follows:

`rtvsrv` binds to the port (in priority order):

- 1 The value of the TCP service named `rtmstr_v`, if defined.

Otherwise,

- 2 A port dynamically chosen by the system.

rtssrv binds to the port (in priority order):

- 1 The value of the TCP service named `rtmstr_s`, if defined.

Otherwise,

- 2 A port dynamically chosen by the system.

Note that the ports defined by these two services are independent. That is, they do not need to be adjacent, nor related to the well-known test Agent port of 8800. They do need to be unique. We suggest using the ports 8801 and 8802 if they are not used for some other service on the Local computer.

For example, if you want the ports on the Local computer to be 8801 and 8802, add two lines to the services file:

```
rtmstr_s8801/tcp# TestStudio Master S server
rtmstr_v8802/tcp# TestStudio Master V server
```

In addition, the `rtagent` network service has been added to control the port at which the test Agent listens. If the well-known Agent port of 8800 is already in use by another application on one or more Agent computers, an alternate port needs to be specified using the `rtagent` service.

The `rtagent` service is put in the services file in the same way that the network services `rtmstr_v` and `rtmstr_s` are put in the file. The difference is that the `rtagent` service must be defined on the Local and all Agents used in the testing run, and must be identical for all systems. The Agents must be rebooted after altering the service file.

For example, if you want the Agent to list on port 8888, add a line to the services file on both the Local and the Agent:

```
rtagent8888/tcp# TestStudio Agent
```

## Setting Up IP Aliasing

---

TestManager provides IP aliasing, which allows many IP addresses to be assigned to the same physical system. Every virtual tester can be assigned a different IP address to realistically emulate your virtual tester community. The requests generated by these virtual testers receive responses back from the Web server with timing characteristics and validation recorded intact.

To use IP aliasing on any particular computer, the system administrator must set up the IP addresses on that system.

For Windows NT, this can be done with the **Settings > Control Panel > Network > Protocols > TCP/IP Protocol > Properties > Advanced > IP Addresses > Add** button.

For UNIX, this can be done with the `ifconfig` (1) command line utility. See the `ifconfig` manual pages for specific details appropriate to that operating system. To set up large numbers of IP addresses, it is convenient to use a Perl or UNIX shell script. A sample Korn shell script for this purpose named `ipalias_setup` can be found in the `bin` directory of UNIX Agent installs. (You must have root privileges to set up IP aliases with `ifconfig`.)

Be careful when assigning IP addresses to a computer, since you may run into problems such as conflicting IP addresses or routing considerations. We recommend that IP addresses be assigned by a qualified network administrator.

After IP Aliasing is set up, open a suite, click the **Suite > Edit Runtime**, and select the **Enable IP Aliasing** check box.

If IP Aliasing is selected in the suite, then at the beginning of a run, the TestManager software on each computer (Local or Agent) queries the system for all available IP addresses. Each suite scheduled to run on that computer is assigned an IP address from that list, in round-robin fashion. In other words, if there are more virtual testers on a computer than IP addresses, then an IP address is assigned to multiple virtual testers. If there are fewer virtual testers than IP addresses, then some IP addresses are not used. This approach optimizes the distribution of IP addresses regardless of the number of virtual testers scheduled on any particular computer, and frees you from having to match IP addresses to specific virtual testers.

## Assigning Values to System Environment Variables

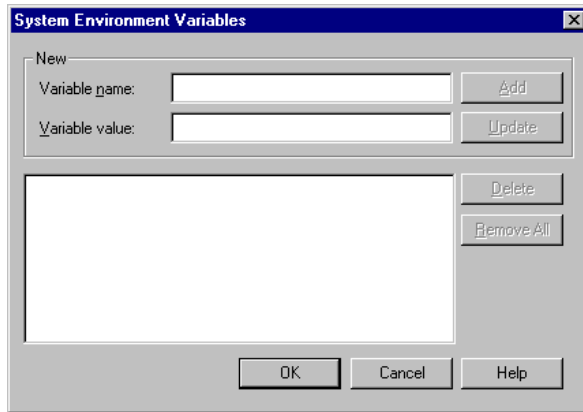
---

TestManager passes the system environment variables set on an Agent computer to each virtual tester. If you are using virtual testers to test a database server or application, you can override these system environment variables.



To override the value of a system environment variable:

- Click **Suite > Edit Settings**, and then click the button in the **Sys Environment Variables** column of the User Settings dialog box.



You can change the value or a previously set system environment variable in the System Environment Variables dialog box. For more information, see TestManager online Help.

You can set system environment variables for listed testing platforms as follows:

Testing Platform	System Environment Variable Settings
Oracle on a UNIX Agent	<p>Specify the directory that contains the client software in the variable ORACLE_HOME.</p> <p>Example:</p> <pre>ORACLE_HOME = /ora/app/oracle/product/8.0.5</pre> <p>If /var/opt/oracle does not contain tnsnames.ora, assign the pathname of the file to the variable TNS_ADMIN.</p> <p>Example: TNS_ADMIN = /home/uname/oracletest</p>
Sybase on a UNIX Agent	<p>Specify the directory that contains the client software in the variable SYBASE.</p> <p>Example: SYBASE = /usr/local/sybasec</p> <p>Specify the directory that contains the Sybase client libraries in the path of one of the following system environment variables:</p> <ul style="list-style-type: none"> <li>PATH (Windows)</li> <li>LD_LIBRARY_PATH (Solaris Agents)</li> <li>SHLIB_PATH (HP-UX Agents)</li> <li>LIBPATH (AIX Agents)</li> </ul>

Testing Platform	System Environment Variable Settings
Java on a UNIX Agent	<p>Specify the directory that contains the Java libraries in the variable <code>LD_LIBRARY_PATH</code>.</p> <p>Example:</p> <pre>LD_LIBRARY_PATH=/usr/jre118/lib/linux/native_threads</pre> <p>When the Agent computer is also using third-party software (such as IBMWebSphere) you must specify the directory location of that software's libraries in the system environment variable <code>LD_LIBRARY_PATH</code> in addition to the Java libraries.</p> <p>In addition, for Java Developers Kit version 1.1, you must also set the following variable:</p> <pre>JAVA_COMPILER=NONE</pre>
Local or Agents running TUXEDO test scripts	<p>Specify the directory that contains the client software in the variable <code>TUXDIR</code>.</p> <p>Set the <code>NLSPATH</code> environment variable to the path of the directory that contains the TUXEDO message file.</p> <p>Set the value of <code>\$TUXDIR/lib</code> to one of the following system environment variables:</p> <ul style="list-style-type: none"> <li><code>LD_LIBRARY_PATH</code> (Solaris Agents)</li> <li><code>SHLIB_PATH</code> (HP-UX Agents)</li> <li><code>LIBPATH</code> (AIX Agents)</li> </ul> <p>For Windows NT Local computers, these must be defined only for TUXEDO client-only installations. The TUXEDO full run-time installation process sets them automatically. For more information, see the TUXEDO installation instructions.</p> <p>Set one of the following:</p> <ul style="list-style-type: none"> <li>The Workstation Listener's address to <code>WSNADDR</code>.</li> </ul> <p>Example:</p> <pre>WSNADDR=//sparky:36001 WSNADDR=00028CA1C0A8F0D6</pre> <ul style="list-style-type: none"> <li>The Workstation Listener's host name and port to <code>WSLHOST</code> and <code>WSLPORT</code>. These variables override <code>WSNADDR</code>, if set.</li> </ul> <p>Example:</p> <pre>WSLHOST=sparky WSLPORT=36001</pre>

Testing Platform	System Environment Variable Settings
Local or Agents running TUXEDO test scripts that use FML typed buffer field names	<p>Set a list of FML field table file names to <code>FIELDTBLS</code>. This variable is used by Agents running test scripts that contain FML typed buffer field name references. If this variable is not set, functions that use FML typed buffer field names that are not included in this list fail, causing dependent commands to fail.</p> <p>Example: <code>FIELDTBLS=ct.fldtbl,inv.fldtbl</code></p> <p>Set the absolute pathname of the directory containing the FML field table file to <code>FLDTBLDIR</code>. This variable is used by Agents running test scripts that contain FML typed buffer field name references. If this variable is not set, functions that use FML typed buffer field names that are not included in this list (for example, <code>tux_setbuf_int()</code>) fail, causing dependent commands to fail.</p> <p>Example: <code>FLDTBLDIR=/u1/tuxapp/dat</code></p>
Local or Agents running TUXEDO test scripts that use FML32 typed buffer field names	<p>Set a list of FML32 field table file names to <code>FIELDTBLS32</code>. This variable is used by Agents that run test scripts that contain FML32 typed buffer field name references. If this variable is not set, functions that use FML32 typed buffer field names that are not included in this list fail, causing dependent commands to fail.</p> <p>Example: <code>FIELDTBLS32=ct32.fldtbl,inv32.fldtbl</code></p> <p>Set the absolute pathname of the directory containing the FML32 field table files to <code>FLDTBLDIR32</code>. This variable is used by Agents running test scripts that contain FML32 typed buffer field name references. If this variable is not set, functions that use FML32 typed buffer field names that are not included in this list (such as <code>tux_setbuf_int()</code>) fail, causing dependent commands to fail.</p> <p>Example: <code>FLDTBLDIR32=/u1/tuxapp/dat</code></p>
Local or Agents running TUXEDO test scripts that use <code>VIEW</code> , <code>X_COMMON</code> , or <code>X_C_TYPE</code> typed buffers	<p>Set a list of view description file names to <code>VIEWFILES</code>. This variable is used by Agents running test scripts that use <code>VIEW</code>, <code>X_COMMON</code> or <code>X_C_TYPE</code> typed buffers. If this variable is not set, functions that use these typed buffers that are defined in view description files not in this list fail, causing dependent commands to fail.</p> <p>Example: <code>VIEWFILES=ct.V,inv.V</code></p> <p>Set the absolute pathname of the directory containing the view description files to <code>VIEWDIR</code>. If this variable is not set, <code>tux_tmalloc()</code> or <code>tux_alloc_buf()</code> calls that try to allocate a buffer of type <code>VIEW</code>, <code>X_COMMON</code>, or <code>X_C_TYPE</code> fail, causing dependent commands or functions to fail.</p> <p>Example:  <code>VIEWDIR=/u1/tuxapp/dat:/u1/tuxapp/dat2</code></p>

Testing Platform	System Environment Variable Settings
Local or Agents running TUXEDO test scripts that use VIEW32 typed buffers	<p>Set a list of view description file names to VIEWFILES32. This variable is used by Agents running scripts that use VIEW32 typed buffers. If this variable is not set, functions that use VIEW32 typed buffers which are defined in view description files not in this list will fail, causing dependent commands to fail.</p> <p>Example: VIEWFILES32=ct32.V,inv32.V</p> <p>Set the absolute pathname of the directory containing the view description files to VIEWDIR32. This variable is used by Agents running test scripts that use VIEW32 typed buffers. If this variable is not set, tux_tpalloc() or tux_alloc_buf() calls that try to allocate a buffer of type VIEW32 fail, causing dependent commands or functions to fail.</p> <p>Example: VIEWDIR32=/u1/tuxapp/dat</p>
Solaris Agents running TUXEDO test scripts	<p>Set the TLI network service provider pathname to WSDEVICE. This value is typically /dev/tcp. If not set, playback terminates with an error message.</p> <p>Example: WSDEVICE=/dev/tcp</p>
INFORMIX on a UNIX Agent computer	<p>Assign a valid entry in the \$INFORMIXDIR/etc/sqlhosts file to INFORMIXSERVER.</p> <p>Assign a value to INFORMIXDIR. The value depends on your version of INFORMIX CLI and INFORMIX ESQ/C.</p>

# Standard Datapool Data Types

# B

This appendix contains:

- A table of standard data types
- A table of minimum and maximum ranges for the standard data types

## Standard Data Type Table

---

Data types supply datapool columns with their values. You assign data types to datapool columns when you define the columns in the Datapool Specification dialog box.

The standard data types listed in the following table are included with your Rational Test software. Use these data types to help populate the datapools that you create.

The standard data types (plus any user-defined data types you create) are listed in the Datapool Specification dialog box under the heading **Type**. You can use this dialog box to set **Type** and the other datapool column definitions (such as **Length** and **Interval**) listed in the following table.

Note that related data types (such as cities and states) are designed to supply appropriate pairings of values in a given datapool row. For example, if the Cities - U.S. data type supplies the value Boston to a row, the State Abbrev. - U.S. data type supplies the value MA to the row.

Standard data type name	Description	Examples
Address - Street	Street numbers and names. No period after abbreviations.	20 Maguire Road 860 S Los Angeles St 8th Fl 75 Wall St 22nd Fl
Cities - U.S.	Names of U.S. cities.	Lexington Cupertino Raleigh
Company Name	Company names (including designations such as Co and Inc where appropriate).	Rational Software Corp TSC Div Harper Lloyd Inc Sofinnova Inc
Date - Aug 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma ( , ) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 01011900 and <b>Maximum</b> to 12312050.</p>	<p>Oct 8, 1997 Jun 17, 1964 Nov 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"Oct 8, 1997" "Jun 17, 1964" "Nov 10, 1978"</p>
Date - August 10, 1994	<p>Dates in the format shown.</p> <p>The day portion of the string is always two characters. Days 1 through 9 begin with a blank space.</p> <p>To include the comma ( , ) as an ordinary character rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 01011900 and <b>Maximum</b> to 12312050.</p>	<p>October 8, 1997 June 17, 1964 November 10, 1978</p> <p>If the comma is the delimiter, the values are stored in the datapool as follows:</p> <p>"October 8, 1997" "June 17, 1964" "November 10, 1978"</p>

Standard data type name	Description	Examples
Date - MM/DD/YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in <b>Maximum</b> must be greater than the year in <b>Minimum</b>).</p> <p>To include the slashes ( / ) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set <b>Minimum</b> to 010100 and <b>Maximum</b> to 123199.</p>	<p>10/08/97 06/17/64 11/10/78</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows: "10/08/97" "06/17/64" "11/10/78"</p>
Date - MM/DD/YYYY	<p>Dates in the format shown.</p> <p>To include the slashes ( / ) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 01011900 and <b>Maximum</b> to 12312050.</p>	<p>10/08/1997 06/17/1964 11/10/1978</p> <p>If the slash is the delimiter, the values are stored in the datapool as follows: "10/08/1997" "06/17/1964" "11/10/1978"</p>
Date - MMDDYY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in <b>Maximum</b> must be greater than the year in <b>Minimum</b>).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set <b>Minimum</b> to 010100 and <b>Maximum</b> to 123199.</p>	<p>100897 061764 111078</p>
Date - MM-DD-YY	<p>Dates in the format shown.</p> <p>You can only specify a range of dates in the same century (that is, the year in <b>Maximum</b> must be greater than the year in <b>Minimum</b>).</p> <p>To set a range of dates from January 1, 1900 through December 31, 1999, set <b>Minimum</b> to 010100 and <b>Maximum</b> to 123199.</p>	<p>10-08-97 06-17-64 11-10-78</p>
Date - MMDDYYYY	<p>Dates in the format shown.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 01011900 and <b>Maximum</b> to 12312050.</p>	<p>10081997 06171964 11101978</p>
Date - MM-DD-YYYY	<p>Dates in the format shown.</p> <p>To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 01011900 and <b>Maximum</b> to 12312050.</p>	<p>10-08-1997 06-17-1964 11-10-1978</p>

Standard data type name	Description	Examples
Date - YYYY/MM/DD	Dates in the format shown. To include the slashes ( / ) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool. To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 19000101 and <b>Maximum</b> to 20501231.	1997/10/08 1964/06/17 1978/11/10 If the slash is the delimiter, the values are stored in the datapool as follows: "1997/10/08" "1964/06/17" "1978/11/10"
Date - YYYYMMDD	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 19000101 and <b>Maximum</b> to 20501231.	19971008 19640617 19781110
Date, Julian - DDDYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 1999, set <b>Minimum</b> to 001100 and <b>Maximum</b> to 36599.	28197 16964 31478
Date, Julian - DDDYYYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 0011900 and <b>Maximum</b> to 3652050.	2811997 1691964 3141978
Date, Julian - YYDDD	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 1999, set <b>Minimum</b> to 00001 and <b>Maximum</b> to 99365.	97281 64169 78314
Date, Julian - YYYYDDD	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 2050, set <b>Minimum</b> to 1900001 and <b>Maximum</b> to 2050365.	1997281 1964169 1978314



Standard data type name	Description	Examples
Float - X.XXX	<p>Positive and negative decimal numbers in the format shown.</p> <p>Set <b>Length</b> to the number of decimal places to allow (up to 6).</p> <p>Set <b>Minimum</b> and <b>Maximum</b> to the range of numbers to generate.</p> <p>To generate numbers with more than 9 digits (the maximum allowed with the Integers - Signed data type), use the Float - X.XXX data type and set <b>Decimals</b> to 0.</p>	<p>243.63918 -95.99 155075028157503</p>
Float - X.XXXE+NN	<p>Positive and negative decimal numbers in the exponential notation format shown.</p> <p>Set <b>Length</b> to the number of decimal places to allow (up to 6).</p> <p>Set <b>Minimum</b> and <b>Maximum</b> to the range of numbers to generate.</p>	<p>4.0285177E+068 -3.2381443E+024 8.8373255E+119</p>
Gender	<p>Either M or F, with no following period.</p>	<p>M F</p>
Hexadecimal	<p>Hexadecimal numbers.</p>	<p>1d6b77 ff 3824e7d</p>
Integers - Signed	<p>Positive and negative whole numbers. This is the default data type.</p> <p>To include negative numbers in the list of generated values, set <b>Minimum</b> to the lowest negative number you want to allow.</p> <p>Maximum range:</p> <ul style="list-style-type: none"> <li>■ <b>Minimum</b> = -999999999 (-999,999,999)</li> <li>■ <b>Maximum</b> = 999999999 (999,999,999)</li> </ul> <p>For larger numbers, use a float data type.</p> <p>If you do not specify a range, the default range is 0 through 999,999,999.</p> <p>Use this data type to generate unique data in a datapool column (for example, when you need a "key" field of unique data). You can also use Read From File and user-defined data types to generate unique data.</p>	<p>1349 -392993 441393316</p>

Standard Data Type Table

Standard data type name	Description	Examples
Name - Middle	Masculine and feminine middle names. If the middle name is preceded by a field with masculine or feminine value (such as a masculine or feminine first name), the middle name is in the same gender category as the earlier field.	Richard Theresa Julius
Name - Prefix (e.g., Mr)	Mr or Ms, with no following period. If the name prefix is preceded by a field with masculine or feminine value (such as a masculine or feminine gender designation), the name prefix is in the same gender category as the earlier field.	Mr Ms
Names - First	Masculine and feminine first names. If the first name is preceded by a field with masculine or feminine value (such as a masculine or feminine name prefix), the first name is in the same gender category as the earlier field.	Richard Theresa Julius
Names - Last	Surnames.	Swidler Larned Buckingham
Names - Middle Initial	Middle initials only, with no following period.	B M L
Packed Decimal	A number where each digit is represented by four bits. Digits are non-printable. Note that commas and other characters that can be used to represent a packed decimal number may cause unpredictable results when the datapool file is read.	Non-printable digits.
Phone - 10 Digit	Telephone area codes, appropriate exchanges, and numbers.	7816762400 4123818993 5052658498
Phone - Area Code	Telephone area codes. To generate correct area code lengths, set <b>Length</b> to 3.	781 412 505
Phone - Exchange	Telephone exchanges. To generate correct exchange lengths, set <b>Length</b> to 3.	676 381 265

Standard data type name	Description	Examples
Phone - Suffix	Four-digit telephone numbers (telephone numbers without area code or exchange). To generate correct telephone number suffix lengths, set <b>Length</b> to 4.	2400 8993 8498
Random Alphanumeric String	Strings of random upper case and lower case letters. <b>Length</b> determines the number of characters generated.	AQSEFuOZUIUpAGsEM DESieAiRFiEqiEIDiicEw edEIDiIcisewsDIEdgP
Random Alphanumeric String	Strings of random upper case and lower case letters and digits. <b>Length</b> determines the number of characters generated.	AYcHI8WmeMeM0AK4 Hsk9vGAQU79esDE 7Eeis93k4ELXie7S32siDI4E
Read From File	Assigns values from an ASCII text file to the datapool column. For example, you could export a database column to a text file, and then use this data type to assign the values in the file to a datapool column.  You can use this data type to generate unique data. You can also use the Integers - Signed and user-defined data types to generate unique data.  For information about using this data type, see "Creating a Column of Values Outside Rational Test" on page 310.	Any values in an ASCII text file.
Space Character	An empty string.	""
State Abbrev. - U.S.	Two-character state abbreviations.	MA CA NC
String Constant	A constant with the value of <b>Seed</b> . The datapool column is filled with this one alphanumeric value.	1234 AAA 1b1b
Time - HH.MM.SS	Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm).  To set a range of times from midnight to 2 pm, set <b>Minimum</b> to 0 and <b>Maximum</b> to 140000.	00.00.00 (midnight) 11.14.38 21.44.19

Standard data type name	Description	Examples
Time - HH:MM:SS	Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm). To include the colons ( : ) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool. To set a range of times from midnight to 2 pm, set <b>Minimum</b> to 0 and <b>Maximum</b> to 140000.	00:00:00 (midnight) 11:14:38 21:44:19 If the colon is the delimiter, the values are stored in the datapool as follows: "00:00:00" (midnight) "11:14:38" "21:44:19"
Time - HHMMSS	Times in the format shown. Hours range from 00 (midnight) through 23 (11 pm). To set a range of times from midnight to 2 pm, set <b>Minimum</b> to 0 and <b>Maximum</b> to 140000.	000000 (midnight) 111438 214419
Zip Code - 5 Digit	Five-digit U.S. postal zip codes. To generate the correct zip code lengths, set <b>Length</b> to 5.	02173 95401 84104
Zip Code - 9 Digit	Nine-digit U. S. postal zip codes.	021733104 954012694 841040190
Zip Code - 9 Digit with Dash	Nine-digit U.S. postal zip codes with a dash between the fifth and sixth digits.	02173-3104 95401-2694 84104-0190
Zoned Decimal	Zoned decimal numbers.	3086036 450 499658196

## Data Type Ranges

---

The following table lists the minimum and maximum ranges for the standard data types:

Type of range	Limitation
Maximum hours	23
Maximum minutes	59
Maximum seconds	59
Maximum two-digit year	99

Type of range	Limitation
Maximum four-digit year	9999
Maximum months	12
Minimum six-digit date	010100 (January 1, 00)
Maximum six-digit date	123199 (December 31, 9999)
Minimum eight-digit date	01010000 (January 1, 0000)
Maximum eight-digit date	12319999 (December 31, 9999)
Minimum negative integer (Integers - Signed)	-999999999 (-999,999,999)
Maximum positive integer (Integers - Signed)	999999999 (999,999,999)
Maximum decimal places (Float data types)	6
Male/Female title	Mr, Ms
Gender designation	M, F

## Data Type Ranges

# Index

## A

Abnormal\_term\_cnt. *See* suites, setting runtime information  
acceptance criteria of test cases 48  
access order of datapool rows 282  
adapters  
    command-line TSCA 12  
    custom 12  
    custom TSCA 12  
addresses data type 366  
Agent computers 14, 177, 226, 248  
    changing settings of 71  
    checking 95  
    controlling port numbers 358  
    Java 362  
    monitoring resources of 122, 315, 323  
    monitoring status of 122  
    Oracle 361  
    preferred user view 114  
    running suite items in parallel 183, 260  
    Sybase 361  
    TUXEDO 362  
AIX Agents 361  
Analog report 315  
analyzing results 237, 313  
ASCII text files 310  
automatically generating values for user-defined data types 302  
axes, inverting in graphs 325

## B

baseline file, editing  
    in Grid Comparator 210  
    in Object Properties Comparator 202  
    in Text Comparator 205  
baseline file, saving  
    in Grid Comparator 210  
    in Image Comparator 217

    in Object Properties Comparator 202  
    in Text Comparator 205  
benchmark tests 223  
blocks, reporting average time 351  
books, Rational xvii  
builds  
    assigning when running tests 90  
    in Test Asset Workspace 138  
    in Test Log Summary 139  
    test logs and 138  
builds, intentional changes to 153

## C

checking  
    Agent computers 95  
    suites 95  
cities data type 366  
Cleanup\_time. *See* suites, terminating  
ClearQuest 148  
client computers, performance tests and 223  
client/server TSS environment variables 74  
columns in datapools  
    assigning data types to 293  
    assigning values from a text file 310, 371  
    deleting 299  
    editing column definitions 298  
    editing values in TestManager 299  
    example of column definition 296  
    field values and 307  
    length of 294  
    maximum number 282, 309  
    names correspond to test script variables 293, 309  
    setting numeric ranges in 295  
    setting unique values in 294  
    unique 305  
    values supplied by data types 286  
command IDs  
    filtering 317, 327

- grouping Command Status reports by 320
  - sorting in reports 320
- Command Status reports 315, 344
  - automatically run 316
  - data summary style 320
  - graphing 319, 324
  - setting response ranges 319, 321
  - setting response type 319
  - setting stable loads 320
  - setting time period for 320
  - sorting command IDs 320
- Command Usage reports 315, 346
  - data summary style 320
- command-line adapters 12
- command-line TSCA 12
- Compact user view 114, 115
  - preferred with Agent computers 114
- company names data type 366
- Comparators
  - Grid 205
  - Image 210
  - Object Properties 196
  - starting 196
  - Text 203
  - viewing verification points in 152
- Compare Performance reports 315, 335, 336
  - absolute comparison 338
  - defining 336
  - graphing 319, 324
  - N/A and Undefined responses 341
  - relative comparison 339
  - setting response ranges 319, 321
  - weighted absolute comparison 338
  - weighted relative comparison 340
- computer lists
  - defining 64, 65
  - running tests on 90
- Computer view 122
- computers
  - defining 64
  - monitoring resources of 101, 122, 240, 315, 323
  - running tests on 90
  - running virtual testers on 177, 248
  - See also* Agent computers, Local computers

- configuration tests 224, 237
- configurations
  - about 5, 35
  - associating with test cases 39
  - attributes and values 36
  - defining 37
- configured test cases 39
- connect TSS environment variables 75
- constant value data type 371
- contention tests 225, 228
- controlling port numbers 358
- copying
  - datapools 299
  - graphs 328
  - log filters in Test Log 142
  - reports 328, 329
  - user-defined data types 304
- credit card numbers 297
- .csv datapool files 281, 301
- .csv exported reports 330
- cursors
  - datapool 282
    - disabling wrapping for unique row retrieval 305
- custom adapters 12
- custom histograms 107
  - adding groups to 133
  - assigning states to 133
  - deleting groups from 133
  - removing states from 133
- custom TSCA adapter 12
- customer support, Rational xviii
- customizing
  - histograms 132
  - properties 60
  - reports 317, 318
  - views 127

## D

- data types
  - assigning to a datapool column 293
  - copying 304
  - creating 288



- deleting 304
- determining which data types you need 288
- editing values in 302
- importing user-defined 303
- list of standard data types 365
- minimum and maximum values 372
- renaming 304
- role of 286
- standard and user-defined 287

datapools 284

- access order 282
- assigning data types to 293
- copying 299
- creating in TestManager 291
- creating outside Rational Test 306
- cursors 282
- data types 286
- deleting 300
- deleting columns from 299
- editing column definitions 298
- editing values, in TestManager 299
- example of column definition 296
- example of value generation 297
- exporting 301
- files 281, 301
- finding data types for 288
- functional tests and 172
- importing from another project 301
- importing from outside Rational test 300
- limits 282
- maximum number of columns 282, 309
- numeric ranges in 295
- planning 284
- role of 280, 282
- row access order 282
- setting unique values in 294
- shared virtual tester access to 99
- structure 307
- unique row retrieval 305
- where stored 281

dates

- Julian 368
- setting ranges 366, 367, 368

dates data types 366, 367, 368

debugging test scripts 130, 131

- decimal numbers 294, 369

default reports

- deleting 329
- restoring 330

default settings

- for virtual testers 72
- monitor 129, 131
- reports 330

defects

- entering in ClearQuest 149
- generating from Test log 149
- TestStudio defect form 149
- tracking 148

defining computer lists 65

delays 187, 192, 235, 263, 277

- inserting into a test script 264
- setting in a suite 187, 264
- suppressing 98

deleting

- datapool column definitions 299
- datapools 300
- log filters in Test Log 142
- reports 329
- user-defined data types 304

dependencies 191, 276

- setting 192, 277

Design Editor 46

designing tests 5, 45

disabling

- test script services 82
- TSS environment variables 82

display library routine 114, 118

distributed functional tests 170, 177, 248

- vs. performance tests 14

DLB\_FREQ. *See* dynamic load balancing selectors

DLB\_TIME. *See* dynamic load balancing selectors

documentation, Rational xvii

dynamic load balancing selectors 183, 261

## E

editing

- datapool column definitions 298
- datapool values in TestManager 299

- default monitor settings 131
- default user settings 72
- in-line 70
- suites 68
- test scripts 67
- user group properties 70
- user options 70
- user-defined data type definitions 302
- user-defined data type values 302
- editor for manual scripts 58
- empty string data type 371
- emulate emulation commands 351, 353
- emulation commands
  - displaying success or failure of 114, 116
- entering defects 149
- environment differences 153
- environment variables. *See* system environment variables, TSS environment variables
- error files
  - displaying 120, 130
  - virtual tester 147
- evaluating tests 7
- event details, displaying 144
- events 191, 276
  - displaying state of 114
  - setting 192, 277
  - viewing failed 144
- Excel, creating datapool files with 308
- executables
  - replacing 70
- executing suites 99
- executing tests 7, 89
- execution order of virtual testers 98
- Execution\_list. *See* suites, setting runtime information
- exponential notation data type 369
- exporting
  - datapools 301
  - reports 330
  - suites 87
- Extended Help 8
- extensible test script types 12

## F

- fields in datapools. *See* columns in datapools
- FLDDBLS system environment variable 363
- FLDDBLS32 system environment variable 363
- file types
  - .csv (datapool files) 281, 301
  - .csv (reports) 330
  - .spc (datapool specification files) 281, 301
- files
  - datapool file location 281
  - total open 357
  - Virtual Tester Error 120, 130
- filtering
  - group views 129
  - report data 317, 327
  - user views 128
- firewalls, controlling port numbers 358
- first names data type 370
- FLDDBLDIR system environment variable 363
- FLDDBLDIR32 system environment variable 363
- float data types 369
- floating point numbers 294, 369
- folders, test case 32
- Full user view 114, 118
- functional tests 169
- functional tests. *See* distributed functional tests

## G

- gender data type (M, F) 369
- generating
  - defects 149
  - values in datapools, example 297
- graphs
  - changing formats 326
  - copying 328
  - data point information 325
  - modifying labels 326
- Grid Comparator 205
  - comparing actual and baseline files 208
  - editing baseline file 210
  - locating differences 208
  - saving baseline file 210
  - setting display options 207

- using keys to compare data 209
- grids, displaying in graphs 325
- Group views 127
- groups
  - adding to custom histograms 133
  - deleting from custom histograms 133
  - filtering 129
- GUI histograms 107, 108
- GUI test scripts 54
- GUI test scripts and datapools
  - associating variable names and datapool columns 309
- GUI testers
  - including in performance tests 223
  - parallel selectors 183
- GUI virtual testers
  - Agent computers and 15
  - parallel selectors 260

## H

- Help
  - Extended 8
  - for TestManager xvii
- hexadecimal data type 369
- histograms
  - custom 107, 132
  - GUI 107, 108
  - HTTP 107, 109
  - IIOp 107, 110
  - SQL 107, 109
  - standard 107, 108
  - zooming in on bars 111
- HP-UX Agents 361
- HTTP emulation commands
  - reporting 347, 348, 349
- HTTP histograms 107, 109
- HTTP TSS environment variables 76

## I

- IIOp histograms 107, 110
- Image Comparator 210
  - changing color of masks and differences 214

- displaying differences 213
- locating differences 213
- Mask/OCR list 212
- masks 216
- saving baseline file 217
- unexpected active windows 217
- images, testing 210
- IME (Input Method Editor) 290, 293, 295
- implementing tests 6
  - as suites 62
- importing
  - datapools from another project 301
  - datapools from outside Rational Test 300
  - user-defined data types 303
- INFORMIXDIR system environment
  - variable 364
- Input Method Editor 290, 293, 295
- integer data type 369
- iterations
  - about 5, 40
  - associating with test cases 41
  - creating and editing 41
  - displaying suite 105

## J

- Japanese characters 288, 290
- Java, setting system environment variables 362
- JAVA\_COMPILER system environment
  - variable 362
- job classes. *See* scenarios
- Julian date data types 368

## K

- Kanji characters 290
- Katakana characters 290
- keys in unique datapool rows 304
- keys, using to compare data in columns 209

## L

- last names data types 370

- LD\_LIBRARY\_PATH system environment variable 361, 362
- legends, displaying in graphs 325
- LIBPATH system environment variable 361, 362
- listing reports
  - about 155
  - creating 160
  - customizing design layouts 155
- literal value data type 371
- load tests, about 223, 224
- Local computers 14, 226, 358
  - monitoring resources of 122, 315, 323
  - monitoring status of 122
  - running suites on 15, 226
  - TUXEDO 362
- log event details 143
- log event properties
  - associated data 145
  - configuration 144
  - general 144
  - types of 151
- log filters, turning off 142
- log folders, assigning when running tests 90
- log scale, displaying in graphs 325
- logging TSS environment variables 77
- logs 99
  - assigning when running tests 90
  - displaying 120, 130
  - folder name 101
  - manual test scripts 92
  - naming 101
  - running reports against 318

## M

- manual test scripts
  - creating 56
  - customizing properties 60
  - editors 58
  - example 57
  - external files 59
  - queries 59
  - running 91

- specification files 59
- steps in 56
- verification points in 57
- viewing results of run 92
- manuals, Rational xvii
- ManualTest, Rational 57
- mapping resource usage onto response time 323
- masks in Image Comparator 214, 216
- maximum response time 239, 332
- mean response time 240, 332
- median response time 240, 332
- memory
  - minimum shared for large user runs 355
  - total shared for large user runs 357
- Message user view 114, 118
- Microsoft Excel, creating datapool files with 308
- middle initials data type 370
- middle names data type 370
- milestones 40
- minimum response time 239, 332
- monitoring computer resources 122, 240, 315, 323
  - setting option to allow 101
- monitoring suites 102
  - changing default settings 131
  - Compact user view 114, 115
  - Computer view 122
  - Full user view 114, 118
  - Group views 127
  - Message user view 114, 118
  - Results user view 114, 116
  - Script view 120
  - setting update rates 101
  - Shared Variables view 119
  - Source user view 114, 117
  - Sync Points view 120
  - Transactor views 125
- multi-byte characters 288, 290, 293, 295
- multiple computers 65

## N

- N\_users. *See* suites, running
- names data types

- company names 366
- first names 370
- last names 370
- middle initials 370
- middle names 370
- titles (Mr, Ms) 370
- naming reports 329
- network services 358
- NLSPATH system environment variable 362
- Normal\_term\_cnt. *See* suites, setting runtime information
- numbers data type 369
- NuTCRACKER settings, changing when running
  - large numbers of users 355

## O

- Object Properties Comparator 196
  - adding properties to test 201
  - displaying differences in baseline and actual files 200
  - editing baseline file 202
  - locating differences 201
- Objects hierarchy 198
- Properties list 198
- removing properties 201
- saving baseline file 202
- OCR regions, creating in Image Comparator 216
- Oracle, setting system environment
  - variables 361
- ORACLE\_HOME system environment
  - variable 361
- outliers 240, 319, 321
- output file, virtual tester 147
- owners of test assets, specifying 34

## P

- packed decimal data type 370
- parallel selectors 183, 260
- PATH system environment variable 361, 362
- Performance reports 238, 239, 315, 332
  - automatically run 316
  - comparing output of 335

- creating 161
- dynamic number of virtual testers 322
- graphing 319, 324
- setting response ranges 319, 321
- setting response time calculation 320
- setting response time percentiles 320
- setting response type 319
- setting stable loads 232, 320, 321
- setting time period for 320
- sorting command IDs 320
- performance tests 222, 229
  - basic concepts 221
  - including GUI testers in 223
  - reports 156
    - vs. distributed functional tests 14
- PERMUTE. *See* random without replacement selectors
- persistent datapool cursors 282
- phone numbers data types 370, 371
- planning
  - datapools 284
  - performance tests 229
- planning tests 3, 25
- playback differences 153
- populating datapools
  - example 297
- port numbers 358
- post-conditions of test cases 48
- preconditions
  - suites 178, 182, 250, 254
  - test cases 178, 181, 250, 253
  - test scripts 178, 250
- pre-conditions of test cases 48
- printing
  - reports 328
  - suites 87
  - test case designs 47
- processes, total TestManager 357
- projects 8
- properties
  - manual test scripts 60
  - report 324, 328
  - suites 69
  - test case folders 32
  - test cases 33

test plans 29  
properties of test scripts 67

## R

random alphabetic string data type 371  
random alphanumeric string data type 371  
random datapool access 282  
random numbers 84  
random value seed 295  
random with replacement selectors 183, 260  
    generating random numbers for 99  
random without replacement selectors 183, 260  
    generating random numbers for 99  
ranges in dates 366, 367, 368  
Rational Administrator 8  
Rational ClearQuest 10, 148, 158  
Rational ManualTest 57  
Rational projects 8  
Rational QualityArchitect 9  
Rational RequisitePro 10, 27  
Rational Robot 9  
Rational Rose 10, 27  
Rational SoDA 11, 158  
Rational Technical Support xviii  
Rational TestManager 1  
Rational Unified Process 8  
Read From File data type 310, 371  
    unique values 311  
records in datapools. *See* rows in datapools  
release times 191, 275  
    ranges 191, 275  
    staggering 190, 273  
renaming  
    log filters in Test Log 142  
    reports 329  
    user-defined data types 304  
reporting TSS environment variables 79  
reports 313  
    about 154  
    changing default settings 330  
    changing graph formats 326  
    changing reports that run automatically 162,  
        330

Command Status 315, 344, 345  
Command Usage 315, 346, 347  
Compare Performance 315, 335, 336, 337  
    comparing 336  
    copying 328, 329  
    creating 158  
    customizing 317, 318  
    default names of 314  
    deleting 329  
    displaying grids 325  
    displaying legends 325  
    displaying log scales 325  
    dynamic number of virtual testers 322  
    exporting 330  
    filtering data 317, 327  
    inverting axes 325  
    Listing 155  
    opening 161  
    Performance 238, 239, 315, 332, 333  
    performance testing 156  
    printing 328  
    properties of 324, 328  
    Rational ClearQuest 158  
    Rational SoDA 158  
    renaming 329  
    Response vs. Time 238, 239, 341, 343  
    restoring default 162, 330, 331  
    running from menu bar 162, 316  
    running from report bar 161, 316  
    saving 135  
    selecting which to run 156  
    setting response time calculations 320  
    setting response type 319  
    setting stable loads 232, 320, 321  
    setting time period for 320  
    sorting command IDs 320  
Test Case Results Distribution  
    reports  
        Test Case Trend 154  
    types of 315  
resource monitoring 122, 240, 315, 323  
    setting option to allow 101  
response timeout TSS environment variables 80  
response times  
    reporting on 239, 320

- standard deviation 240, 332
- Response vs. Time reports 238, 239, 341
  - graphing 319, 324
  - including passed or failed responses 320
  - resource monitoring 315, 323
  - setting response ranges 319, 321
  - setting response time calculations 320
  - setting response type 319
  - setting stable loads 320
  - setting time period for 320
  - sorting command IDs 320
- restoring default reports 162, 330, 331
- Results user view 114, 116
- row access order 282
- rows in datapools
  - access order 282
  - maximum number 282
  - records and 307
  - unique 305
- RT\_MASTER\_NTUSERLIMIT system environment variable 356
- Run\_time. *See* suites, setting runtime information
- running
  - automated test scripts 90
  - manual test scripts 91
  - reports 161, 162, 316
  - suites 15, 94, 99, 226
  - test cases 92
  - test scripts 89
  - suites *See also* monitoring suites
  - suites. *See also* monitoring suites
- running test on multiple computers 65
- running tests 89

## S

- saving suites 87
- scenarios 184, 192, 255, 277
  - and suites 186, 256
  - replacing 70
- scientific notation data type 369
- scope of a synchronization point 190, 275
- Script view 120
- seeds 86

- base 99
  - for random selectors 99
  - for virtual testers 84
  - random datapool values 295
- selectors 182, 257
  - dynamic load balancing 183, 261
  - inserting into a suite 183, 257
  - parallel 183, 260
  - random 183, 260
  - sequential 183, 260
- semaphores
  - changing maximum number of 355, 357
  - per set 357
  - set IDs 357
- sequential datapool access 282
  - unique row retrieval and 306
- sequential selectors 183, 260
- servers, testing 233
- sessions, creating suite from 246
- shared datapool cursors 282
- shared memory 355, 357
- shared variables
  - changing value of 130
  - displaying users waiting on 130
  - initializing 86
  - viewing values of 119
- SHLIB\_PATH system environment variable 361, 362
- shuffle datapool access 282
  - unique row retrieval and 306
- socket emulation commands
  - reporting 347, 348, 350
- Solaris Agents 361
- sorting virtual testers 128
- Source user view 114, 117
- space data type 371
- .spc datapool specification files 281, 301
- SQABasic language, monitoring 113
- SQL emulation commands
  - reporting 347, 348
- SQL histograms 107, 109
- stable loads
  - planning 232, 238, 239
  - setting in reports 232, 320, 321
- staggering release times 190, 273

- standard data types
  - list of 365
  - minimum and maximum values 372
  - role of 287
  - when to use 288
- standard deviation of response times 240, 332
- standard histograms 107, 108
- start scripts 83
  - setting maximum initialization time for 98
- Start\_group\_size. *See* suites, setting runtime information
- start\_time emulation commands
  - reporting on 353
- Start\_time. *See* suites, setting runtime information
- state abbreviations data type 371
- states
  - assigning to custom histograms 133
  - removing from custom histograms 133
- steps in manual test scripts 56
- stop\_time emulation commands
  - reporting on 351
- street names data type 366
- stress tests 225, 236, 271
- string constant data type 371
- structure of datapools 307
- suites 15, 173, 243
  - and scenarios 186, 256
  - changing logs in 101
  - checking 95
  - creating from a session 246
  - editing 68
  - execution order of virtual testers 98
  - exporting 87
  - implementing tests as 62
  - inserting selectors 182, 183, 257
  - inserting test cases 179, 252
  - inserting test scripts 177, 249
  - inserting transactors 265
  - inserting user groups 246
  - minimum requirements for running 95, 179, 251
  - monitoring 102
  - opening 67
  - percent done 104
  - preconditions 178, 182, 250, 254
  - printing 87
  - replacing items in 70
  - reporting on portion of 320
  - results 99
  - running 94, 99
  - saving 87
  - setting delays in 187, 264
  - setting maximum time for run 98
  - setting number of virtual testers 100
  - setting pass or fail criteria 97
  - setting runtime information 96
  - synchronization points and 188, 270
  - synchronizing items in 187, 188, 191, 263, 269, 270, 276
  - terminating 99, 101, 134, 135, 352
  - time in run 104
- survey utility. *See* suites, monitoring
- suspending virtual testers 131, 134
- Sybase, setting system environment
  - variables 361
- synchronization points 120, 187, 188, 190, 269, 270, 273
  - displaying state of 121
  - example of 190, 274
  - inserting into a suite 270
  - inserting into a test script 270
  - inserting into suites 187, 188, 269, 270
  - multiple 191, 274
  - number of virtual testers waiting 120, 122
  - release time ranges 191, 275
  - releasing 122
  - releasing virtual testers from 120, 190, 191, 273, 275
  - replacing 70
  - scope of 190, 275
  - timeout 121, 191, 275
- synchronizing items in suites
  - delays 187, 263
  - events and dependencies 191, 276
  - synchronization points 187, 188, 269, 270
- system environment variables 360



## T

Task\_dir. *See* suites, running

Task\_term\_cnt. *See* suites, terminating

Task\_ts\_init. *See* suites, setting runtime information

tasks. *See* test scripts

technical publications, Rational xvii

technical support, Rational xviii

terminating suites 99, 101, 134, 352  
    planning 232

terminating virtual testers 131  
    abnormal 104, 115, 116, 127  
    normal 104, 127

Test Asset Workspace  
    copying reports with 329  
    deleting reports with 329  
    renaming reports with 329

Test Case Distribution reports  
    about 154  
    creating 158

test case folders  
    about 5  
    creating 31  
    properties 32

Test Case Results Distribution reports  
    about 154  
    creating 159

Test Case Trend reports  
    about  
    reports

### Test Case Distribution 154

    creating 160

test cases 5  
    acceptance criteria 48  
    associating configurations with 39  
    associating iterations with 41  
    associating test inputs with 42  
    configured 39, 180, 253  
    creating 33  
    displaying 143  
    filtering 143  
    folders 5  
    inserting into a suite 179, 252  
    inserting into test case folder 33

    organizing in folders 31

    post-conditions 48

    pre-conditions 48

    preconditions in suites 178, 181, 250, 253

    properties 33

    running 92

    sorting 143

    viewing events from 143

    when to run 40

test input types

    built-in 26

    custom 28

Test Input window 26

test inputs

    about 4

    appearing in defect form 149

    associating with test cases 42

    defining what to test 26

test log filters

    turning off 142

Test Log window

    virtual tester error file 147

Test log window 97

    associated data 151

    entering defects 148, 149

    log event details 143

    log event properties 143

    log event property types 151

    main window 138

    usage scenarios 137

    user interface 138

    virtual tester output file 147

test plans

    about 4

    creating 29

    properties 29

test script properties 67

test script types, extensible 12

test scripts

    changing number 83

    debugging 130, 131

    editing 67

    grouping into scenarios 184, 255

    GUI 54

    initializing timestamps for 98

- inserting delays 264
- inserting into a suite 177, 249
- limiting number 83
- opening from Test log window 146
- preconditions 178, 250
- replacing 70
- reporting active times of 347
- reporting inactive times of 347
- running 89
- setting dependencies 192, 277
- setting events 191, 276
- variable names and datapool column names 293
- VB (Visual Basic) 54
- TEST7\_LTMASTER\_SHM\_MINSZ system environment variable 356
- testcase emulation commands 352, 353
- TestManager 227
  - about 1
  - hardware and software environment 228
  - total processes 357
- tests
  - benchmark 223
  - configuration 224
  - contention 225, 228
  - designing 5, 45
  - evaluating 7
  - executing 7, 89
  - implementing 6
  - load 224
  - performance 222, 229
  - planning 3, 25
  - stress 225, 236, 271
- TestStudio defect schema and form 149
- Text Comparator 203
  - comparing actual and baseline files 204
  - editing baseline file 205
  - locating differences 204
  - saving baseline file 205
  - viewing verification point properties 204
- text files, assigning values to a datapool column 310, 371
- think time TSS environment variables 81
- throughput 348, 352
- time data types 371, 372
- timeout values, for synchronization points 191, 275
- times
  - reporting active 347
  - reporting inactive 347
  - setting maximum initialization 98
  - setting maximum suite run 98
  - standard deviation of response 240, 332
  - suppressing delays 98
- timestamps, initializing for test scripts 98
- TNS\_ADMIN system environment variable 361
- Total\_term\_cnt. *See* suites, setting runtime information
- Trace report 315
- traceability 42
- training, Rational xvii
- Transaction reports. *See* Response vs. Time reports
- transactors 125, 192, 265, 277
  - displaying information about 125
  - inserting into a suite 265
- TSS environment variables 73
  - client/server 74
  - connect 75
  - disabling 82
  - HTTP 76
  - logging 77
  - reporting 79
  - response timeout 80
  - think time 81
- TUXEDO
  - emulation commands in Command Usage report 347, 348, 351
  - setting system environment variables 362

## U

- unexpected active windows 217
- unique datapool rows
  - guidelines for 304
  - Read From File data type and 311
  - setting unique values 294
  - user-defined data types and 289
- update rates, when monitoring suites 101

- U.S. cities data type 366
- U.S. state abbreviations data type 371
- user groups 176, 246
  - changing information about 71
  - displaying information about 127
  - editing properties of 70
  - fixed 247
  - inserting into suites 176, 246, 247
  - planning 231
  - replacing 70
  - reserved words 247
  - scalable 247
  - temporarily disabling 84, 98
- user settings, changing 72
- User\_term\_mode. *See* suites, terminating
- user-defined data types 288
  - automatically generating values for 302
  - copying 304
  - deleting 304
  - editing definitions of 302
  - editing values in 302
  - importing 303
  - renaming 304
  - role of 287
  - unique values 289
  - when to use 288
- userlists. *See* user groups

## V

- variable names, and datapool column
  - names 293, 309
- VB test scripts 54
- verification points
  - manual test scripts 57
  - viewing in Comparators 152
- VIEWDIR system environment variable 363
- VIEWDIR32 system environment variable 364
- VIEWFILES system environment variable 363
- VIEWFILES32 system environment variable 364
- viewing
  - datapool values in TestManager 299
  - user-defined data type values 302
  - verification points in the Comparators 152

- views
  - Compact user 114, 115
  - Computer 122
  - customizing 127
  - Full user 114, 118
  - Message user 114, 118
  - restoring default 129
  - Results user 114, 116
  - Script 120
  - Shared Variables 119
  - Source user 114, 117
  - Sync Points 120
- virtual tester
  - error file 147
  - output file 147
- Virtual Tester Error files
  - displaying 120, 130
- virtual testers 249
  - abnormal termination 104, 116, 127
  - active 103
  - combining test script types 249
  - determining number supported 233
  - displaying information about 104
  - execution order 98
  - filtering 128
  - incrementally loading 234
  - limiting number of test scripts run by 83
  - normal termination of 104, 127
  - percentage executing suite 105
  - reporting active times of 347
  - reporting inactive times of 347
  - reporting on dynamic number of 322
  - resuming suspended 131
  - running large numbers 355
  - seeds for 84
  - setting limit for large virtual tester runs 356
  - setting number of 100, 176, 247
  - sorting 128
  - starting at different times 83
  - suspended 103, 127
  - suspending 131, 134
  - terminating 131
  - total number in run 103

## W

Windows NT Agents 361

workloads 14

- adding to system-under-test 14, 223, 226

- designing 231

- planning stable 232, 238, 239

- reporting on stable 232, 320, 321

WSDEVICE system environment variable 364

WSLHOST system environment variable 362

WSLPORT system environment variable 362

WSNADDR system environment variable 362

## Z

Zap\_mode. *See* suites, setting runtime information

zip code data types 372

zoned decimal data type 372