Using Rational TestManager

VERSION 2001A.04.00

PART NUMBER 800-024531-000

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

Rational®
the e-development company™

Contents

Part 3: Performance Testing with Rational TestManager

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.

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
- Functional and performance testing
- Local and Agent computers
- Suites
- Starting Rational TestManager
- The TestManager main window

What Is Rational TestManager

Testing is the feedback mechanism in the software development process. It tells you where corrections need to be made to stay on course at any given iteration of a development effort. It also tells you 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. Further, they need to understand and define the relationships between the elements of their testing plan and the other elements 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.
- Project managers need to use the information that 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 draw their knowledge of the state of the system from these same measurements.

Testing efforts often represent 25–50% of the overall project effort. Collecting the required data, tracking the relationships among test assets, and providing a common presentation of the output of the testing effort often involves use of several tools. 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, assets, and data both related to and produced by the testing effort. Under this single framework, all 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 point in time.

Quality assurance professionals can use TestManager to coordinate and track their testing activities. Testers use TestManager to see what work needs to be done by whom and by what date. Testers can also see 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

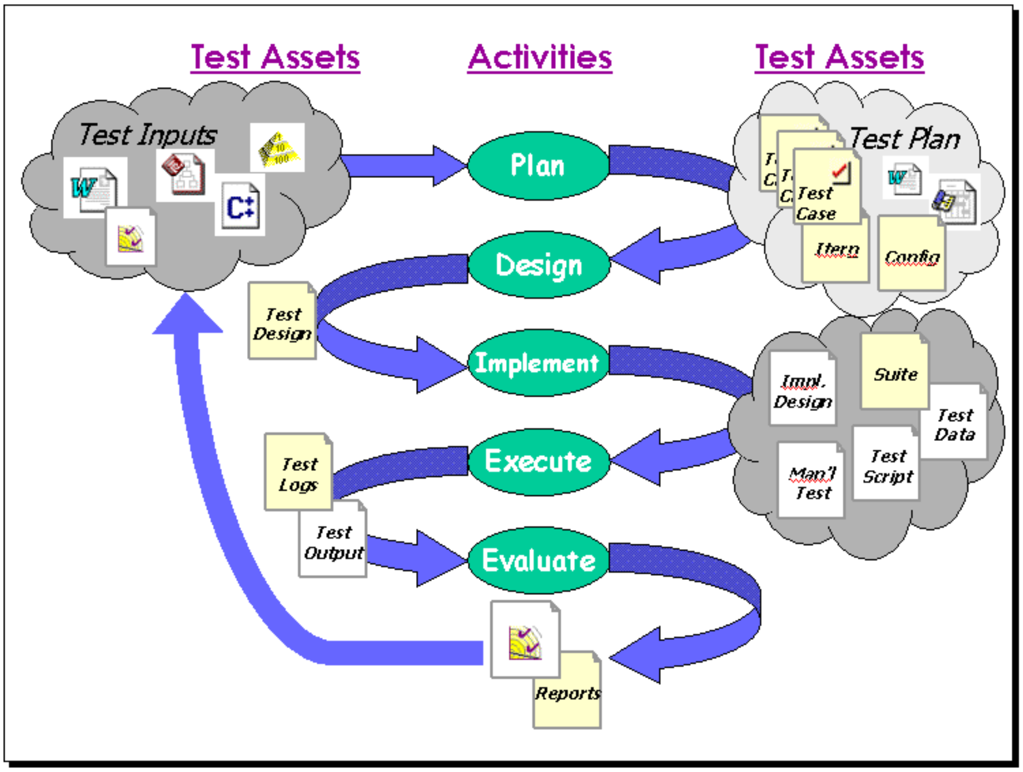
The TestManager workflow supports the five major testing activities defined by the Rational Unified Process, which is a software engineering process:

- Planning tests
- Designing tests
- Implementing tests
- Executing tests

- Evaluating tests

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

Testing Workflow



Planning Tests

The activity of planning tests involves answering the following questions:

- What and Where? – Requirements, visual models, and other test inputs tell you what to test and where to run the tests.
- Why – Test inputs tell you why you are going to do certain tests. For example, tests may be performed to validate system requirements.
- When? – Iteration plans tell you when the tests must be run and must pass.
- Who? – Test plans, iteration plans, or project plans tell you who will perform the testing activities.

For information about planning tests, see *Planning Tests* on page 23.

Test Inputs

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 important in iterative development where change is a frequent, necessary part of the process.

TestManager has two built-in test input types:

- Requirements 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 assets for traceability purposes.

TestManager also supports custom test input types. To use a custom test input within TestManager, you need to write a Test Input Adapter or use an adapter provided by Rational Software or Rational's partners. For example, to use the values in a Microsoft Excel spreadsheet as test inputs, you would need to write a test input adapter for Excel. For more information, see *Defining Custom Test Input Types* on page 13.

Test Plans

When you have identified your test inputs, you can use TestManager to create a test plan. The *test plan* provides an organizational structure for the other test assets in the project.

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

- What tests must be performed?
- When must the tests be performed and be expected to pass?
- Who is responsible for each test?
- Where must the tests 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 (for example, test the file maintenance utility).

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

Test Case Folders

Within a test plan, you can create *test case folders* to organize your test cases hierarchically. Common organizations may reflect system architecture, major use cases, requirements, or combinations of these.

Test Cases

The *test case* is the test asset in TestManager that answers the question, “What am I going to test?” You develop test cases to define the individual things that you need to validate to ensure that the system is working the way that it is supposed to work and is built with the quality necessary before you can ship it.

Each test case is owned by 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*. At some point in time during an iteration, 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.

Iterations might be defined by several team members — such as project managers, product managers, and analysts — iteratively throughout the testing process.

Configurations

Use *configurations* to specify where test cases must be run — on what hardware and software configurations. For example, to ensure that your test case passes when it runs on 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 c*onfigured test cases*. In order for the test case to pass, all of its configured test cases must pass.

Configurations might be defined by several team members — such as project managers, product managers, and analysts — iteratively throughout the testing process.

Designing Tests

The activity of designing tests answers the question, “How am I going to perform the testing?” A complete test design informs readers about what actions need to be taken with the system and what behaviors and characteristics they should expect to observe if the system is functioning properly.

Designing tests is an iterative and ongoing process. You should be able to start designing tests 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 should become more detailed along with it.

Note: A test design is different from the design work that should be done in determining how to build your test implementation.

In TestManager, you can design your test cases by:

- Indicating the basic steps needed to interact with the application and the system in order to perform the test.
- Indicating how to validate that the features are working properly.
- Specifying the preconditions and post-conditions for the test.
- Specifying the acceptance criteria for the test.

Given the iterative nature of software development, the design is typically more abstract (less specific) than a manual implementation of the test, but it can easily evolve into one.

For information about designing test, see *Designing Tests* on page 47.

Implementing Tests

The activity of implementing tests involves the design and development of reusable *test scripts* that implement your test case. After you create the implementation, you can associate it 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 another project, you might need to write modular pieces of software using a combination of tools.

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

Type of Test Script	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.)

Type of Test Script	Description
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.) **Note:** When you start to record a VU test script, you actually record a session. You can generate VU or VB (Visual Basic) test scripts from the recorded session, depending on a recording option that you select in Robot.
Manual	A set of testing instructions to be run by a human tester. Created in Rational ManualTest.

TestManager also supports implementation of other types of test scripts that you have registered. For information, see *Defining Custom Test Script Types* on page 13.

You can also use suites to implement tests. A *suite* is a container that lets you design a larger set of test cases and implementations that you want to run. A suite can have parameters such as order, dependencies, iterations, random operations, and so on.

For information about implementing test cases, see *Implementing Tests* on page 55.

Executing Tests

The activity of executing your tests involves running the test implementations to make sure that the system functions correctly. In TestManager, you can run any of the following:

- An individual test script
- One or more test cases
- A suite, which runs any combination of test cases and test scripts across one or more computers and virtual testers.

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

Type of Test Script	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.

Type of Test Script	Description
VB	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 types of test scripts that you have registered. For information, see *Defining Custom Test Script Types* on page 13.

For information about executing tests, see *Executing Tests* on page 87.

Evaluating Tests

The activity of evaluating tests involves:

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

For information about evaluating tests, see *Evaluating Tests* on page 127.

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 for all critical activities. It delivers software best practices via guidelines, templates, and tool mentors.

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 Rational *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.

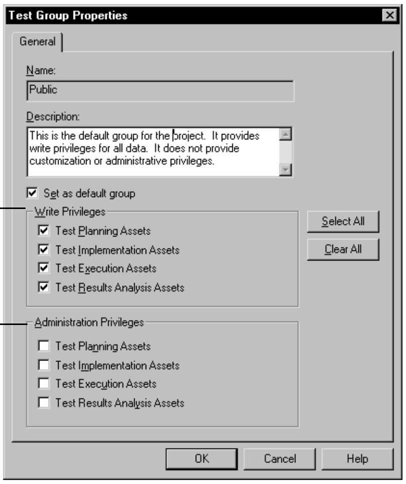
A Rational project can consist of the following:

- *Rational Test datastore* – Stores application testing information such as test plans, test cases, test logs, reports, and builds.
- *Rational RequisitePro project* – Stores product or system requirements, software and hardware requirements, and user requirements. When a RequisitePro datastore is associated with a project in the Rational Administrator, TestManager automatically uses the requirements in the datastore as test inputs.
- *Rational Rose models* – Stores visual models for business processes, software components, classes and objects, and distribution and deployment processes. You can use Rose model elements as test inputs.
- *Rational ClearQuest database* – Stores change-request information for software development, including enhancement requests, defect reports, and documentation modifications.

Security and Privileges for the Rational Test Datastore

When administrators create Rational projects using the Rational Administrator, they determine the security of the Rational Test datastore. When they create test users, the test users become part of the Public test group, by default. The test users take on the privileges of that test group. An administrator can change group privileges and create new groups using the Rational Administrator.

The following figure shows the Test Group Properties dialog box in the Rational Administrator:



Test Group Properties

General

Name: Public

Description: This is the default group for the project. It provides write privileges for all data. It does not provide customization or administrative privileges.

☑ Set as default group

Write Privileges
- ☑ Test Planning Assets
- ☑ Test Implementation Assets
- ☑ Test Execution Assets
- ☑ Test Results Analysis Assets

Administration Privileges
- ☐ Test Planning Assets
- ☐ Test Implementation Assets
- ☐ Test Execution Assets
- ☐ Test Results Analysis Assets

Select to give privileges to a group to create, modify, copy, or delete test assets.

Select to give privileges to a group to customize test assets.

For information about setting privileges, see *Using the Rational Administrator* manual or the Rational Administrator Help.

Automated Test Scripts and Rational Robot

With Rational Robot, you can 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 system is performing within user-defined response-time standards under varying loads.
- Test applications developed with IDEs (Integrated Development Environments) 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 test log in TestManager.

Component Testing and Rational QualityArchitect

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

QualityArchitect, in conjunction with Rational Rose, generates test scripts for components and interactions in your Rose model. When generated, the test scripts can be edited and run from your 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. Test scripts can be generated directly from Rose interaction diagrams or from live components using the Session Recorder.
- Generate stubs that you can use 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 Rational 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. You can also use requirements in other RequisitePro projects as test inputs.

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.

You can use Rose model elements as test inputs in TestManager, and you can easily associate the model 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, you can submit defects directly from a test log into ClearQuest. TestManager automatically fills in some of the fields in the ClearQuest defect form with information from the test log. The defect ID is automatically recorded in 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 RequisitePro, Rose, 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 about a software development project:

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

TestManager and Extensibility

By providing various application programming interfaces (APIs), TestManager is not limited to supporting software development assets and test assets generated by Rational tools. The APIs provide hooks into and from the TestManager software, enabling implementers to plug in functionality that suits specific testing purposes.

In addition to the built-in test input types and test script types, TestManager supports custom test input types and custom test script types.

Defining Custom Test Input Types

Any kind of object needed for testing can be defined and managed as a test input type — for example, objects in Microsoft Project files or Excel spreadsheets.

As an example, you could define C++ language project files as a test input type if you wanted to know which tests needed to be changed or rerun when a source file changes.

For TestManager to support a custom test input type, there must be a user-implemented dynamic-link library (DLL) called a Test Input Adapter (TIA). The adapter includes functions for tasks such as connecting to and disconnecting from the test input source, checking whether an input has been modified, and setting various kinds of filters. For information about custom adapters, see the *Rational TestManager Extensibility Reference* manual.

For information about defining new test input types in TestManager, see *Custom Test Input Types* on page 27.

Defining Custom Test Script Types

TestManager's extensible test script type functionality enables you to implement custom test scripts using any tool that is appropriate for your testing environment.

There are several ways to extend TestManager to support a new test script type:

- Use the Command Line Test Script Console Adapter to create, open, and edit test scripts.
- Build a custom Test Script Console Adapter or use an adapter provided by Rational or its partners to open (and optionally create and edit) test scripts. (For information about custom adapters, see the *Rational TestManager Extensibility Reference* manual.)
- Use the Command Line Test Script Execution Adapter to run test scripts.
- Build a custom Test Script Execution Adapter or use an adapter provided by Rational or its partners to run test scripts. (For information about custom adapters, see the *Rational TestManager Extensibility Reference* manual.)

For information about defining test script types in TestManager, see *Custom Test Script Types* on page 57.

Virtual Testers

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

Note: Virtual testers run on computers that have Agent software installed. For information, see *Local and Agent Computers* on page 15.

A virtual tester running a performance test emulates traffic between a client and its servers. 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.

Performance testing allows you add a workload to a client/server system by running many virtual testers on a server. 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 functional testing, performance testing, or both.

Functional Testing

In functional testing, you typically test the accuracy of the application and how it behaves on different computers.

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, see Part 2, *Functional Testing with Rational TestManager*.

Performance Testing

In performance testing, you can measure the following:

- 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 user to enter a request, the server to respond to the request, and the user 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 acceptable. Therefore, when planning performance tests, you need to put some thought into what constitutes acceptable performance.

Another complexity 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, see Part 3, *Performance Testing with Rational TestManager.*

Local and Agent Computers

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

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 to:

- Add 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.
- Run 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.
- Run 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.
- Test configurations. If you are testing different hardware and software configurations, you can run test scripts on different Agent computers that are set up with these configurations.

Suites

Multiple test scripts and multiple computers can be involved in a test. *Suites* enable you to coordinate the way that the test scripts run. In functional testing, suites let you run test scripts in parallel on the computers that are available, so that your tests can run more quickly. In performance testing, suites add workload to the server.

When you have used TestManager to create suites, you can run these suites repeatedly against successive builds of your product, and then analyze the results using TestManager's reporting tools.

For information about suites, see *Implementing Tests as Suites* on page 67.

Starting Rational TestManager

Before you start using TestManager, you need to have:

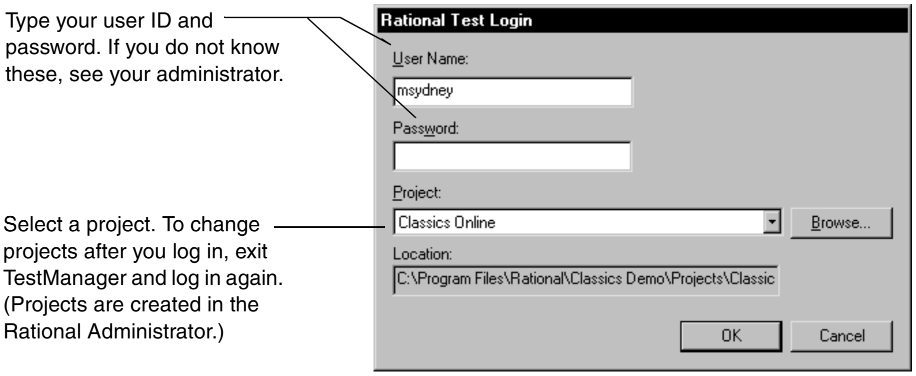
- Rational TestManager installed. For information, see the *Installing Rational Testing Products* manual.
- A Rational project. For information, see the *Using the Rational Administrator* manual or the Rational Administrator Help.

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:

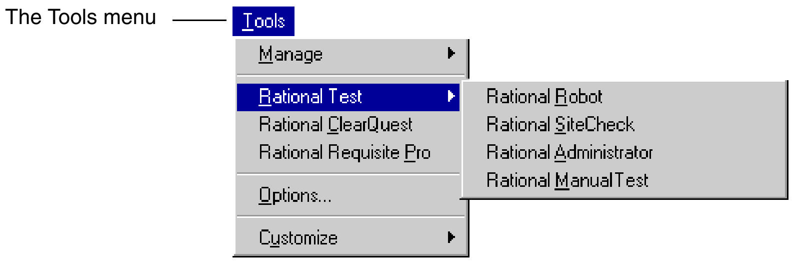
- Click **Start > Programs >** ***Rational product name*** **> Rational TestManager** to open the Rational Test Login dialog box.

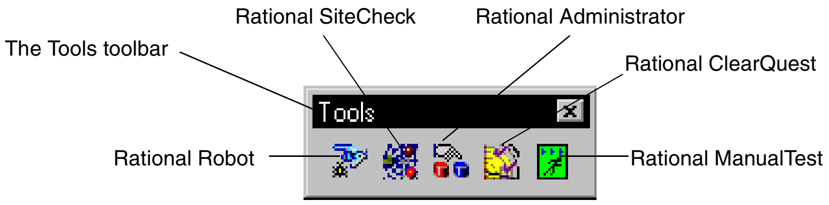

Type your user ID and password. If you do not know these, see your administrator.
Select a project. To change projects after you log in, exit TestManager and log in again. (Projects are created in the Rational Administrator.)
Rational Test Login
Browse...
OK
Cancel

User Name: msydney
Password:
Project: Classics Online
Location: C:\Program Files\Rational\Classics Demo\Projects\Classic

Starting Other Rational Products and Components from TestManager

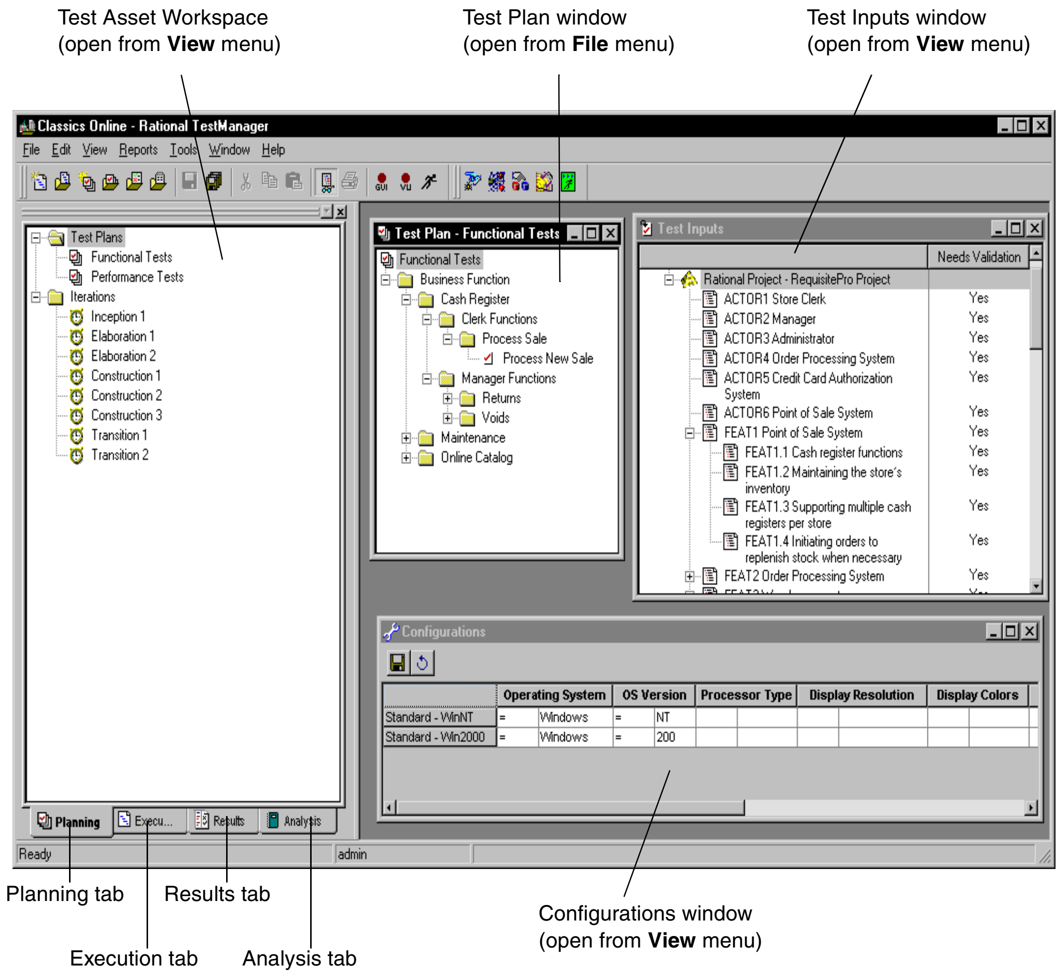
When you are logged into TestManager, you can start other Rational products and components from either the Tools menu or the Tools toolbar.





The TestManager Main Window

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



	Needs Validation
Rational Project - RequisitePro Project	
ACTOR1 Store Clerk	Yes
ACTOR2 Manager	Yes
ACTOR3 Administrator	Yes
ACTOR4 Order Processing System	Yes
ACTOR5 Credit Card Authorization System	Yes
ACTOR6 Point of Sale System	Yes
FEAT1 Point of Sale System	Yes
FEAT1.1 Cash register functions	Yes
FEAT1.2 Maintaining the store's inventory	Yes
FEAT1.3 Supporting multiple cash registers per store	Yes
FEAT1.4 Initiating orders to replenish stock when necessary	Yes
FEAT2 Order Processing System	Yes

	Operating System		OS Version		Processor Type		Display Resolution		Display Colors	
Standard - WinNT	=	Windows	=	NT						
Standard - Win2000	=	Windows	=	200						

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, and Analysis.

To show or hide the Test Asset Workspace:

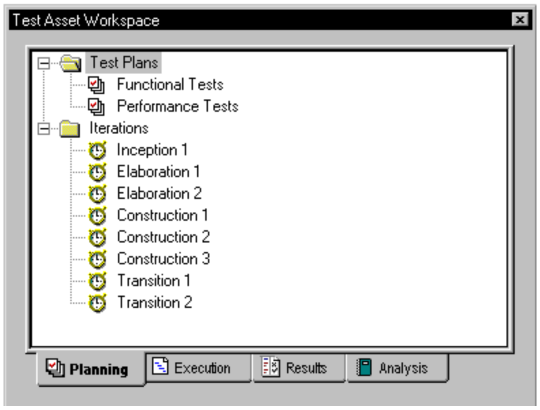
- Click **View > Test Asset Workspace**.

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.

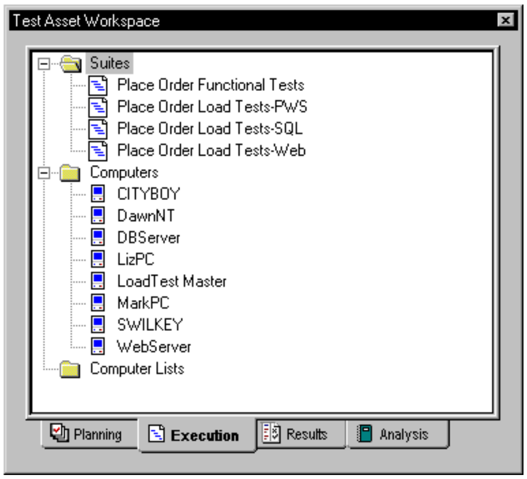


Right-click any test asset to display a shortcut menu.

For information about test plans, see *Creating a Test Plan* on page 28. For information about iterations, see *Specifying When to Run Tests* on page 43.

Execution Tab

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

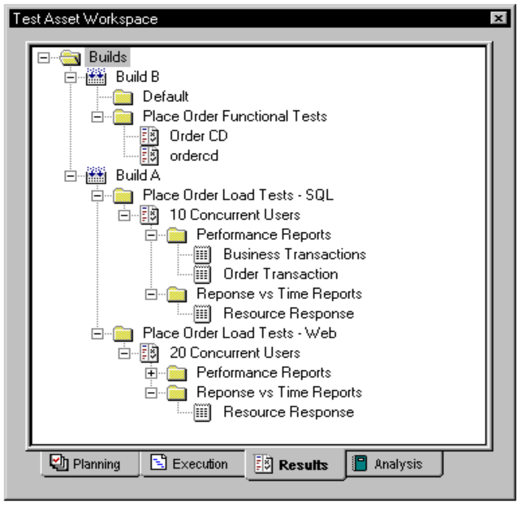


Right-click any test asset to display a shortcut menu.

For information about suites, see *Implementing Tests as Suites* on page 67. For information about computers and computer lists, see *Defining Agent Computers and Computer Lists* on page 69.

Results Tab

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



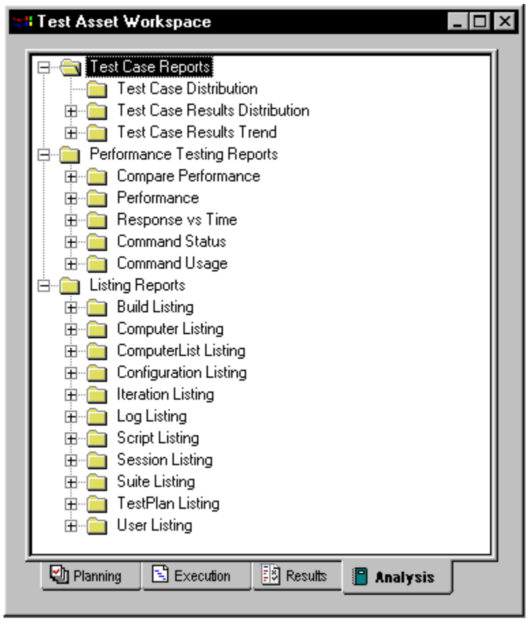
- Builds
  - Build B
    - Default
    - Place Order Functional Tests
      - Order CD
      - ordercd
  - Build A
    - Place Order Load Tests - SQL
      - 10 Concurrent Users
        - Performance Reports
          - Business Transactions
          - Order Transaction
        - Reponse vs Time Reports
          - Resource Response
    - Place Order Load Tests - Web
      - 20 Concurrent Users
        - Performance Reports
        - Reponse vs Time Reports
          - Resource Response

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

For information about builds, test log folders, and test logs, see *Evaluating Tests* on page 127.

Analysis Tab

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



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

For information about reports, see *Reporting Results* on page 145 and *Reporting Performance Testing Results* on page 291.

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.	*Identifying What to Test by Using Test Inputs* on page 24
Test Plan	Shows a test plan and all of its test case folders and test cases.	*Creating a Test Plan* on page 28
Configuration	Shows all of the configurations and configuration attributes in the project.	*Defining the Configurations to Test* on page 35
Suite	Shows all of the items contained in a suite.	*Implementing Tests as Suites* on page 67
Monitoring	Shows up-to-date information as a test case, test script, or suite runs.	*Monitoring Suites* on page 103
Test Log	Shows test logs created after you run a suite, test case, or test script.	*About Test Logs* on page 127
Reports	Shows the results of running reports.	*Reporting Results* on page 145

Planning Tests 2

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

- About test planning
- Identifying 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.

Test planning happens over time. You continually add things to the test plan. Different members of the team — such as product managers, analysts, testers, and developers — might 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, you don't just create a test plan at the beginning of the process and then view it as a stagnant object. A test plan is an evolving asset that is defined iteratively.

In TestManager, a test plan contains test cases. The test cases are 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 or test plans
- 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

Identifying 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 can help you determine what you need to test. For example, you can look at:

- Prototypes
- Builds of the software
- Functional specifications
- Requirements
- Visual models
- Source code files
- Change requests

You as a tester might look at all of these materials to help you decide, "What do I need 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.

After you build this checklist, you can create test cases. The *test cases* define what you are going to test, based on the test inputs. 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 45.

You can also run reports to identify the test inputs that have test cases and implementations associated with them, and to identify which of those test cases have been run. For example, analysts might be interested in reports based on requirements. Architects might be interested in reports based on model elements. For information about reports, see *Reporting Results* on page 145.

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 has two built-in test input types:

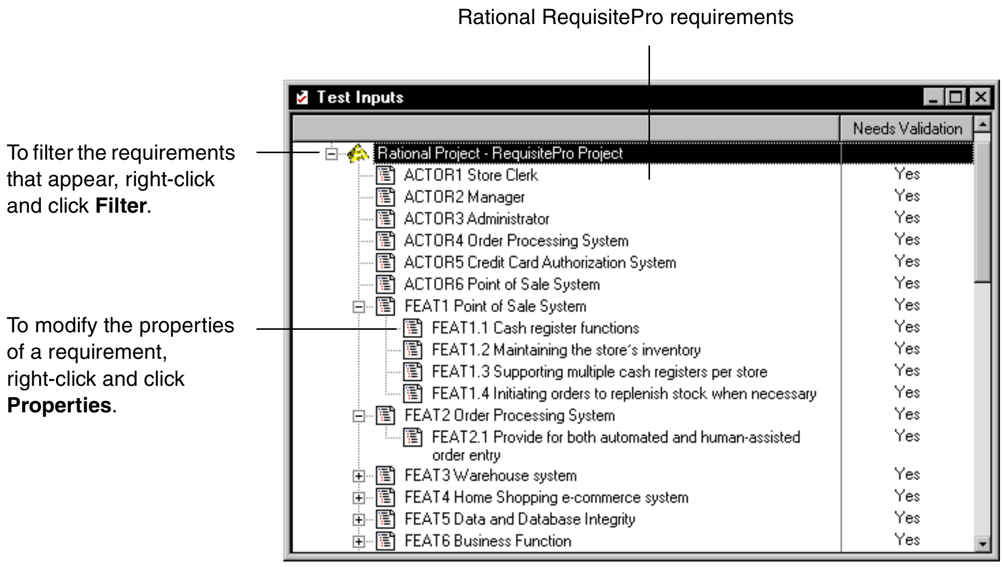
- Requirements in a Rational RequisitePro project
- Elements in a Rational Rose visual model

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. When this is done, the requirements appear automatically in the Test Input window after you log on to that project in TestManager. You can then create an association between a requirement and a test case. You can also use requirements in other RequisitePro projects as test inputs.

Note: The requirements themselves are created and managed in RequisitePro, but you can modify the properties of the requirements from TestManager.

The following figure shows requirements from a RequisitePro project. When you open the Test Inputs window, you can see the requirements.



Rational RequisitePro requirements

To filter the requirements that appear, right-click and click **Filter**.

To modify the properties of a requirement, right-click and click **Properties**.

Test Inputs

	Needs Validation
Rational Project - RequisitePro Project	
ACTOR1 Store Clerk	Yes
ACTOR2 Manager	Yes
ACTOR3 Administrator	Yes
ACTOR4 Order Processing System	Yes
ACTOR5 Credit Card Authorization System	Yes
ACTOR6 Point of Sale System	Yes
FEAT1 Point of Sale System	Yes
FEAT1.1 Cash register functions	Yes
FEAT1.2 Maintaining the store's inventory	Yes
FEAT1.3 Supporting multiple cash registers per store	Yes
FEAT1.4 Initiating orders to replenish stock when necessary	Yes
FEAT2 Order Processing System	Yes
FEAT2.1 Provide for both automated and human-assisted order entry	Yes
FEAT3 Warehouse system	Yes
FEAT4 Home Shopping e-commerce system	Yes
FEAT5 Data and Database Integrity	Yes
FEAT6 Business Function	Yes

You can use one or more RequisitePro projects as test input sources even if they are not associated with a Rational project. However, in that case, you must register the RequisitePro project with TestManager.

To register a RequisitePro project as a test input source:

1 Click **Tools > Manage > Test Input Types**.

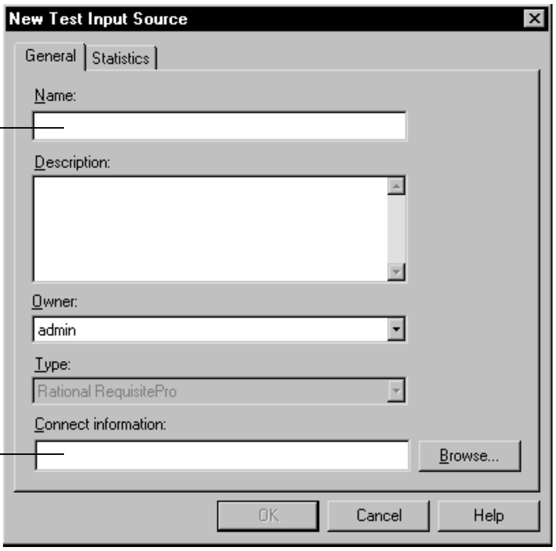
2 Click **Rational RequisitePro** and click **Edit**.

Note: If the **Edit** button is disabled, you do not have Administrator privileges. For information, see the *Using the Rational Administrator* manual or Help.

3 Click the **Sources** tab and click **Insert**.

Type a name for the RequisitePro source. This can be any name up to 40 characters.

Browse to or type the path to the RequisitePro project. For example: c:\Demo\ClassicsOnline.rqs



Model Elements from Rational Rose

If you have Rational Rose installed and licensed, you can use Rose model elements as test inputs. To use Rose model elements, you must register the source of the model with TestManager. After you do this, you can view each model element in the Test Inputs window, and you can create an association between a model element and a test case.

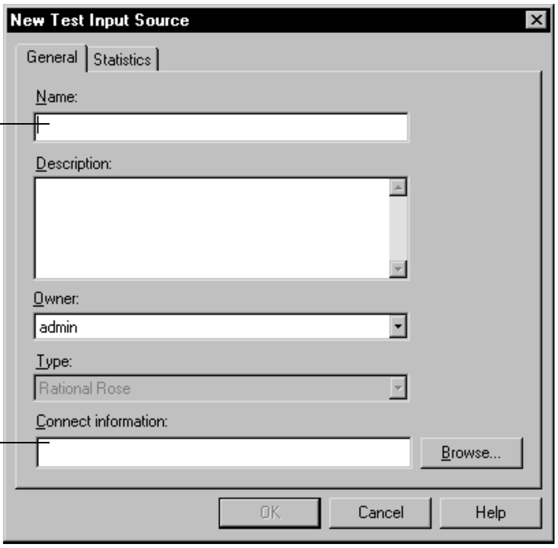
To register a Rose model:

1 Click **Tools > Manage > Test Input Types**.

2 Click **Rational Rose** and click **Edit**.

Note: If the **Edit** button is disabled, you do not have Administrator privileges. For information, see the *Using the Rational Administrator* manual or Help.

3 Click the **Sources** tab and click **Insert**.



Type a name for the Rose model source. This can be any name up to 40 characters.

Browse to or type the path to the \Rose model. For example: c:\Demo\Classics Online\learning.mdl

Custom Test Input Types

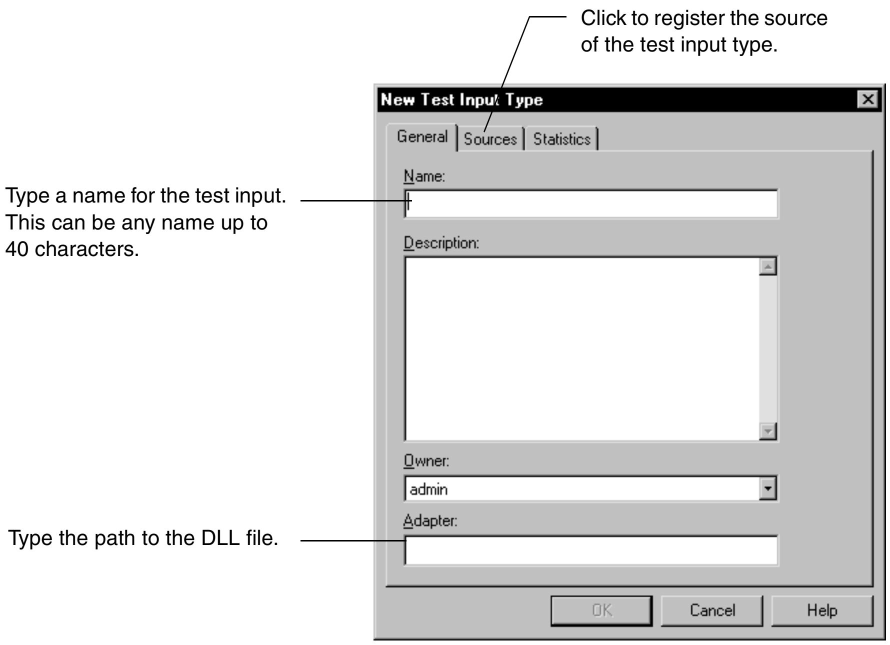
TestManager supports using test input types other than RequisitePro requirements or Rational Rose model elements. For example, you might want to use the values in a Microsoft Excel spreadsheet as test inputs. You could also define C++ language project files as a test input type if you wanted to know which tests needed to be changed or rerun when a source file changes.

For TestManager to support an extensible test input type, someone in your organization must write a custom Test Input Adapter. An adapter is a dynamic-link library (DLL) with certain required functions for TestManager to call when necessary — for example, when connecting to or disconnecting from the test input source. In addition to adapters written in your organization, some custom adapters are available from Rational Software or its partners. (For information about writing test input adapters, see the *Rational TestManager Extensibility Reference* manual.)

After the DLL is implemented, you need to define the new test input type in TestManager and register the source. To do this:

- 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 a name for the test input. This can be any name up to 40 characters.
Type the path to the DLL file.
New Test Input Type
General
Sources
Statistics
Name:
Description:
Owner:
admin
Adapter:
OK
Cancel
Help

For more information, see *test input types:registering new* in the TestManager Help Index.

After you define a new test input type and register the source, that source appears in the Test Inputs window (**View > Test Inputs**).

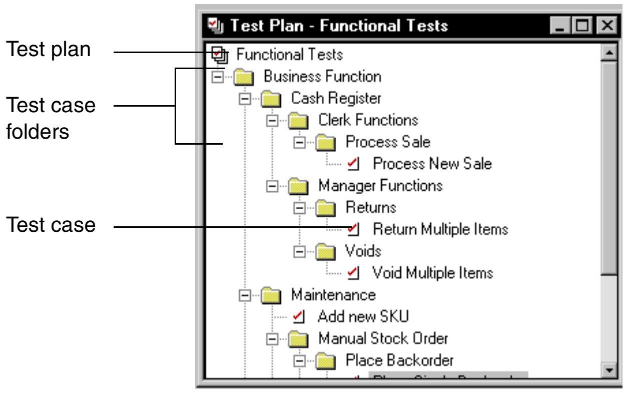
Creating a Test Plan

In TestManager, a test plan is an asset of a Rational Test datastore. You can have one or more test plans in a project, and you can organize them in any way that makes sense for your testing situation. 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. Each test plan can contain multiple test case folders and test cases.

You work with test plans in the Test Plan window. To open the Test Plan window:

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

In the following example, the test plan named Functional Tests contains multiple test case folders and test cases:



Test Plan - Functional Tests

```
Functional Tests
  Business Function
    Cash Register
      Clerk Functions
        Process Sale
          Process New Sale
      Manager Functions
        Returns
          Return Multiple Items
        Voids
          Void Multiple Items
    Maintenance
      Add new SKU
      Manual Stock Order
        Place Backorder
```

You can run reports to view information about the test plans in a project. For information about reports, see *Reporting Results* on page 145.

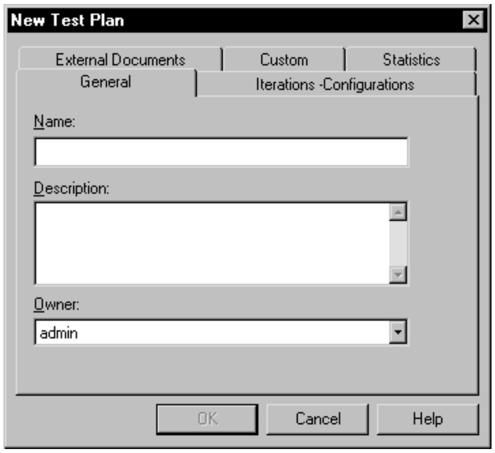
Creating Test Plans

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 create a new test plan:

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

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



Properties of a Test Plan

A test plan has many properties. These can include:

- 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 43.
- Any external documents associated with the test plan. For example, you could associate a Microsoft Word document that has detailed information about your plan.

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.

Note: You can customize the properties of some test assets from the **Tools > Customize** menu.

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.

To change the properties of a test plan:

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

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

Organizing Test Cases with Folders

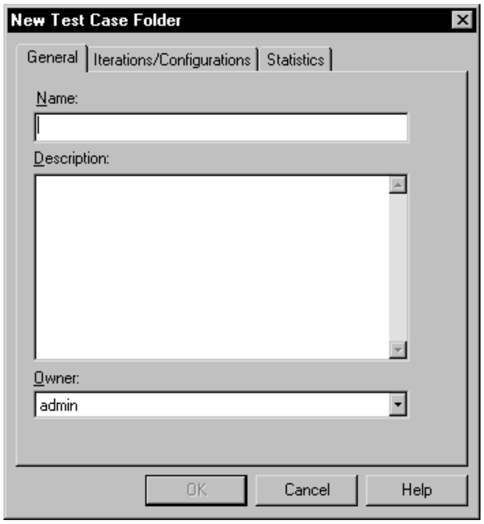
Within a test plan, you can create *test case folder*s to organize your test cases hierarchically. You can organize your test cases in any way that makes sense for your testing effort. For example, you might have a test case folder:

- For each tester in your department.
- For each category or type of test (unit, functional, performance, and so on).
- 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 that needs to be tested.

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. These can include:

- 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 configurations associated with the folder. For information, see *Defining the Configurations to Test* on page 35.
- The iterations associated with the folder. For information, see *Specifying When to Run Tests* on page 43.

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.

Creating Test Cases

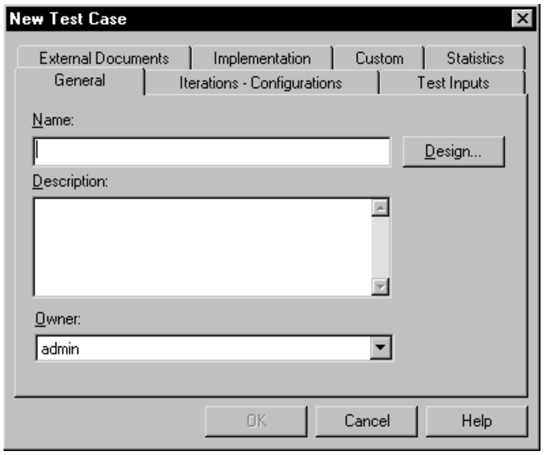
The test plan is centered on test cases. After you identify your test inputs and decide what you plan to test, you can create your test cases.

The *test case* is the test asset in TestManager that answers the question, “What am I going to test?” You develop test cases to validate that the system is working the way that it's supposed to work and is built with the quality necessary before you can ship it.

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

You can create a test case in two ways:

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



Note: You can run several types of Test Case reports to gather information about the test cases in your project. For information about reports, see *Reporting Results* on page 145.

Properties of a Test Case

A test case has many properties. These can include:

- 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 43.
- Any test inputs associated with the test case. For information, see *Setting up Traceability Using Test Inputs* on page 45.
- 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 55.
- The design of the test case (in other words, the steps and verifications to be performed when the test case is implemented). For information, see *Specifying the Testing Steps and Verification Points* on page 49.
- Preconditions, post-conditions, and the acceptance criteria of the test case. For information, see *Specifying Conditions and Acceptance Criteria of Test Cases* on page 50.

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.

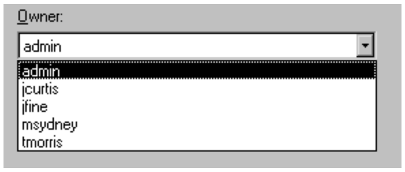
Note: You can customize the properties of some test assets from the **Tools > Customize** menu.

To change the properties of a test case:

- In the Test Plan window, right-click the test case and click **Properties**.

Specifying the Owner

You can select 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. (For information, see the *Using Rational Administrator* manual or Help.)

The owner is important for planning and 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 145.

Defining the Configurations to Test

You can use *configurations* to set up test cases so that they run automatically on computers with specific hardware or software. For example, you might need to make sure that a test case runs successfully 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.

For example, a test case might need to run successfully on the following combinations 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 combinations is a configuration that you need to test. After you define the configurations in TestManager, you can associate the configurations with test cases to create *configured test cases*. You can then run the configured test cases on the appropriate computers.

There are four main steps when setting up configurations:

1. For any attributes that are not already built in to TestManager, define the custom attributes and their possible values. (See *Defining Configuration Attributes and their Values* on page 36.)

   For example, *Browser* is not a built-in configuration attribute. You would create a configuration attribute named *Browser*, with values of *Internet Explorer 4* and *Netscape 4*.

2. Create a file named tmsconfig.csv for each computer on which you intend to run a configured test case. This file contains the custom attributes and the appropriate values for that computer. (For information, see *Setting Up Custom Attributes in tmsconfig.csv* on page 38.)

   For example, if a computer has Internet Explorer 4, you must create a tmsconfig.csv file on that computer to indicate the browser that is on it.

3 Define the specific configurations that you need to test. (See *Defining the Configurations You Need to Test* on page 39.)

   For example, *Win2000 - IE4* is one configuration, and *Win2000 - Netscape4* is another configuration.

4 Associate each configuration with a test case to create a configured test case. (See *Associating a Configuration with a Test Case* on page 42.)

   For example, if your test case needs to run on Windows 2000 and Internet Explorer 4, you would associate that configuration with the test case.

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 your testing strategy, think about how to combine the pertinent configuration attributes (for example, *Browser* and *Operating System*) to define the configurations to test against (for example, *Browser = Netscape 4* and *Operating System = Windows*).

Keep in mind that you can run reports against these configurations after you run your test cases. For example, you can create and run a Test Case Results Distribution report that distributes the results over the configurations. 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 Internet Explorer 4 and Netscape 4. Later in the project, the analyst may decide that you also need to test Internet Explorer 5. You can open the configuration attribute named *Browser* and add *Internet Explorer 5* as a new value.

Viewing Built-in Configuration Attributes

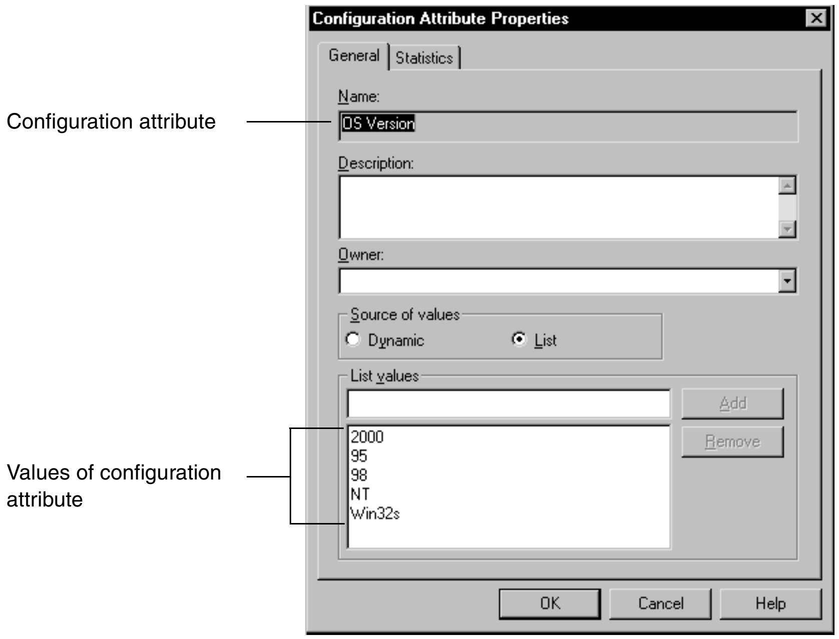
TestManager comes with many built-in attributes. These include: display colors, display resolution, memory size, operating system, OS service pack, OS version, processor MHz, processor number, and processor type.

To view these built-in attributes and their values:

1 Click **Tools > Manage > Configuration Attributes**.

2 To see the properties of each attribute, select each attribute and click **Edit**.

Any defined values appear in the List values field, as shown in the following example:



Name: OS Version

Source of values: Dynamic [✓ List]

List values:
2000
95
98
NT
Win32s

If an attribute has a value in the list, you can select that value when you create a configuration. When you run a configured test case, TestManager examines the computer and determines if there is a match with the value.

Note: If the source of values for the attribute is set to **Dynamic**, you can type a value when you create a configuration. To determine the appropriate value for a computer, run a test script on that computer and view the configuration of the computer in the Test Log window. For information, see *Viewing Events Details* on page 133.

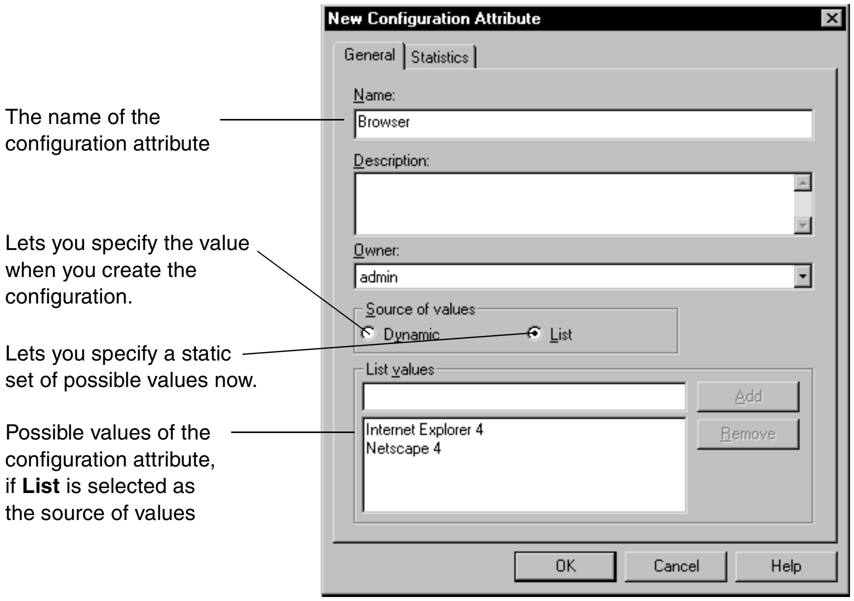
Defining Custom Configuration Attributes

To use a configuration attribute that is not already built in, you define a custom attribute.

To define a custom configuration attribute:

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

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


The name of the configuration attribute
Lets you specify the value when you create the configuration.
Lets you specify a static set of possible values now.
Possible values of the configuration attribute, if **List** is selected as the source of values
New Configuration Attribute
General
Statistics
Add
Remove
OK
Cancel
Help

Name: Browser

Description:

Owner: admin

Source of values: Dynamic [✓ List]

List values:
Internet Explorer 4
Netscape 4

If you use a custom attribute in a configuration, you need to create a file named tmsconfig.csv on each computer on which you intend to run that configuration. For information, see the next section, *Setting Up Custom Attributes in tmsconfig.csv*.

Setting Up Custom Attributes in tmsconfig.csv

As indicated in the previous section, if you have defined custom attributes and values, you need to create a tmsconfig.csv file on each computer on which you intend to run a configured test case. This file contains the attributes and values appropriate to that computer. When you run a configured test case, TestManager checks this file and runs the test case only if both the custom and the built-in attributes match the configuration of the test case.

For example, suppose that you have a configured test case that should run only on computers that have Internet Explorer 4 installed. You have defined the custom attribute and its values by clicking **Tools > Manage > Configuration Attributes**, and then clicking the **New** button, as described in the previous section. You now need to create a tmsconfig.csv file that includes the attribute and its value on that computer.

To set up custom attributes and values on a computer:

1 Create a file named *tmsconfig.csv*. You can create this file through Excel or through any text editor. (Be sure to save the file in the csv format.)

2 Add the appropriate attribute/value pairs to the file.

  In this example, the computer is running Internet Explorer 4. Therefore, the configuration file contains the following row:

  ```
  Browser,Internet Explorer 4
  ```

  The case of attribute/value pairs in tmsconfig.csv must match the case of the custom attributes and values that you defined in TestManager.

3 Save the file as `tmsconfig.csv`.

4 Move the tmsconfig.csv file to the Rational Test folder on the appropriate Local or Agent computer.

If a test case's configuration uses custom attributes, the configured test case will run only on computers that fully match those attributes as defined in the computer's tmsconfig.csv file.

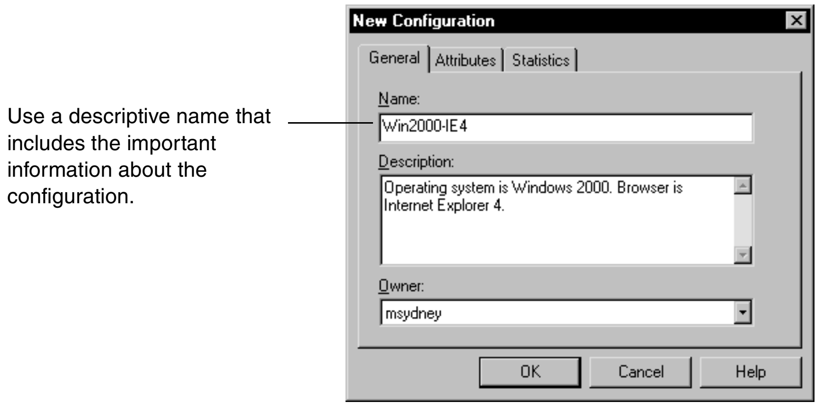
Defining the Configurations You Need to Test

Now that you have defined the configuration attributes and their values, you can 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 define a configuration:

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

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



Use a descriptive name that includes the important information about the configuration.

New Configuration

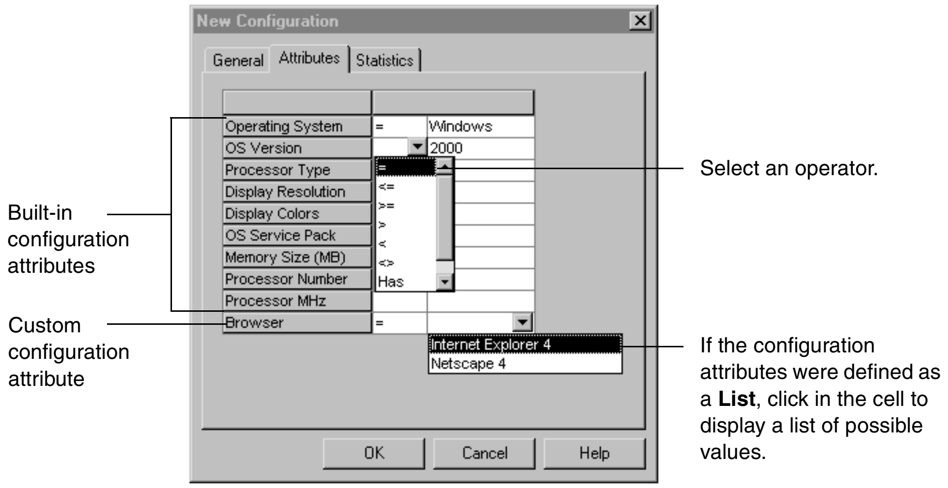
Name: Win2000-IE4

Description: Operating system is Windows 2000. Browser is Internet Explorer 4.

Owner: msydney

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 computer that has a combination of Windows 2000 and Internet Explorer 4.

2 Click the **Attributes** tab.



Built-in configuration attributes

Custom configuration attribute

Select an operator.

If the configuration attributes were defined as a **List**, click in the cell to display a list of possible values.

New Configuration

Operating System	=	Windows
OS Version		2000
Processor Type	=	
Display Resolution		
Display Colors		
OS Service Pack		
Memory Size (MB)		
Processor Number		
Processor MHz		
Browser	=	

In this tab, you can:

- Select a value for each appropriate built-in configuration attribute (in this example, Operating System and OS Version).
- Select a value for the custom configuration attribute (in this example, Browser) from the list of values you defined when you created the configuration attribute.

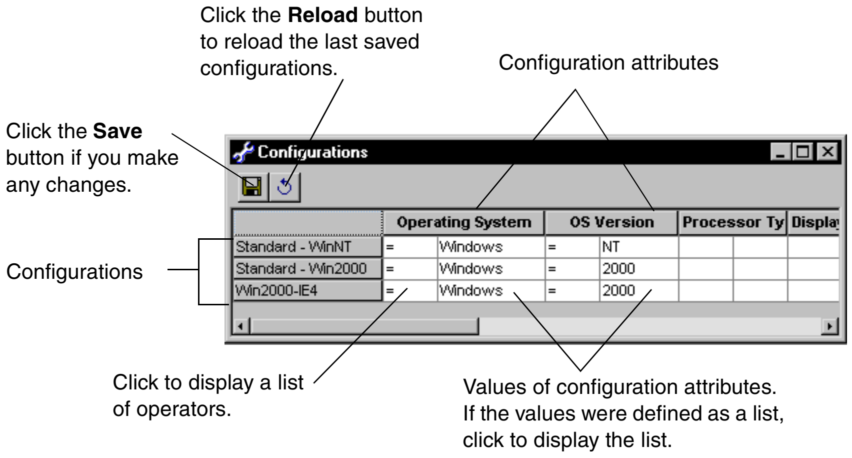
Note: If the configuration attributes were defined as dynamic, you would need to type in a value. For custom attributes, the valid value for a computer should be in the tmsconfig.csv file on that computer. For built-in attributes, to determine the appropriate value for a computer, run any test script on that computer and view the configuration of the computer in the Log Event window of the test log. For information, see *Viewing Events Details* on page 133.

Viewing and Editing Your Configurations

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

To open the Configurations window:

- Click **View > Configurations**.


Click the **Reload** button to reload the last saved configurations.
Configuration attributes
Click the **Save** button if you make any changes.
Configurations
Click to display a list of operators.
Values of configuration attributes. If the values were defined as a list, click to display the list.

	Operating System		OS Version		Processor Ty		Displa
Standard - WinNT	=	Windows	=	NT			
Standard - Win2000	=	Windows	=	2000			
Win2000-IE4	=	Windows	=	2000			

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.

Associating a Configuration with a Test Case

After you've created configurations, you can associate a configuration with a test case to create a *configured test case.*

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 two configurations: Windows 2000 with Internet Explorer 4, and Windows 2000 with Netscape 4. You could create two configured test cases associated with the main test case. In order for the test case to pass, all of its configured test cases need to pass.

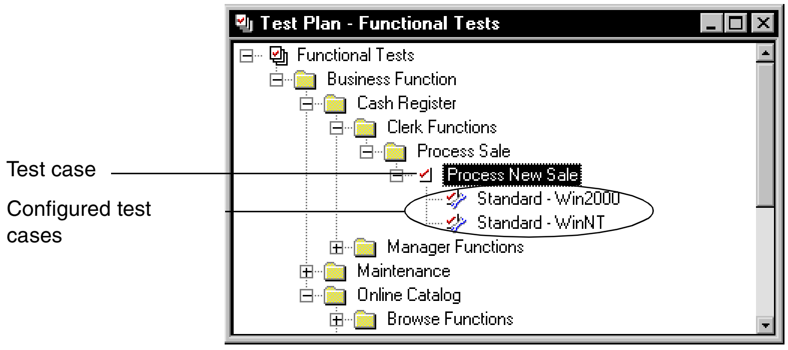
After you run the configured test cases, you can create a Test Case Results Distribution report filtered on the specific configurations you are interested in. For information about reports, see *Reporting Results* on page 145.

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.
- When editing the properties of an existing test case, click the **Iterations - Configurations** tab in the Test Case Properties dialog box.
- In the Test Plan window, right-click a test case and click **Associate Configuration**. Select the configurations to associate.

You can also associate configurations with test plans and test case folders. When you associate a configuration with a test plan or folder, the configuration is automatically associated with all new test assets that are direct children of that plan or folder. When you associate a configuration in the Test Plan window, you can also associate the configuration with all of the existing children of the test plan or folder.

Configured test cases appear under test cases in the Test Plan window.



Specifying When to Run Tests

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

An *iteration* is a defined span of time during a project. The end of an iteration is typically a major project milestone. In an iteration, 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. In many organizations, the tester works with an analyst or project manager 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 or the analyst go 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 TestManager Help and 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

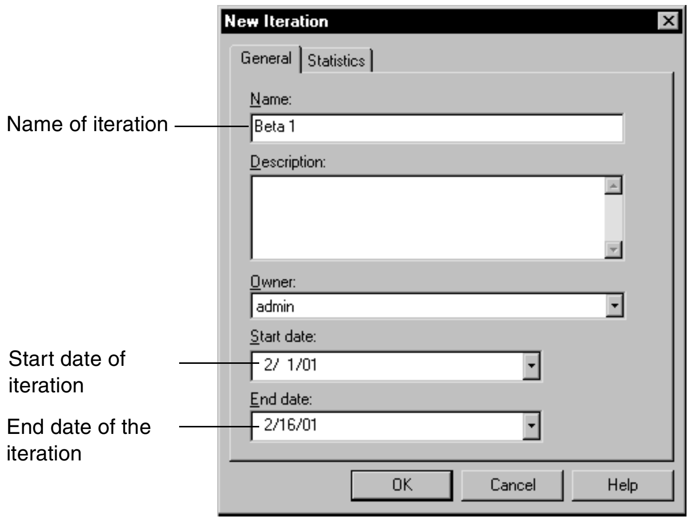
To create or edit iterations:

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. For information, see the *Using the Rational Administrator* manual or Help.

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



Name: Beta 1
Description:
Owner: admin
Start date: 2/ 1/01
End date: 2/16/01

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.
- When editing the properties of an existing test case, click the **Iterations - Configurations** tab in the Test Case Properties dialog box.
- In the Test Plan window, right-click a test case and click **Associate Iteration**.

You can also associate iterations with test plans and test case folders. When you associate an iteration with a test plan or folder, the iteration is automatically associated with all new test assets that are direct children of that plan or folder. When you associate an iteration in the Test Plan window, you can also associate the iteration with all of the existing children of the test plan or folder.

You can run all test cases associated with a specific iteration. For information, see *Running a Test Case* on page 91.

You can create and run Test Case reports that distribute over iterations. For example you can create a Test Case Result Distribution report that is distributed over 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 objective for that milestone.

Setting up Traceability Using Test Inputs

As described in *Identifying What to Test by Using Test Inputs* on page 24, 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.

You can also use associations to 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. You can use the Test Case reports to determine this information.

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

- When creating a new test case, click the **Test Inputs** tab in the New Test Case dialog box.
- When editing the properties of an existing test case, click the **Test Inputs** tab in the Test Case Properties dialog box.
- In the Test Plan window, right-click a test case and click **Associate Test Input**.
- In the Test Inputs window, right-click a test input and click **Associate Test Case**.

Designing Tests 3

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

- About designing tests
- Specifying the testing steps and verification points
- Specifying preconditions, 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 perform this test case?"

As part of the design of the test case, you can identify:

- The basic set of steps required to perform the test.
- How to validate that the items or features you are testing are working properly.
- The preconditions 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 an automated 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 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.

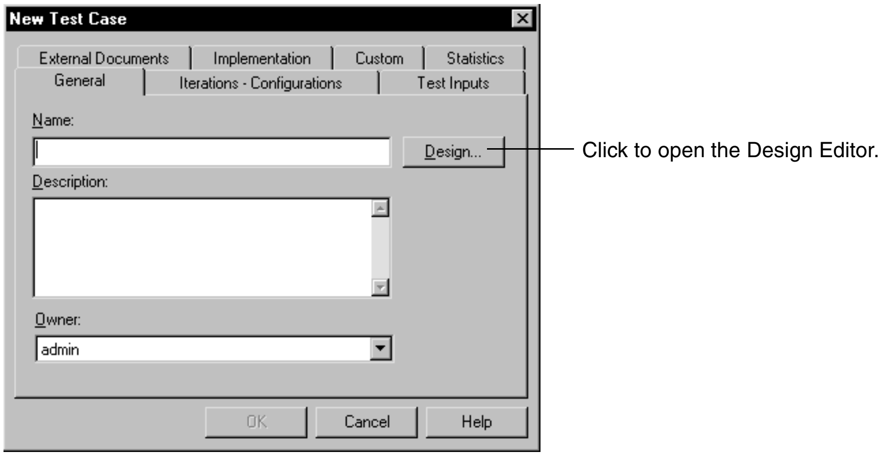
You can easily import a test design into a manual test script, which then becomes the implementation of the test case. For information about manual test scripts, see *Creating Manual Test Scripts* on page 61.

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**.



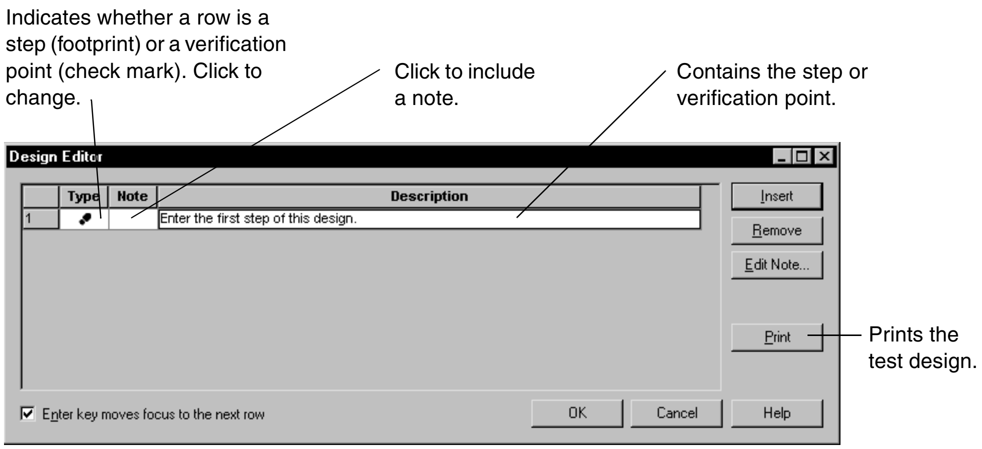
Name:

Description:

Owner: admin

To design an existing test case:

- In the Test Plan window, right-click the test case. Click **Design**.


Indicates whether a row is a step (footprint) or a verification point (check mark). Click to change.
Click to include a note.
Contains the step or verification point.
Design Editor
Type
Note
Description
Insert
Remove
Edit Note...
Print
OK
Cancel
Help
Prints the test design.

1 Enter the first step of this design.

☑ Enter key moves focus to the next row

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

Step – An action to be taken in the application or system. This could be general when you first begin the design, and then become more specific over time.

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.

Note: To create a manual test script from a test case design: Click the **Implementation** tab, and then click the **Import from Test Case Design** button. For more information, see *Creating Manual Test Scripts* on page 61.

Specifying Conditions and Acceptance Criteria of Test Cases

Preconditions 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 ends, therefore ensuring that the test case can run properly and that it leaves the system in an appropriate state. Failure of a precondition 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:

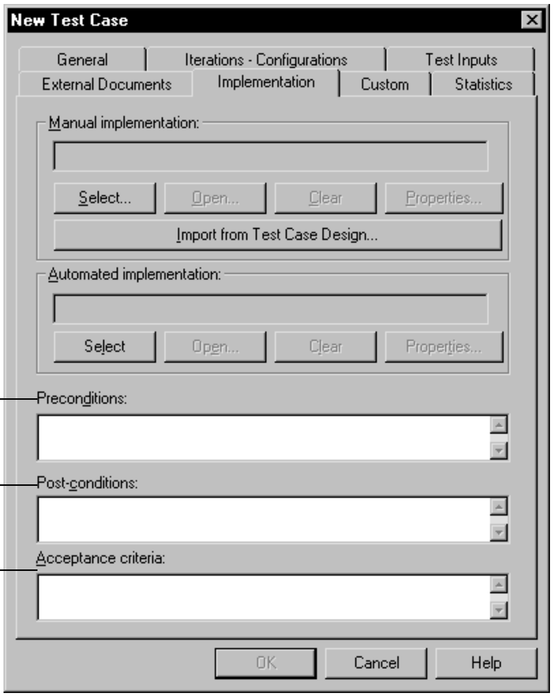
1 In the Test Plan window, right-click a test case. Click **Properties**.

2 Click the **Implementation** tab.

Any setup dependency that is required for the test case to run.

Any cleanup steps after the test case is run, to bring it back to a known state.

The expected results or performance characteristics that define whether the test case passed or failed.



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:

Precondition – 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:

Preconditions

- 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 test assets 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.

Implementing Tests 4

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

- About implementing tests
- Implementing test cases
- Calling Test Script Services from test scripts
- 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 or manual test efforts 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 65.

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 87.

Implementing Test Cases

You can implement a test case by creating a test script or a suite.

When you create a test script, you can use a Rational test implementation tool to create a built-in type of test script, or you can create a custom type of test script.

Built-in Test Scripts Types

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

Automated Test Scripts Recorded in Rational Robot

TestManager comes with built-in support for implementing the following types of test scripts in Rational Robot (if Rational Robot is installed):

- **GUI** – A test script written in SQABasic, a Rational proprietary Basic-like scripting language. GUI test 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. (When you start to record a VU test script, you actually record a session. You can generate VU or VB test scripts from the recorded session, depending on a recording option that you select in Robot.)

To record a test script in Robot, starting from TestManager:

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

  This starts Robot and opens the Record dialog box.

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

Manual Test Scripts Created in Rational ManualTest

TestManager comes with built-in support for implementing manual test scripts in Rational ManualTest. A *manual test script* contains a set of testing instructions to be run by a human tester.

For information, see *Creating Manual Test Scripts* on page 61.

Custom Test Script Types

TestManager's extensible test script type functionality enables you to implement test scripts using any tool that is appropriate for your testing environment.

Command Line and Custom Adapters

There are two ways to extend TestManager to support a new test script type:

- Use the command line adapters
- Build custom adapters or use adapters provided by Rational and its partners

Command Line Adapters

TestManager provides two command line adapters:

- **Command Line Test Script Console Adapter** – Use when the test script's test tool or editor provides a command-line interface for creating and editing the test script, and when the test script can be opened with the standard File Open dialog box (for example, Perl scripts).
- **Command Line Test Script Execution Adapter** – Use when the test script's test tool provides a command-line interface for running the test script.

The advantage of using these adapters is that they require no custom programming. You just need to define the new test script type in TestManager. For information, see *Defining a New Test Script Type* on page 58.

Custom Adapters

Instead of using the command line adapters, you can build your own custom adapters or use adapters provided by Rational Software and its partners. An adapter is a dynamic-link library (DLL).

There are two types of custom adapters:

- **Custom Test Script Console Adapter** – Required for test scripts that are created with a test tool that does not provide a command-line interface for creating and editing test scripts, and for test scripts that cannot be opened with the standard File Open dialog box.
- **Custom Test Script Execution Adapter** – Required if the test script's test tool does not provide a command-line interface for running the test script.

For information about creating custom adapters, see the *Rational TestManager Extensibility Reference* manual.

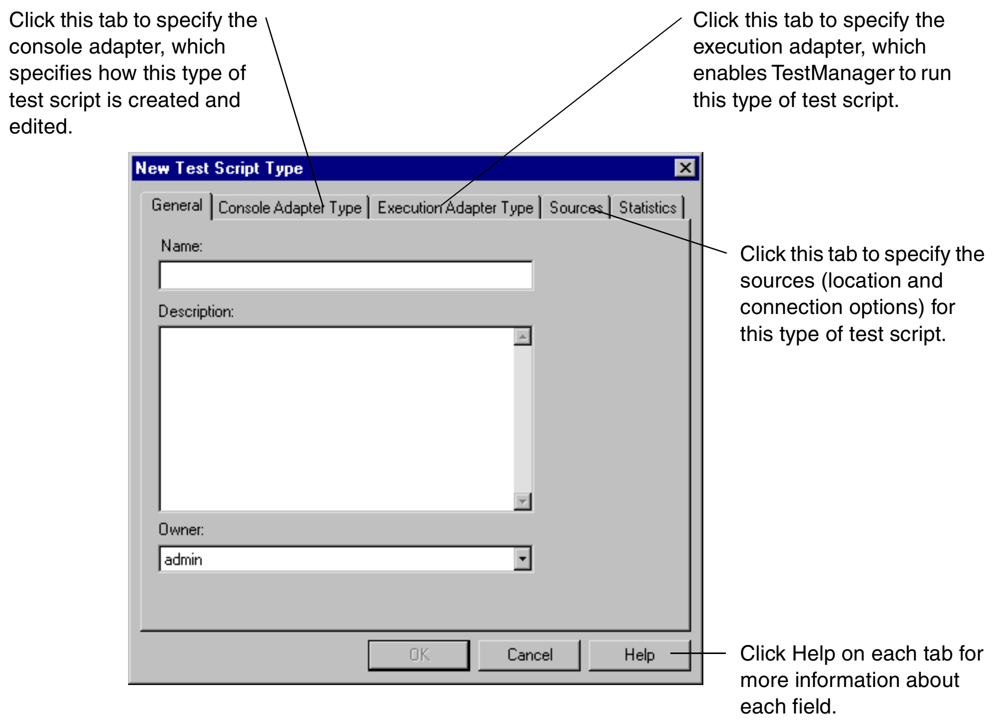
After these adapters have been created, you or someone in your organization must define the new test script type in TestManager and register the DLL. (For information, see the next section, *Defining a New Test Script Type*.) You should then be able to open and run a test script of that type from TestManager. Depending on how the Test Script Console Adapter was implemented, you might also be able to create and edit test scripts of that type.

Defining a New Test Script Type

To define 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. For information, see the *Using the Rational Administrator* manual or Help.


Click this tab to specify the console adapter, which specifies how this type of test script is created and edited.

Click this tab to specify the execution adapter, which enables TestManager to run this type of test script.

Click this tab to specify the sources (location and connection options) for this type of test script.

Click Help on each tab for more information about each field.

Owner: admin

Suites Created in TestManager

TestManager lets you build test *suites* from test scripts, test cases, and other items. Suites provide great flexibility and power for creating functional and performance tests via a point-and-click interface.

Suite basics are covered in *Implementing Tests as Suites* on page 67. For details about functional testing suites, see *Creating Functional Testing Suites* on page 165. For details about performance testing suites, see *Designing Performance Testing Suites* on page 225.

Calling Test Script Services from Test Scripts

Rational Test Script Services (TSS) are testing services that you can call from your test scripts using the commands in the Test Script Services API. For example, you can call logging, synchronization, timing, and datapool services from test scripts. You can call verification point services to validate the state or behavior of a component or system.

The following table lists the categories of services that TSS provides:

Category	**Description**
Datapool	Provides variable data to test scripts during playback, allowing virtual testers to send different data to the server with each transaction.
Logging	Logs messages for reporting and analysis.
Measurement	Provides the means of fine-tuning and controlling your tests through operations such as timing actions, setting think time delays, and setting environment variables.
Utility	Performs common test script operations such as retrieving error information, controlling the generation of random numbers, and printing messages.
Monitor	Monitors playback progress of a test script.
Synchronization	Synchronizes multiple virtual testers running on a single computer or across multiple computers.
Session	Manages test script session execution and playback.
Advanced	Advanced features, such as setting values for internal variables.
Verification Point	Validates the state or behavior of a component or system.

Test Script Services and Test Script Types

VB, Java, and command line test script types — as well as custom test script types that you might add to TestManager — can take advantage of Test Script Services.

You can add TSS commands to test scripts manually during test script editing.

Test Script Services commands can be added to test scripts automatically during the following operations:

- Recording VB test scripts with Rational Robot.
- Recording with the EJB Session Recorder that is included with Rational QualityArchitect. The Session Recorder lets you visually connect to and interact with EJBs. As you execute transactions against the component, interaction data is recorded and stored in an external XML file. Afterwards, you can use the XML Script Generator to generate Java test scripts for testing the EJB.
- Generating test scripts using Rational QualityArchitect. QualityArchitect generates Java test scripts for testing EJB components and Visual Basic test scripts for testing COM/DCOM objects.

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.

Type of Test Script	Method of Adding Test Script Services Commands	Documentation
VB	Recording a session and generating the test script with Rational Robot or manually editing	*Rational Test Script Services for Visual Basic*
Java	Manual editing	*Rational Test Script Services for Java*
Command Line	Manual editing	*Command Line Interface to Rational Test Script Services*

Note: Robot VU and GUI test script types automatically provide most TSS equivalents. For example, when you record a script with Rational Robot, the script will contain a datapool, if appropriate. TestManager also provides built-in monitoring and session functions.

Test Script Services and TestManager

Test Script Services are designed for use with TestManager. As a result, TSS features that are included in any type of test script — including custom test script types — are fully integrated with TestManager's reporting, monitoring, and analysis framework. For example:

- TestManager adheres 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 status information about your test script through the test script monitoring commands.
- During test script playback, TestManager provides realistic and variable 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 the state or behavior of a component or system.
- TestManager can run test scripts of different types within a single suite. For example, VU, VB, and test scripts of custom types can all be run within the same suite.

Creating Manual Test Scripts

A *manual test script* is a set of testing instructions to be run by a human tester. The test 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 test 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, the test script passes. If any verification points failed, the test 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.

You can create a manual test script in two ways:

- By importing a test case design
- In Rational ManualTest

Creating a Manual Test Script from a Test Case Design

As described in *About Designing Tests* on page 47, a test case design primarily answers the question "How can I perform this test case?" A design can contain steps and verification points.

You can easily create a manual test script from a test case design. The steps and verification points in the design become steps and verification points in the manual test script.

To create a manual test script from a test case design:

- In the Test Case dialog box, click the **Implementation** tab. Click the **Import from Test Case Design** button.

The manual test script becomes the implementation of the test case.

You can view and edit the manual test script by clicking the **Open** button in the **Implementation** tab. This opens Rational ManualTest. For more information, see the next section, *Creating a Manual Test Script in Rational ManualTest*.

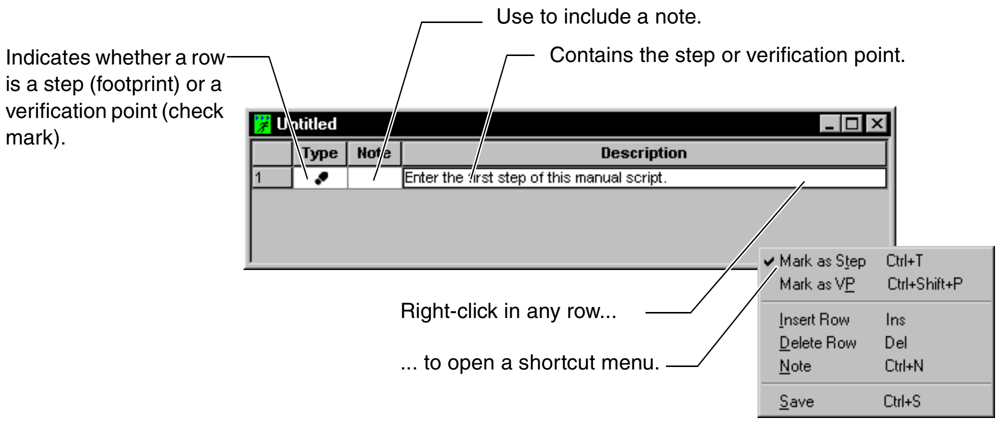
Creating a Manual Test Script in Rational ManualTest

Rational ManualTest is tightly integrated with Rational TestManager. Use Rational ManualTest to create and run manual test scripts.

Starting Rational ManualTest

To start Rational ManualTest and create a new manual test script:

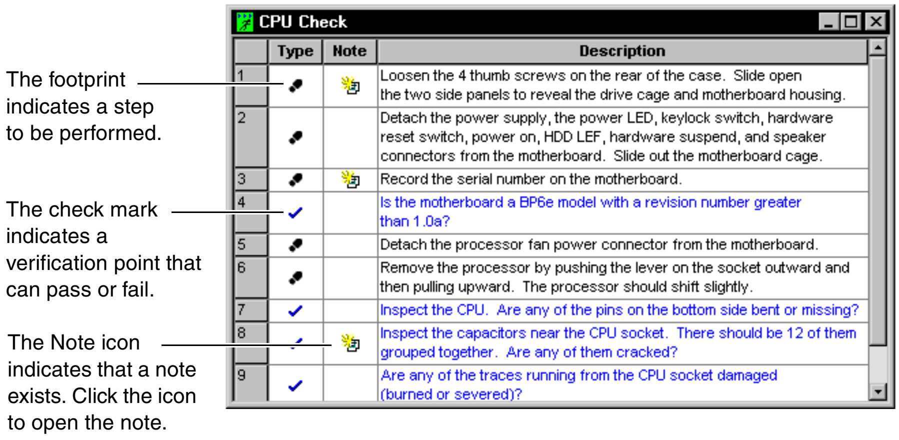
- In TestManager, click **File > New Test Script > Manual**.


Use to include a note.
Contains the step or verification point.
Indicates whether a row is a step (footprint) or a verification point (check mark).
Untitled
Type
Note
Description
1
Enter the first step of this manual script.
Mark as Step Ctrl+T
Mark as VP Ctrl+Shift+P
Insert Row Ins
Delete Row Del
Note Ctrl+N
Save Ctrl+S
Right-click in any row...
... to open a shortcut menu.

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 test 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.

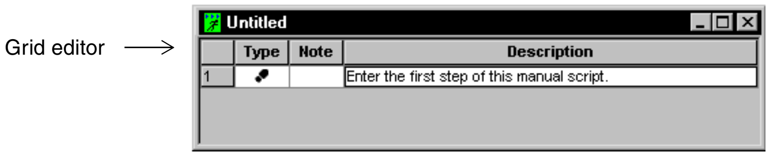
The Note icon indicates that a note exists. Click the icon to open the note.

CPU Check

	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

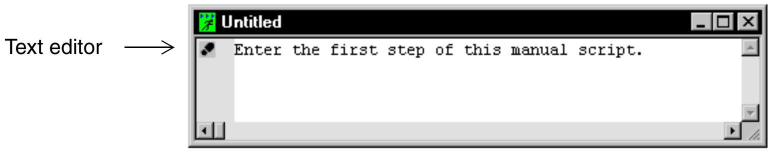
You can use either the grid editor or the text editor when you create a manual test script. The grid editor is a structured editor that makes it easy to enter your steps and verification points. The text editor is a free-form editor that makes it easy to manipulate text.



Grid editor

Untitled

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



Text editor

Untitled

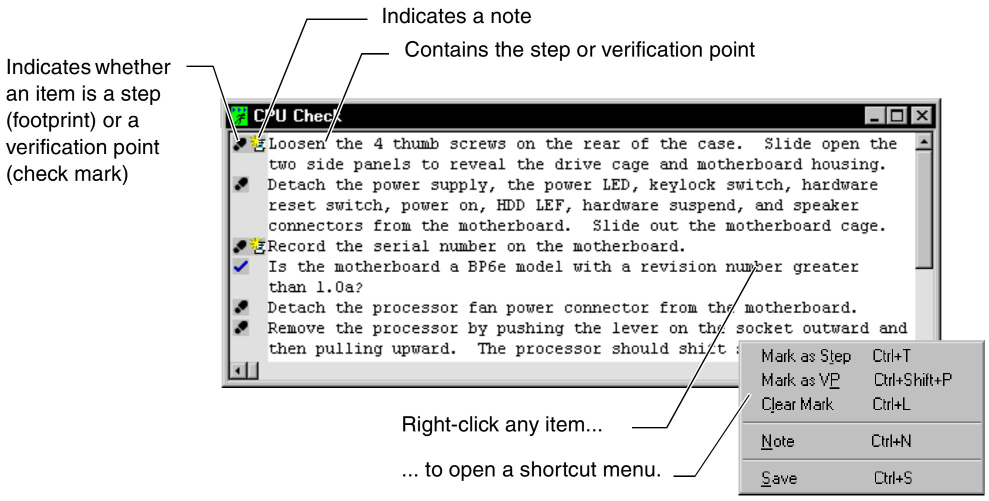
```
Enter the first step of this manual script.
```

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.

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


Indicates a note
Contains the step or verification point
Indicates whether an item is a step (footprint) or a verification point (check mark)
CPU Check
Right-click any item...
... to open a shortcut menu.
Mark as Step Ctrl+T
Mark as VP Ctrl+Shift+P
Clear Mark Ctrl+L
Note Ctrl+N
Save Ctrl+S

```
Loosen the 4 thumb screws on the rear of the case.  Slide open the
two side panels to reveal the drive cage and motherboard housing.
Detach the power supply, the power LED, keylock switch, hardware
reset switch, power on, HDD LEF, hardware suspend, and speaker
connectors from the motherboard.  Slide out the motherboard cage.
Record the serial number on the motherboard.
Is the motherboard a BP6e model with a revision number greater
than 1.0a?
Detach the processor fan power connector from the motherboard.
Remove the processor by pushing the lever on the socket outward and
then pulling upward.  The processor should shift
```

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.

Creating Test 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 test scripts to the standards and practices used within your organization.

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

- Add up to three custom properties and values. (These appear in the **Custom** tab of the New Test Script/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 test script.

To see the standard properties of a manual test script in Rational ManualTest:

- Click **File > Properties**.

To customize a manual test script in Rational ManualTest:

- 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 test script, see the ManualTest Help.

You can also view and customize the properties of any test script in TestManager.

Associating an Implementation with a Test Case

After 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:

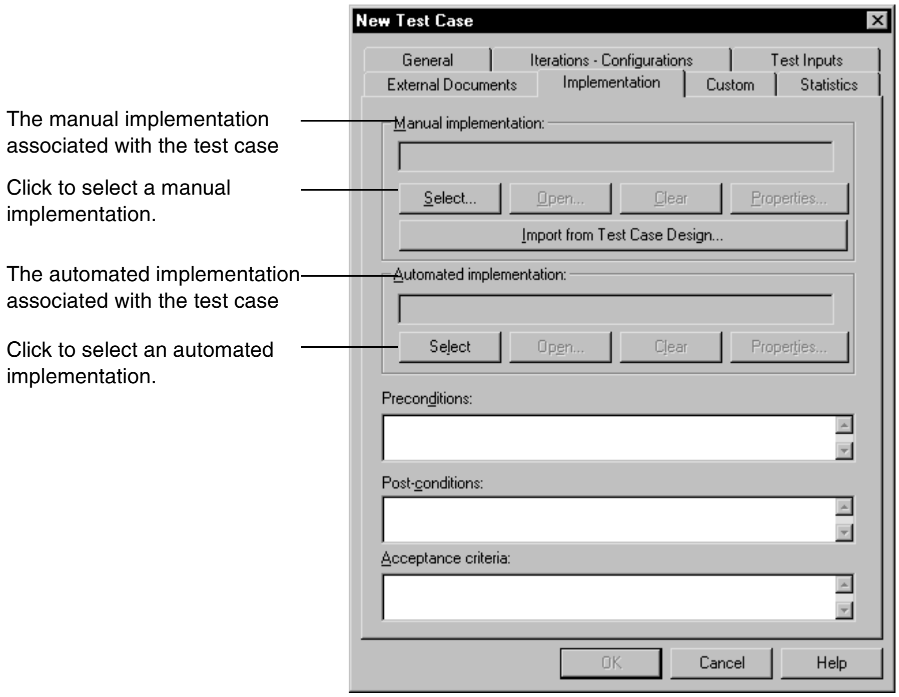
- GUI test scripts
- VU test scripts
- VB test scripts
- Java test scripts
- Command-line executable programs
- Suites
- Manual test scripts

TestManager also supports associating other test script types that you have registered. For information, see *Custom Test Script Types* on page 57.

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.

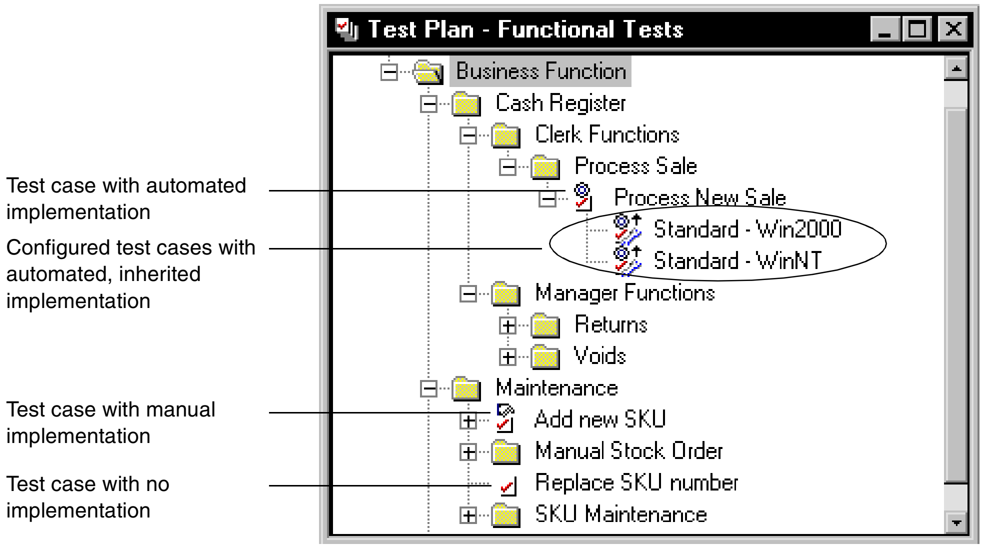

The manual implementation associated with the test case
Click to select a manual implementation.
The automated implementation associated with the test case
Click to select an automated implementation.
New Test Case
General
Iterations - Configurations
Test Inputs
External Documents
Implementation
Custom
Statistics
Manual implementation:
Select...
Open...
Clear
Properties...
Import from Test Case Design...
Automated implementation:
Select
Open...
Clear
Properties...
Preconditions:
Post-conditions:
Acceptance criteria:
OK
Cancel
Help

You can have at most two implementations associated with a test case: one manual and one automated. If both are associated with a test case, TestManager runs the automated implementation when you run the test case. For information about running test cases, see *Running Test Cases* on page 90.

Note: If you run a test case from the Rational ManualTest Web Execution component, the manual test script will run. For information, see *About ManualTest Web Execution* on page 351.

When you create a configured test case, it inherits the implementation of its parent test case. You can change the implementation of a configured test case by clicking the **Select** button in the Configured Test Case dialog box.

Icons in the Test Plan window indicate if a test case or configured test case has an implementation, and whether it is automated or manual. For a configured test case, the icons also indicate if the implementation was inherited from the parent test case. The following figure shows some of the icons. For a complete list of the icons, see the Rational TestManager Help.



Note: Even if an icon shows that the implementation is automated, the test case could also have a manual implementation.

Implementing Tests as Suites

Suites are another way to implement tests in TestManager. A *suite* shows a hierarchical representation of the tasks that you want to test, or the 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 implementing a test as a suite, you can:

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

  Groups are collections of virtual testers 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 suites to suites.

  You can use suites as building blocks within suites.

- Add test cases.

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

You can use suites in both performance and functional testing. Although the concepts of a suite are the same in both types of testing, you will probably insert different suite items, and select different options, depending on whether you are running a functional or a performance test. The next two sections give an overview about using suites in functional and performance testing.

Using Suites in Functional Tests

When you use a suite for functional testing, you start by setting up one computer group. This computer group can run test cases or test scripts on designated computers. Only one test script can run on one computer at a time. You can set up test scripts so that they run on:

- Specific computers that you assign beforehand.
- Specific computers that you assign at runtime.
- Any computer that is free to run a test script.

You may want to run a functional test on a specific computer. For example, your test may be designed for a particular computer. Or your test may require specific software, which is installed on a particular computer.

Conversely, you may want to distribute the test scripts in a functional test among several computers. For example, your tests may not be designed for a specific computer. You may want to run your tests on a group of computers, so that they can complete as fast as possible.

Implementing a test as a suite enables you to accomplish each goal. For more information about using suites in functional tests, see *Creating Functional Testing Suites* on page 165.

Using Suites in Performance Tests

In performance testing, a *suite* enables you to not only run test scripts, but to also emulate the actions of users adding workload to a server. 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.

For more information about using suites in performance testing, see *Designing Performance Testing Suites* on page 225.

Activities Common to Performance and Functional Testing

Whether you are running performance tests or functional tests, you will perform certain common activities. For example, you define computers and computer lists, create suites, open suites, edit test scripts, and edit items and properties in suites. These sections discuss the tasks that you do in both performance testing and functional testing suites.

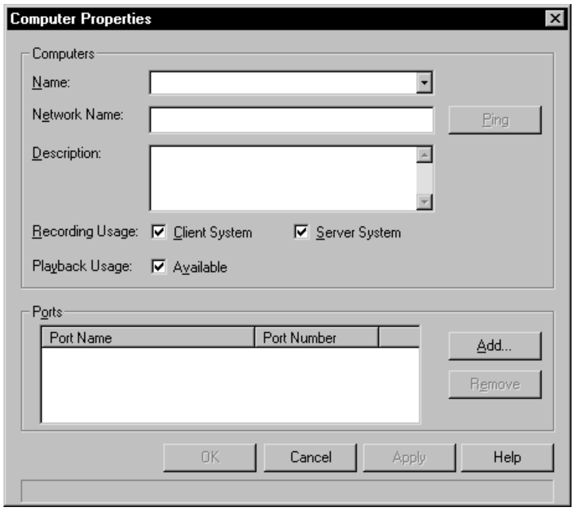
Defining Agent Computers and Computer Lists

Tests run, by default, on the Local computer. However, you can add other computers, called *Agents*, to your test environment. These Agent computers are useful in both performance and functional testing. In performance testing, you use Agents to add workload to the server. In functional testing, you use Agents to run your tests in parallel, thus saving time.

Adding an Agent Computer

To add an Agent computer:

- Click **Tools > Manage > Computers**, and then click **New**.



When you add an Agent computer to TestManager, you can include the following properties:

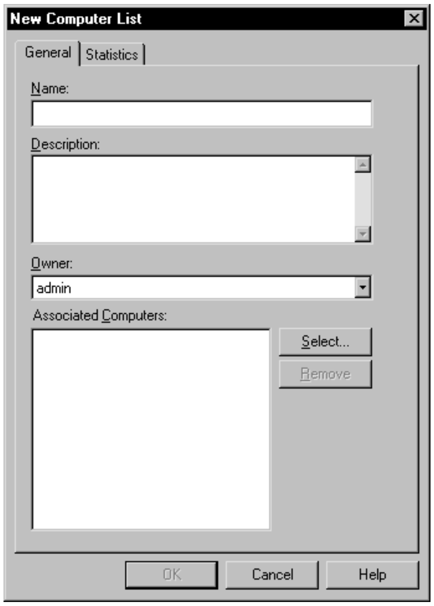
- **Name** – A name for the computer. Use this name to help you easily identify the computer if the network name is not very descriptive.
- **Network Name** – The name of the computer as recognized on the computer network. To verify the network name and confirm that the computer is available, click **Ping**.
- **Description** – A description of the computer, perhaps noting its role in the testing process.
- **Recording Usage** – Whether TestManager treats the computer as a client or a server system during recording.
- **Playback Usage** – Whether the system is available for test script playback.
- **Port Information** – TCP/IP port information associated with the system. For information about port settings, see *Configuring Local and Agent Computers* on page 331.

Combining Computers into Lists

After you have defined computers within TestManager, you can combine them into lists. Lists are useful if you run tests on several computers, or if several computers have a similar configuration or perform a similar organizational task. By using a list, you can reference the list instead of the individual computers.

To create a computer list:

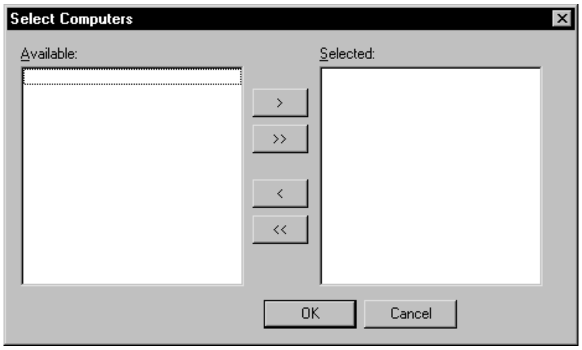
- Click **Tools > Manage > Computer Lists**, and then click **New**.



After you have defined the computer list with a name and description, add computers to the list. You must add a computer to TestManager individually before you can include it in a list.

To add computers to a computer list:

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



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

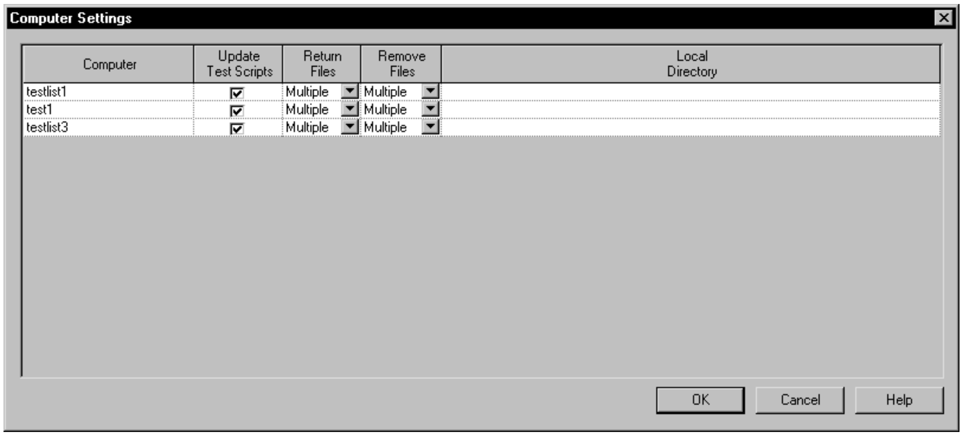
Note: Be sure to copy any custom-created external C libraries, Java class files, or COM components that the suite requires to the Agent computer.

Changing the Settings of an Agent Computer

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

To change the computer settings:

- Open a suite, and then click **Suite > Edit Computers**.



Computer	Update Test Scripts	Return Files	Remove Files	Local Directory
testlist1	✓	Multiple	Multiple	
test1	✓	Multiple	Multiple	
testlist3	✓	Multiple	Multiple	

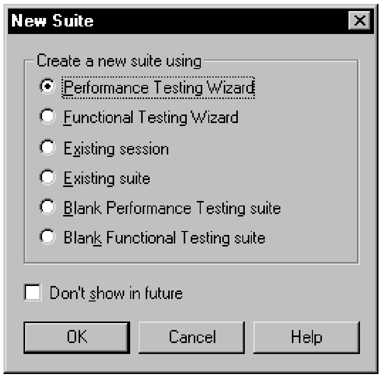
Creating a Suite

You can create a suite in several ways. You create a suite

- With wizards.
- Based on a Robot session or another suite.
- From scratch, using a blank template.

To create a suite using any of these methods:

- Click **File > New Suite**.



New Suite

Create a new suite using
- (●) Performance Testing Wizard
- ( ) Functional Testing Wizard
- ( ) Existing session
- ( ) Existing suite
- ( ) Blank Performance Testing suite
- ( ) Blank Functional Testing suite

[ ] Don't show in future

Creating a Suite from a Wizard

If you are new to testing, using the suite wizards may be the easiest way to create a working test. Each wizard guides you through the process of creating a suite. You must have test scripts or test cases available for use in the suite.

When you create a suite with the performance testing wizard, TestManager helps you choose the computer on which the test runs and helps you associate test scripts that become the basis for the test.

When you create a suite with the functional testing wizard, TestManager helps you choose test cases and test scripts that become the basis for the test.

Opening a Suite

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:

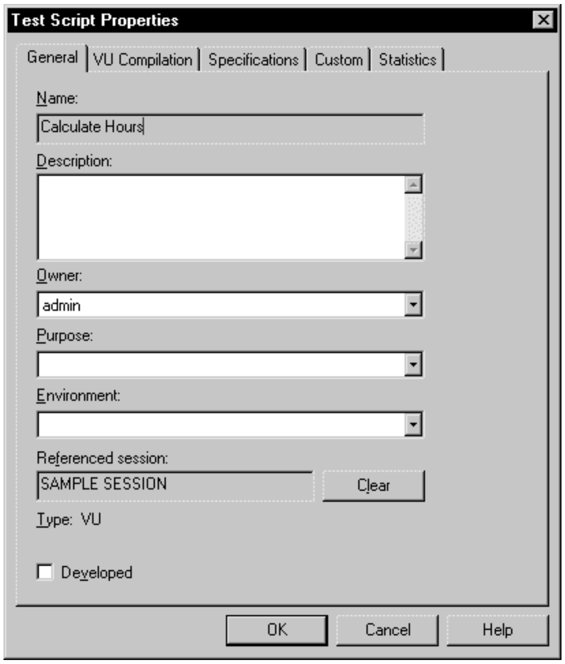
- Edit the properties of a test script.
- Edit the text of a test script.

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:

- Open a suite, select the test script to edit, and then click **Edit > Properties**.


Test Script Properties

General | VU Compilation | Specifications | Custom | Statistics

Name: Calculate Hours
Description:
Owner: admin
Purpose:
Environment:
Referenced session: SAMPLE SESSION
Type: VU
Developed: ☐

Note: The tabs in this dialog box vary slightly, depending on the type of test script you open.

Editing the Text of a Test Script

To edit the text of a test script:

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

The script is loaded and the appropriate editor is launched.

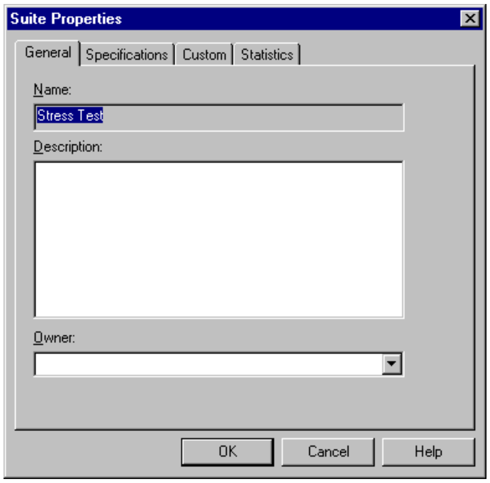
For more information about editing test scripts, see the *VU Language Reference* or the Rational Test Script Services manual for your language.

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, and then click **File > Properties**.



Name: Stress Test
Description:
Owner:

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.

To replace an item:

1 Click **Tools > Options > Create Suite**, and clear the **Show numeric values** check box. Clearing this option lets you use inline editing to rename suite items.

2 Open a suite, select the item, and then type over the new item name.

Editing the Run Properties of Items a Suite

As your testing process evolves, you may want to edit the run properties associated with suite items. You can edit the run properties of any items contained in a suite.

To edit the run properties of an item:

1 Open a suite, and select the item to edit.

2 Click **Edit > Run Properties**.

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

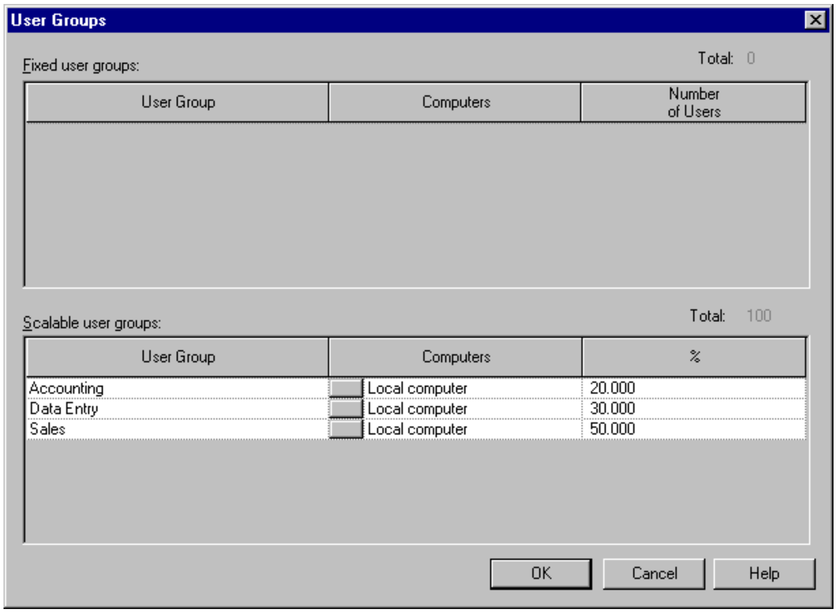
Note: When you edit the run properties of a suite item, the changes affect only that instance of the item. For example, if you insert a test script into a suite twice, and change the run properties of the first test script, the second test script still retains the previous run properties.

Editing Information for All User and Computer Groups

At times, you might want to edit information for more than one user or computer group. Although you can edit each group individually, it is easier to edit the information for all groups at the same time.

To edit information for all groups:

- Open a suite, and then click **Suite > Edit Groups**.



User Group	Computers	Number of Users

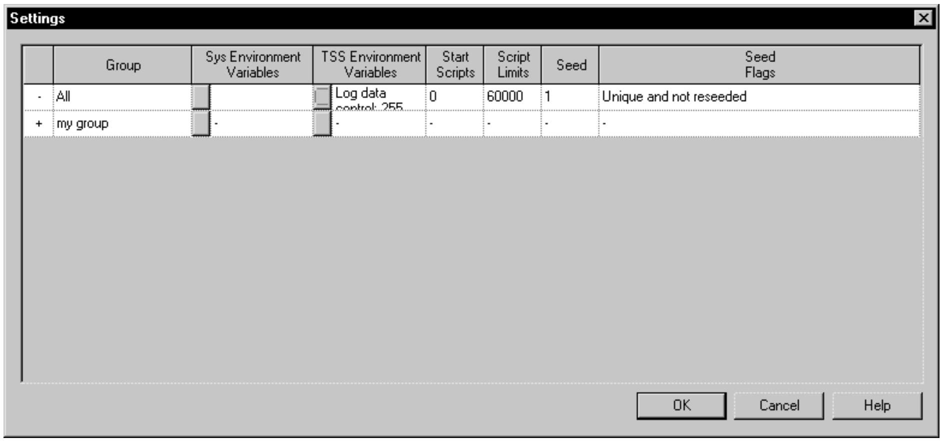
User Group	Computers	%
Accounting	Local computer	20.000
Data Entry	Local computer	30.000
Sales	Local computer	50.000

Editing Settings for Virtual Testers

You can change the default settings associated with virtual testers. You can set Test Script Services (TSS) environment variables to change which information is logged when you run a suite. For example, if you have problems running a suite, set one virtual tester to log all events and the other virtual testers to log failed events only so that you can investigate the problem more thoroughly.

To edit virtual tester settings:

▪ Open a suite, and then click **Suite > Edit Settings**.



Settings

	Group	Sys Environment Variables	TSS Environment Variables	Start Scripts	Script Limits	Seed	Seed Flags
-	All		Log data control: 255	0	60000	1	Unique and not reseeded
+	my group	-	-	-	-	-	-

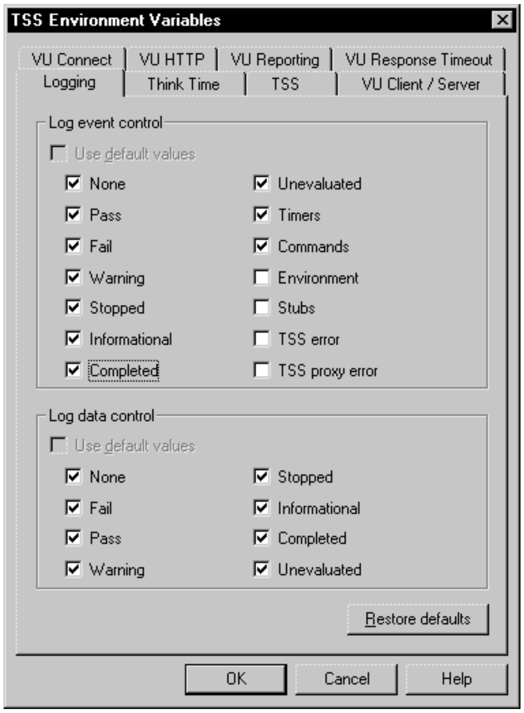
Initializing TSS Environment Variables

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

To initialize a TSS environment variable:

1 Open a suite, and then click **Suite > Edit Settings.**

2 In the Settings dialog box, click the button in the **TSS Environment Variables** column.



Log event control

☐ Use default values

✓ None	✓ Unevaluated
✓ Pass	✓ Timers
✓ Fail	✓ Commands
✓ Warning	☐ Environment
✓ Stopped	☐ Stubs
✓ Informational	☐ TSS error
✓ Completed	☐ TSS proxy error

Log data control

☐ Use default values

✓ None	✓ Stopped
✓ Fail	✓ Informational
✓ Pass	✓ Completed
✓ Warning	✓ Unevaluated

3 Choose one of the following tabs in the TSS Environment Variables dialog box.

- **Logging** – Logging environment variables pertain to any test script in the suite. They let you set the level of detail that appears in the log files associated with a suite run.
- **Think time** – Think time environment variables pertain to any test script in the suite. They control the virtual tester's "think time" behavior. This is the time that a typical user would delay, or think, between submitting commands.
- **TSS** – TSS enables or disables Test Script Services, as described in the following section.
- **VU Client/Server** – Client/server environment variables pertain to VU test scripts that access a SQL database. These variables let you include SQL column headers, set the number of bytes to include in a response, and halt data retrieval at the end of a SQL table.

- **VU Connect** – Connect environment variables pertain to VU test scripts that record HTTP and socket protocols. They control the number of times a test script retries to connect to a server and the delay between retries.
- **VU HTTP** – HTTP environment variables pertain to VU test scripts that record the HTTP protocol. These variables let you emulate line speed for http and socket requests. They also enable an HTTP test script to play back successfully if the server responds with data that does not exactly match what was recorded—for example, if the server responds with partial data, if the response was cached, or if the test script was directed to another server during playback.
- **VU Reporting** – Reporting environment variables pertain to VU test scripts. These variables let you set how to check for SQL unread row results and set the maximum number of bytes to receive in a SQL response.
- **VU Response Timeout** – Response timeout environment variables pertain to VU test scripts that record HTTP, SQL, IIOP, or socket protocols. These variables let you set the timeout for a VU emulation command, scale timeouts, and set the action to take if a timeout occurs.

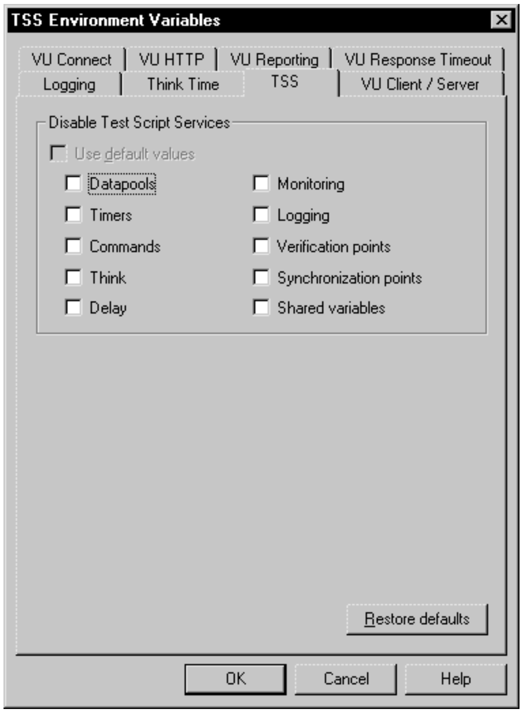
Note: For more information on TSS environment variables, see the *VU Language Reference* or the Rational Test Script Services manual for your language.

Disabling Test Script Services

You can disable Test Script Services during a suite run. Disabling Test Script Services lets you reduce runtime overhead. Disabling also lets you create a test script that uses all Test Script Services and then, based on what is enabled or disabled, test only functional- or performance-related services.

By default, no Test Script Services are disabled. To disable Test Script Services:

1. Open a suite, and then click **Suite > Edit Settings.**
2. Click the button in the **TSS Environment Variables** column.
3. Choose the **TSS** tab in the TSS Environment Variables dialog box.



Enable or disable the following services:

- **Datapools** – Use of datapools in test scripts.
- **Timers** – Use of timers in test scripts.
- **Commands** – Use of TSS `start` and `end` commands in test scripts. Commands will not be timed, command information will not be written to a log as an event, and there will not be any reports to analyze.
- **Think time** – Use of the TSS think time routines in test scripts.
- **Delays** – Use of TSS delay routines in test scripts.
- **Monitoring** – Monitoring of a suite run.
- **Logging** – Logging during a suite run.
- **Verification points** – Use of verification points in test scripts.
- **Synchronization points** – Use of synchronization points in test scripts.
- **Shared variables** – Use of shared variables in test scripts.

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 successfully before the next group starts. You may need to do this to control the maximum number of virtual testers that log on to a system 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:

- Open a suite, and then click **Suite > Edit Settings**.

Limiting the Number of Test Scripts

TestManager lets you limit the number of test scripts that virtual testers can run without having to remove any test scripts from the user group. You can test the suite run in an abbreviated form to ensure that all suite items are working correctly.

For example, limit the number of test scripts to:

- Ensure that a sequence can run for a limited time before you repeat it indefinitely in a suite. Assume that your suite has a sequence of eight test scripts that repeats indefinitely. To ensure that virtual testers are able to start, execute, and complete the sequence twice, insert a sequential selector before the test scripts, and then set **Script Limits** to 16.
- 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.)
- Check that a datapool works correctly. For example, assume that a virtual tester enters records into a database 100 times. To check that the virtual tester enters a unique record, set **Script Limits** to 2. If a virtual tester enters the same record twice, the run fails.

To limit the number of test scripts that a suite runs:

- Open a suite, and then click **Suite > Edit Settings**.

Changing the Way Random Numbers Are Generated

Each virtual tester has a *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, in the VU language, if the first virtual tester calls the `uniform` routine twice 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 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.

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. However, 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:

- Open a suite, and then click **Suite > Edit Settings**.

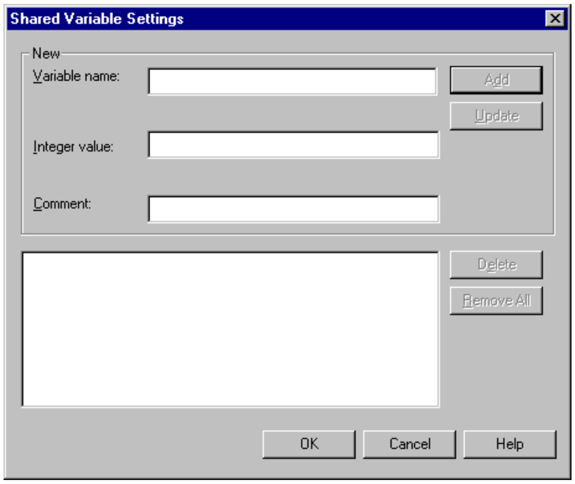
The VU routines that generate random numbers are `NegExp`, `Rand`, and `Uniform`. For more information about these routines, see the *VU Language Reference* or the Rational Test Script Services manual for your language.

Initializing Shared Variables

TestManager lets you 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:

- Open a suite, and then click **Suite > Edit Shared Variables**.



Shared variables are useful in the following situations:

- 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 the appropriate wait routine in your scripting language.
- 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.
- 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.
- Monitoring specific transaction counts and conditions. When you monitor a suite, shared variables 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" routine. For more information about this routine, see the *VU Language Reference* or the Rational Test Script Services manual for your language.

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:

- Open the suite to print, and then click **File > Print**.

To export a suite as a .txt file:

- Open the suite to export, and then click **File > Export to File**

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. Running a suite automatically performs a save.

To save a suite manually:

- Open the suite to save, and then click **File > Save**.

To save more than one suite:

1. Click **File > Open Suite** to open the suites to save.
2. Click **File > Save All**.

Note: If you click **Tools > Options**, click the **Create Suite** tab, and 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:

- Open the suite to save, and then click **File > Save As**.

Executing Tests 5

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
- Stopping suites

Note: For detailed procedures, see the TestManager Help.

About Running Tests

The activity of running your tests is primarily running the implementation of each test case to validate the specific behavior that the test case is intended to validate.

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 types of test scripts:

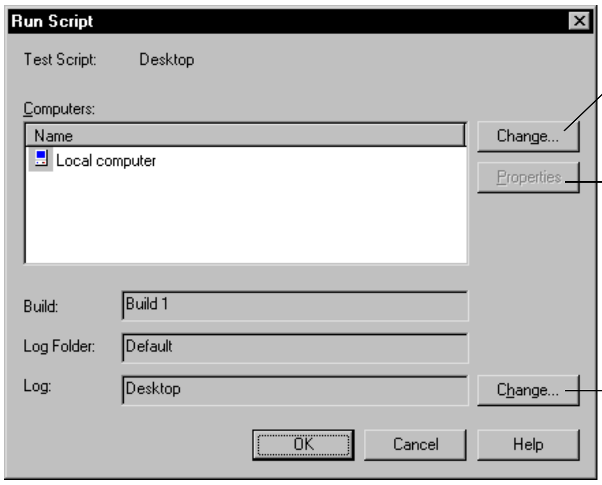
Type of Test Script	Description
GUI	A functional test script written in SQABasic, a Rational proprietary Basic-like scripting language.
VU	A performance test script written in VU, a Rational proprietary C-like scripting language.
Manual	A set of testing instructions to be run by a human tester.
VB	A 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 from TestManager:

1 Click **File > Run Test Script**, and select the test script type.

2 Select the test script to run and click **OK** to open the Run Script dialog box.


Run Script

Test Script: Desktop

Computers:

Name
Local computer

Build: Build 1

Log Folder: Default

Log: Desktop

Click to change the computers and computer lists that are available for the test script to run on.

Click to view or edit the properties of the selected computer or computer list. (This is disabled if Local computer is selected.)

Click to change the build, log folder, or log name.

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 Agent Computers and Computer Lists* on page 69.

For information about monitoring progress, see *Monitoring Suites* on page 103.

For information about test logs, see *About Test Logs* on page 127.

Note: When you run a test script, TestManager does not generate test case coverage results (even if the test script is associated with a test case). To generate test case results, run the test case instead of the test script. For information, see *Running Test Cases* on page 90.

Running Manual Test Scripts

You can do the following when you run a manual test script:

- Optionally, set a run option to log unchecked steps and verification points as warnings (in Rational ManualTest, select **Tools > Options**).
- Indicate that you have performed each step.
- Indicate whether each verification point passed or failed.

Note: For information about creating manual test scripts, see *Creating Manual Test Scripts* on page 61.

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

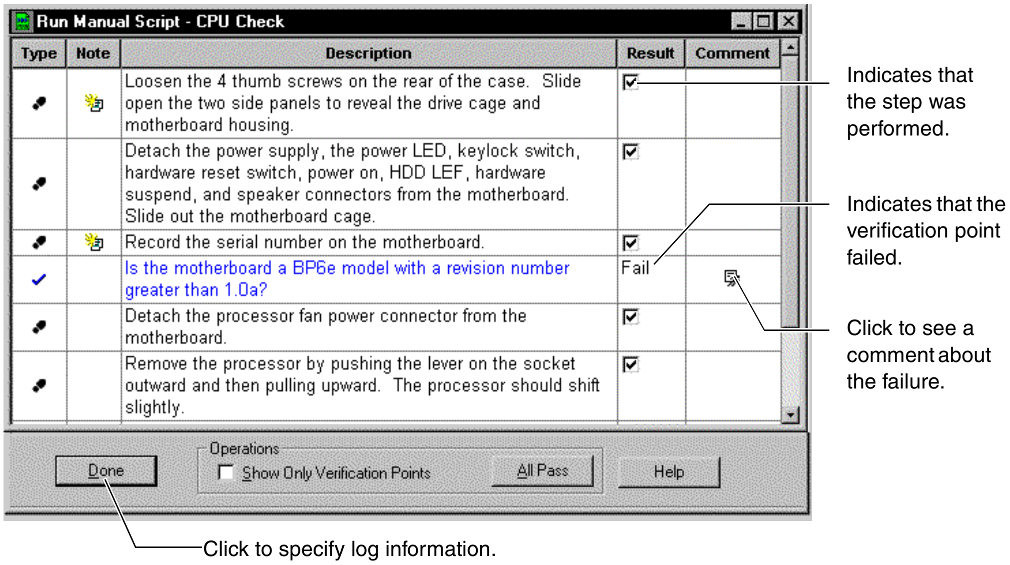
- In TestManager, click **File > Run Test Script > Manual** and select a test script. Rational ManualTest opens.
- In Rational ManualTest, click **File > Run** and select a test script.

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.

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 test log, you will be able to see the failure and the comment.


Run Manual Script - CPU Check
Indicates that the step was performed.
Indicates that the verification point failed.
Click to see a comment about the failure.
Operations
Show Only Verification Points
Done
All Pass
Help
Click to specify log information.

Type	Note	Description	Result	Comment
		Loosen the 4 thumb screws on the rear of the case. Slide open the two side panels to reveal the drive cage and motherboard housing.	☑	
		Detach the power supply, the power LED, keylock switch, hardware reset switch, power on, HDD LEF, hardware suspend, and speaker connectors from the motherboard. Slide out the motherboard cage.	☑	
		Record the serial number on the motherboard.	☑	
✓		Is the motherboard a BP6e model with a revision number greater than 1.0a?	Fail	
		Detach the processor fan power connector from the motherboard.	☑	
		Remove the processor by pushing the lever on the socket outward and then pulling upward. The processor should shift slightly.	☑	

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

For information about test logs, see *About Test Logs* on page 127.

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. In the Test Plan window, right-click a test case, and then 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.

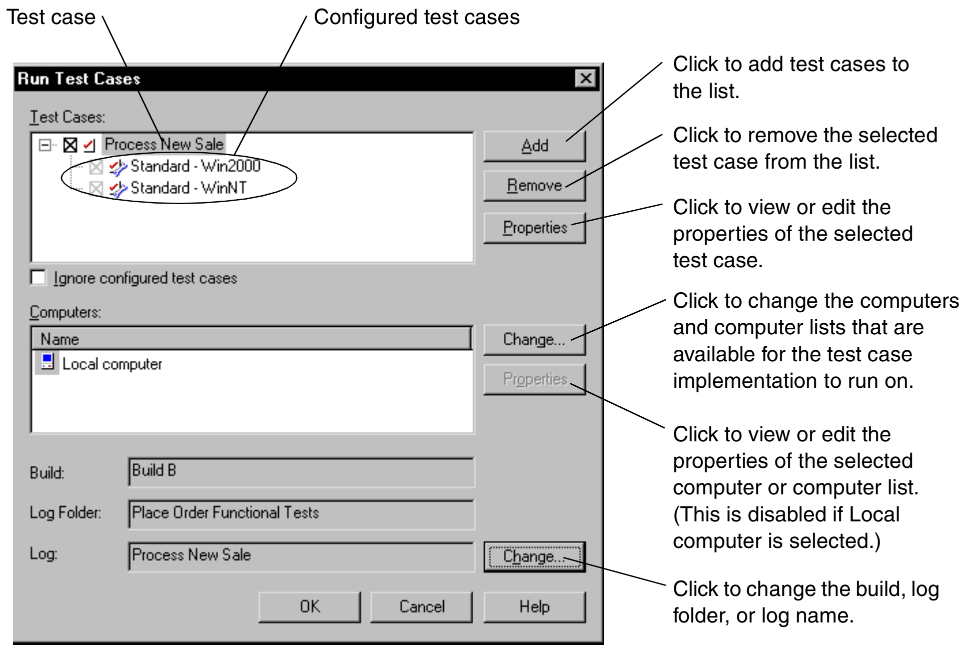
Note: If you run a test case from the Rational ManualTest Web Execution component, the manual test will run. For information, see *About ManualTest Web Execution* on page 351.

For more information about implementing test cases and associating implementations, see *Implementing Tests* on page 55.

Running a Test Case

To run a test case, do one of the following:

- Click **File > Run Test Case**. Select the test case to run and click **OK**.
- Click **File > Run Test Cases for Iteration**. Select the iteration and click **OK**.



Test Cases: Process New Sale; Standard - Win2000; Standard - WinNT

Ignore configured test cases

Computers: Local computer

Build: Build B

Log Folder: Place Order Functional Tests

Log: Process New Sale

Click to add test cases to the list.

Click to remove the selected test case from the list.

Click to view or edit the properties of the selected test case.

Click to change the computers and computer lists that are available for the test case implementation to run on.

Click to view or edit the properties of the selected computer or computer list. (This is disabled if Local computer is selected.)

Click to change the build, log folder, or log name.

Note: When you run a test case, TestManager creates a temporary suite and actually runs the suite. TestManager removes the suite after the run is completed.

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 first available computer in the list.

  A configured test case will run only on computers that fully match the test case's configuration. TestManager takes into account the values of that computer's built-in configuration attributes and the values of the custom attributes specified in the computer's tmsconfig.csv file. (For information about the tmsconfig.csv file, see *Setting Up Custom Attributes in tmsconfig.csv* on page 38.)

  For example, if the configured test case indicates that it should run on a computer with Windows 2000, TestManager examines each computer in the Computers list until it finds one that has Windows 2000, and then runs the test case implementation 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 103. For information about test logs, see *About Test Logs* on page 127.

- If you run a test case that has a manual implementation and no automated implementation, Rational ManualTest starts. 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 89.

Ignoring Configured Test Cases

If you select the **Ignore configured test cases** check box in the Run Test Cases dialog box, TestManager ignores the configurations and runs the test cases on any available computers.

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

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

- If the selected test case has configured test cases but does not have an implementation, TestManager does not run the test case or any configured test cases.

- If a single configured test case is selected, TestManager runs the test case on the specified configuration.

Running Suites

The following steps are involved with running a suite:

- 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.

The following sections describe these steps. For information about creating a functional testing suite, see *Creating Functional Testing Suites* on page 165. For information about creating a performance testing suite, see *Designing Performance Testing Suites* on page 225.

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. This can help you identify and correct problems.

To check a suite:

1. Click **File > Open Suite**, and select a suite.
2. 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.

- A test asset (computer, computer list, test script, suite, or test case) in the suite has been deleted.

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.

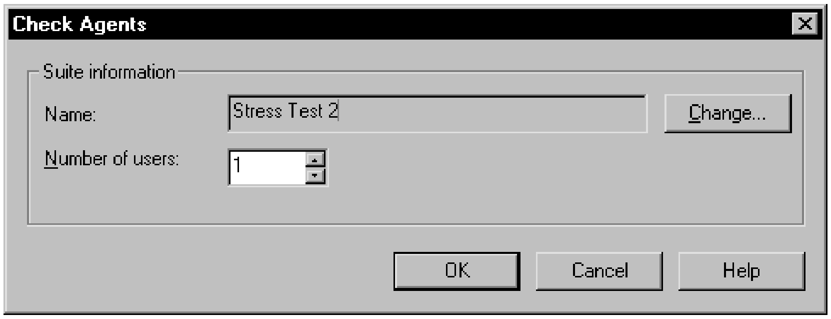
- The Agent computers are available and running.
- The Agent software is running.

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

To check the Agent computers:

1 Click **File > Open Suite**, and select a suite.

2 Click **Suite > Check Agents**.



Name: Stress Test 2

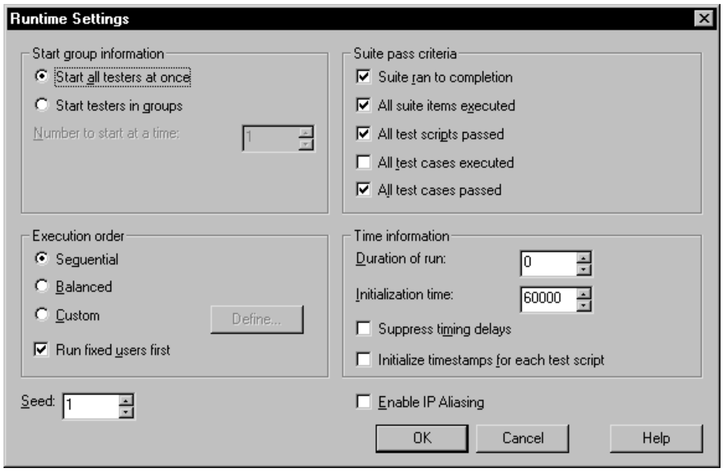
Number of users: 1

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

Controlling Runtime Information of a Suite

To set the runtime settings for a suite:

1. Click **File > Open Suite**, and select a suite.
2. Click **Suite > Edit Runtime**.



Start group information
- (●) Start all testers at once
- ( ) Start testers in groups
- Number to start at a time: 1

Suite pass criteria
- [✓] Suite ran to completion
- [✓] All suite items executed
- [✓] All test scripts passed
- [ ] All test cases executed
- [✓] All test cases passed

Execution order
- (●) Sequential
- ( ) Balanced
- ( ) Custom
- [✓] Run fixed users first

Time information
- Duration of run: 0
- Initialization time: 60000
- [ ] Suppress timing delays
- [ ] Initialize timestamps for each test script

Seed: 1

[ ] Enable IP Aliasing

By modifying the runtime settings, you can manage:

- **Start group information** – in performance testing, controls how virtual users are started.
- **Suite pass criteria** – the criteria for whether a suite passes or fails.
- **Execution order** – in performance testing, controls the order in which a user group runs.
- **Time information** – controls how long the suite runs.
- **Seed** – sets the number to feed to the random number generator.
- **Enable IP aliasing** – controls aliasing in VU HTTP scripts.

Start Group Information

In performance testing, the **Start group information** controls how virtual testers are started. Virtual testers can start all at once or in groups.

To avoid overloading a server, start virtual testers in groups, and specify 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.

Suite Pass Criteria

The **Suite pass criteria** controls whether a suite passes or fails. Select one of the following:

- **Suite ran to completion** – The suite must run to completion without manual termination of the run.
- **All suite items executed** – All items in the suite must complete their assigned tasks.
- **All test scripts passed** – No events fail and no commands time out.
- **All test cases executed** – All test cases in the suite must complete all of their assigned tasks.
- **All test cases passed** – All test cases pass, 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.”

Execution Order

In performance testing, 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:

- **Sequential** – Suites that run in a sequential order run each virtual tester as it is encountered in the suite (from the top to the bottom).
- **Balanced** – Suites that run in a balanced order evenly distribute the run among the user groups in proportion to the suite.
- **Run fixed users first** – Runs the fixed user groups before the scalable user groups, regardless of the execution order. If your user groups are all fixed, this option has no effect.

Assume that you have defined four user groups with a total of 101 virtual testers, and your suite contains them in this order:

- End of Month Accounting: 1 virtual tester
- Accounting: 20 virtual testers
- Data Entry: 30 virtual testers
- Sales: 50 virtual testers

Although your suite contains 101 virtual testers, you want to run it with only 10 virtual testers. The following table lists how the execution order affects which users are run:

If you select	You run these user groups
Sequential	1 End of Month Accounting 9 Accounting
Balanced	2 Accounting 3 Data Entry 5 Sales
Balanced and **Run Fixed Users First**	1 End of Month Accounting 2 Accounting 3 Data Entry 4 Sales

- **Custom** – 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.

You can also temporarily disable a fixed user group by selecting it, clicking **Edit > Run Properties**, and then setting **User Count** 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.

Time Information

This option controls 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.

- Suppressing timing delays, thus running the suite very quickly. 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. However, 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.
- Initializing timestamps for each test script, which indicates whether timestamps are carried over from script to script or are reinitialized with each script.

Seed

This option sets 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.

IP Aliasing

In VU test scripts that use the HTTP protocol, this option sets 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 about setting up IP Aliasing, see *Configuring Local and Agent Computers* on page 331.

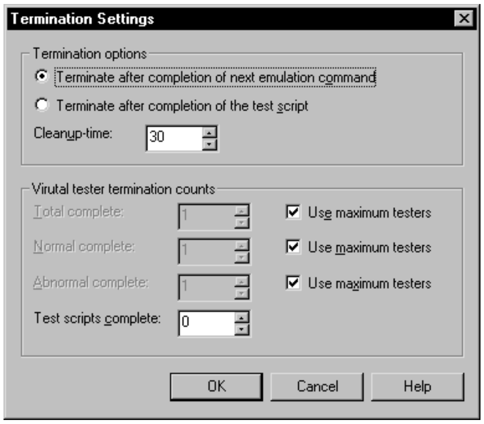
Controlling How a Suite Terminates

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

To control how a suite terminates:

1 Click **File > Open Suite**, and select a suite.

2 Click **Suite > Edit Termination**.



Termination Settings

Termination options

(●) Terminate after completion of next emulation command

( ) Terminate after completion of the test script

Cleanup-time: 30

Virutal tester termination counts

Total complete: 1 [✓] Use maximum testers

Normal complete: 1 [✓] Use maximum testers

Abnormal complete: 1 [✓] Use maximum testers

Test scripts complete: 0

Running a Suite

When you run a suite, you supply runtime-specific guidelines. Each virtual tester that executes its assigned suite items run within these guidelines.

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

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

When you run a suite, you specify:

- Number of virtual testers, if you are running a performance testing suite.

  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 you are running a functional testing suite and have not specified resources.

- 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 test 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.

  When you monitor a suite, TestManager records how computer resources are used and then graphs this usage data over the corresponding virtual tester response times when you analyze your results. Specify the interval at which monitoring 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 the configurations and to run the test cases on any available computers.

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

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

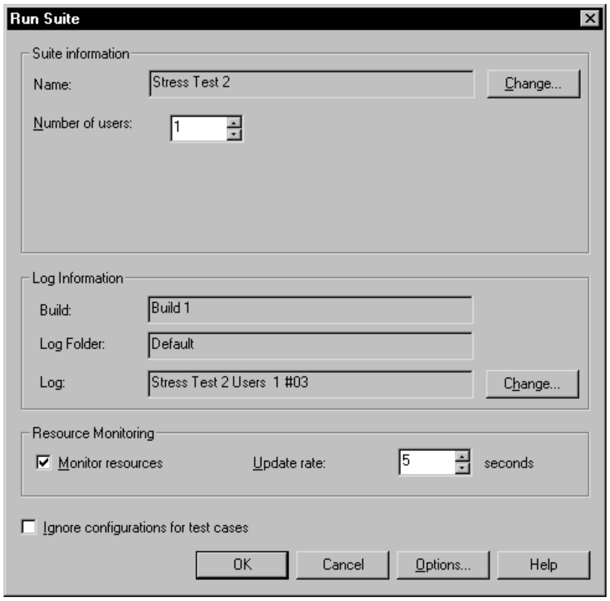
  - If the selected test case has configured test cases but does not have an implementation, TestManager does not run the test case or any configured test cases.

  - If a single configured test case is selected, TestManager runs the test case on the specified configuration.

Running a Suite from TestManager

To run a suite from TestManager:

- Click **File** > **Run Suite**.



Suite information
Name: Stress Test 2
Number of users: 1

Log Information
Build: Build 1
Log Folder: Default
Log: Stress Test 2 Users 1 #03

Resource Monitoring
☑ Monitor resources Update rate: 5 seconds

☐ Ignore configurations for test cases

The dialog box that TestManager displays differs depending on the type of suite that you run.

If you are running a performance testing suite, you must specify the number of virtual testers to run.

If you are running a functional testing suite, and you have not specified computer resources, you must specify the computers on which to run the suite.

Running a Suite from the Command Line

You can run a suite from the command line or in batch mode. For example, if you have a suite that takes a long time to run, or if you want to run several suites at one time, you can schedule them to run in a batch at any time that's convenient. When you run a suite from the command line, the TestManager application opens, runs the suite, and optionally, closes.

The following syntax shows how to run a suite from the command line:

```
rtmanager.exe suitename /runsuite /user userid [/password password]
/project .rsp path [/computers [Local];
[computer 1; computer 2; ... computer n]
[/computerlists computerlists] /build buildname
/logfolder logfoldername /log logname [/overwritelog] [/numusers nnn]
[/ignoreconfiguredtestcases] [/close]
```

If an argument contains spaces, enclose it in quotation marks.

Syntax Element	**Description**
suitename	The name of the suite to run.
/runsuite	Runs the suite referenced in suitename.
user *userid*	The user name for logging on to TestManager.
password *password*	An optional password for logging on to TestManager. Do not use this option if there is no password.
/project *.rsp path*	The full path to the project's .rsp file.
/computers [Local]; [*computer 1; computer 2;… computer n*]	The names of the computers to use when running the suite. **Local** is the computer on which TestManager is running. This option is ignored if the specified suite indicates the computers on which to run it. If the suite does not indicate the computers or computer lists to use, this option is required.
/computerlists [*computerlist 1; computerlist 2,… computerlist n*]	The computer lists to use when running the suite. This option is ignored if the specified suite indicates which computers to use to run it, or if you specified computers in the /computers option.
/logfolder *logfoldername*	The name of the log folder in which to create the test log. If the log folder does not exist, TestManager creates it.
/build *buildname*	The build in which to create the log folder. If the specified build does not exist, TestManager creates it.
/log *logname*	The name of the test log in which to record the results of running the suite. If the log exists, you must specify the overwritelog option.
/overwritelog	Overwrite the test log if it exists. If you omit this option and the log exists, a message appears that says the log exists and cannot be overwritten.
/ignoreconfiguredtestcases	Ignore configuration matching (such as test cases that specify computers with specific operating systems or other attributes) when running test cases that are in the specified suite.

Syntax Element	Description
/numusers *nnn*	The target number of virtual testers when executing a suite that contains user groups. This is mandatory for running performance testing suites.
/close	Closes TestManager after running the suite.

Example

This example runs the suite named **MySuite**:

```
rtmanager.exe MySuite /runsuite /user admin /project "C:\Sample
Project\Sample.rsp" /computers Local /build "Build 1" /logfolder
Default /log Sample /close
```

In this example, TestManager logs on to the project C:\Sample Project\Sample.rsp as the virtual tester admin and runs **MySuite** on the local computer. The results of running **MySuite** are stored in the test log **Sample**, which is created in the log folder **Default** in build **Build 1**. After the run is complete, TestManager closes.

To make the results of the suite run remain on the screen, omit the **/close** option. The results also appear in the default reports.

Monitoring Suites

While a suite is running, you can monitor its progress. You can:

- Confirm that a suite is progressing successfully.
- Discover potential problems early in the run so you can intervene if necessary.
- Suspend and restart virtual testers.
- Change the values of shared variables.
- 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.

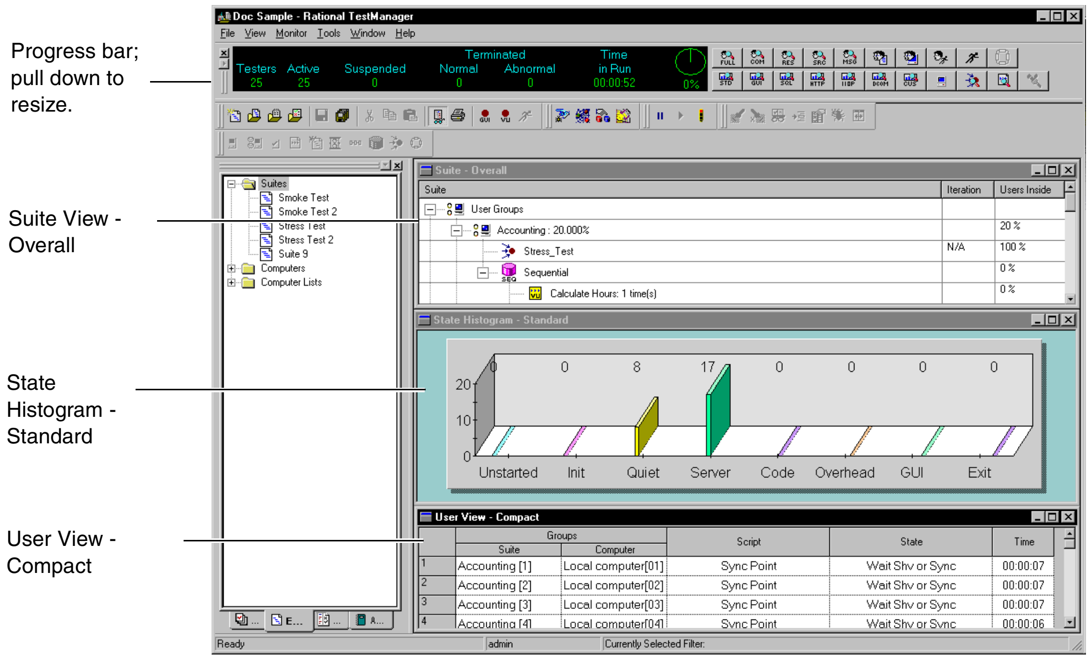
- Whether any virtual testers have terminated abnormally.

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

The Progress Bar and the Default Views

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:



Suite	Iteration	Users Inside
User Groups		
Accounting : 20.000%		20 %
Stress_Test	N/A	100 %
Sequential		0 %
Calculate Hours: 1 time(s)		0 %

	Groups		Script	State	Time
	Suite	Computer			
1	Accounting [1]	Local computer[01]	Sync Point	Wait Shv or Sync	00:00:07
2	Accounting [2]	Local computer[02]	Sync Point	Wait Shv or Sync	00:00:07
3	Accounting [3]	Local computer[03]	Sync Point	Wait Shv or Sync	00:00:07
4	Accounting [4]	Local computer[04]	Sync Point	Wait Shv or Sync	00:00:06

With the Progress bar, you can 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 **Suite - Overall** view, which displays general information about the status of virtual testers. For more information, see *Displaying the Suite Views* on page 105.
- The **State Histogram - Standard**, which is a bar chart that provides a general idea of which 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 State Histogram - Standard by default. However, if you are running GUI test scripts, or if you are testing a SQL database or a Web server, you may want to display a bar chart specifically geared to those tests. For more information, see *Displaying the State Histograms* on page 107.
- The **User View - Compact** or the **Computer View - Compact**, which displays information about the current state of the virtual testers. In this view, you can click a particular virtual tester to display additional information about that virtual tester or control its operation. For more information, see *Displaying the User and Computer Views* on page 109.

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.

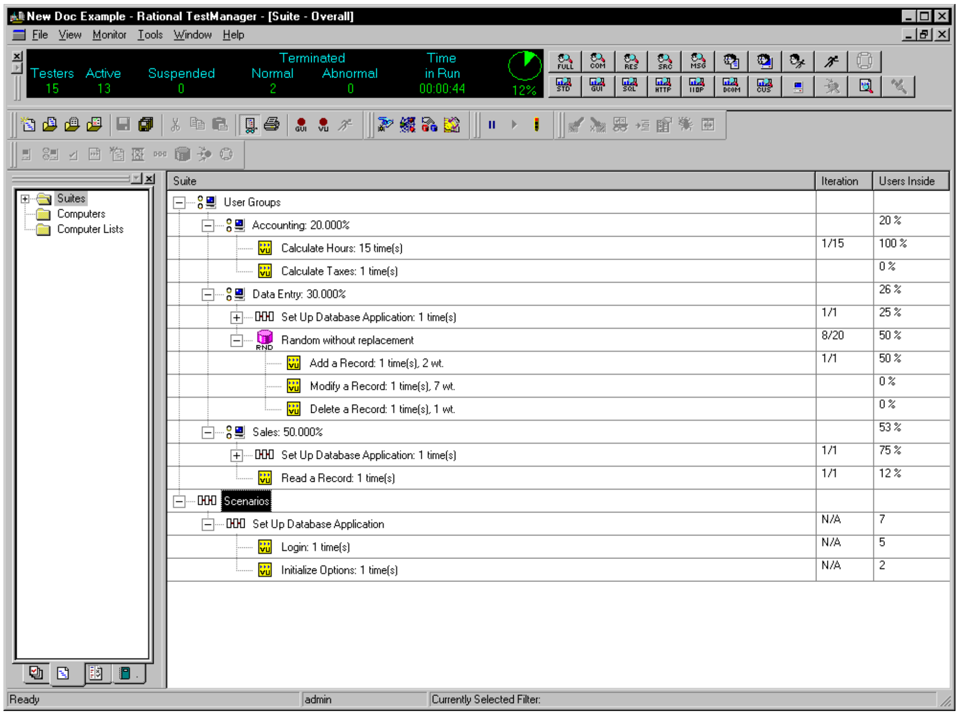
The two suite views are:

- **Suite - 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.
- **Suite - Users** or **Suite - Computers** – Use this view to display the exact suite progress of a particular virtual tester.

The Suite - Overall View

To display the Suite - Overall view:

- During a suite run, click **Monitor > Suite > Overall**.



New Doc Example - Rational TestManager - [Suite - Overall]

File View Monitor Tools Window Help

Testers	Active	Suspended	Terminated Normal	Terminated Abnormal	Time in Run	
15	13	0	2	0	00:00:44	12%

Suite	Iteration	Users Inside
User Groups		
Accounting: 20.000%		20 %
Calculate Hours: 15 time(s)	1/15	100 %
Calculate Taxes: 1 time(s)		0 %
Data Entry: 30.000%		26 %
Set Up Database Application: 1 time(s)	1/1	25 %
Random without replacement	8/20	50 %
Add a Record: 1 time(s), 2 wt.	1/1	50 %
Modify a Record: 1 time(s), 7 wt.		0 %
Delete a Record: 1 time(s), 1 wt.		0 %
Sales: 50.000%		53 %
Set Up Database Application: 1 time(s)	1/1	75 %
Read a Record: 1 time(s)	1/1	12 %
Scenarios		
Set Up Database Application	N/A	7
Login: 1 time(s)	N/A	5
Initialize Options: 1 time(s)	N/A	2

The Suite - Overall 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.

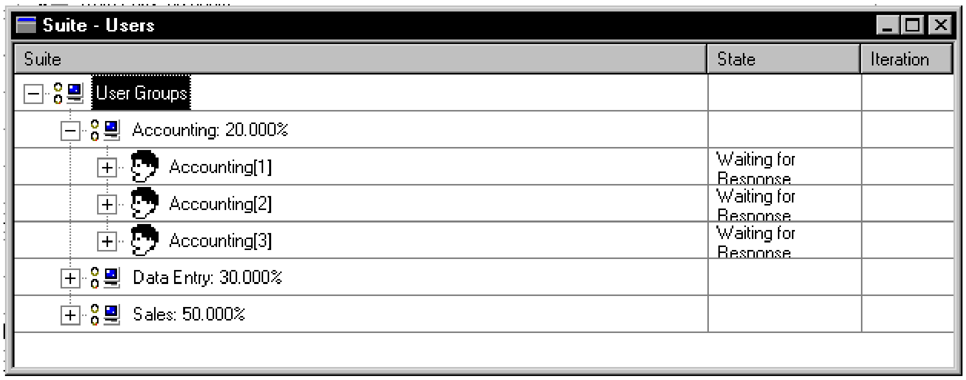
The **Users Inside** column shows the percentage of virtual testers that are currently executing each portion of the suite. The percentage next to the user group shows the percentage of total virtual testers that have been assigned to the group and have not yet exited the suite. The percentage next to 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 Suite - Users and Suite - Computers Views

To display the Suite - Users or the Suite - Computers view:

- During a suite run, click **Monitor > Suite > Users** to display a Suite - Users view. Click **Monitor > Suite > Computers** to display a Suite - Computers view.



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%		

For information about each column in this view, see *Suite - Users* or *Suite - Computers* in the TestManager Help Index.

Displaying the State Histograms

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

To display a state histogram:

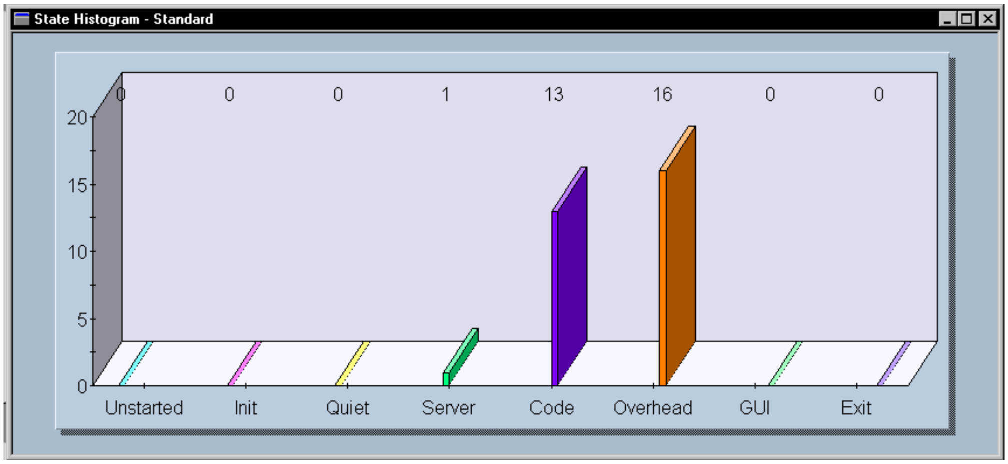
- During a suite run, click **Monitor > Histogram**, and select the desired histogram.

The state histograms are:

- **Standard** – Data is grouped in a general way. Select this histogram if you want a general overview of the virtual tester states. For information about each bar in this histogram, see *State Histogram - Standard* in the TestManager Help Index.
- **GUI** – Data is grouped appropriately for tests that run GUI test scripts. For information about each bar in this histogram, see *State Histogram - GUI* in the TestManager Help Index.

- **SQL** – Data is grouped appropriately for tests that access SQL databases. For information about each bar in this histogram, see *State Histogram - SQL* in the TestManager Help Index.
- **HTTP** – Data is grouped appropriately for tests that access Web servers. For information about each bar in this histogram, see *State Histogram - HTTP* in the TestManager Help Index.
- **IIOP** – Data is grouped appropriately for tests that access IIOP servers. For information about each bar in this histogram, see *State Histogram - IIOP* in the TestManager Help Index.
- **DCOM** – Data is grouped appropriately for the tests that access DCOM functions. For information about each bar in this histogram, see *State Histogram - DCOM* in the TestManager Help Index.
- **Custom** – Data is grouped according to your needs. For information about customizing a histogram, see *Configuring Custom Histograms* on page 122.

The following figure shows a State Histogram - Standard:



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.

Zooming In on Histogram Bars

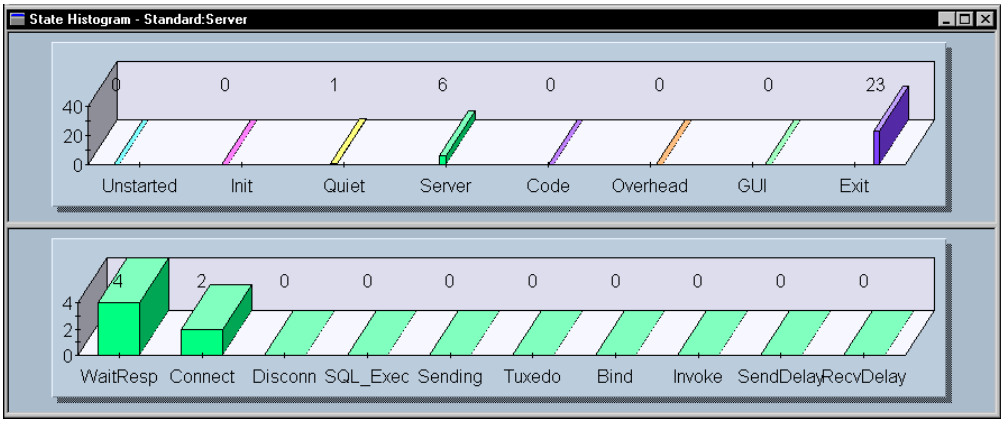
Each bar in a histogram shows a summary state that contains individual states. You can zoom in on a bar to see a breakdown of how many virtual testers are in each state.

To zoom in on a histogram bar:

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

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

The following figure shows an expanded histogram, after you have clicked 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.

For information about each bar in this histogram, see *List of Histogram Bars* in the TestManager Help Index.

Displaying the User and Computer Views

The User and 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.

Note: TestManager displays a view based on *users* or based on *computers,* depending on the type of suite that you are running.

To display a view:

- During a suite run, click **Monitor > User** or **Monitor > Computer**, and select a view.

The User/Computer views are:

- **User/Computer View - Full** – Contains complete information about all virtual testers.

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

The following items pertain to all user or computer views:

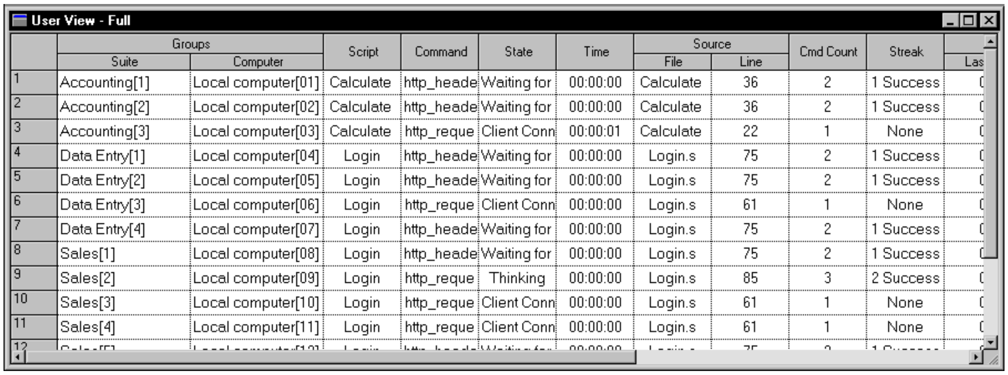
- To make tracking certain virtual testers easier, you can change which virtual testers are displayed. For more information, see *Filtering and Sorting Views* on page 119.
- When you display a user or computer view, you can also display the test script that the virtual tester is running. Double-click the number in the first column, next to the virtual tester. TestManager displays the test script. For more information, see *Displaying the Test Script View* on page 114.
- When a virtual tester terminates abnormally, TestManager writes a message stating the reason for termination to the running Suite window. Right-click the terminated virtual tester, and then select the **View Termination Message** option.

  You can easily identify a virtual tester that terminates abnormally because its **Exited** state is red in the views.

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

User/Computer View - Full

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

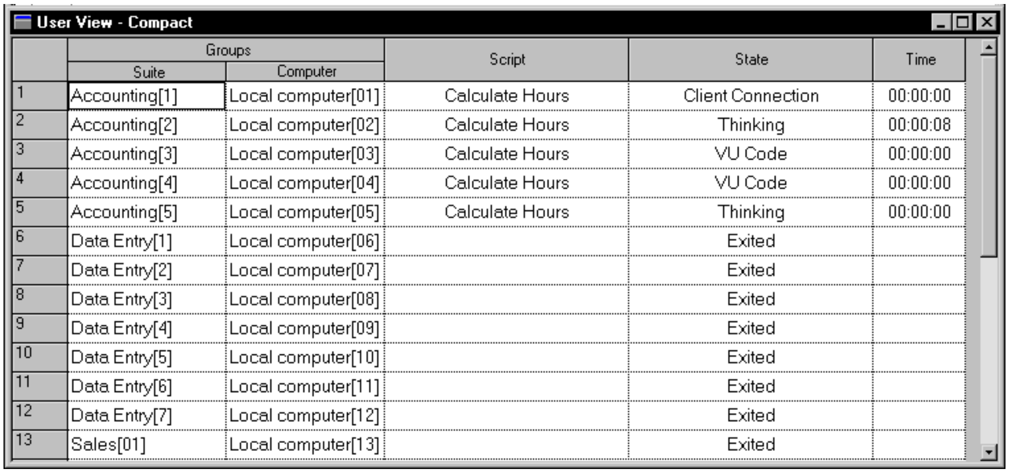

User View - Full

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

For information about each column in this view, see *User View - Full* or *Computer View - Full* in the TestManager Help Index.

User/Computer View - Compact

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

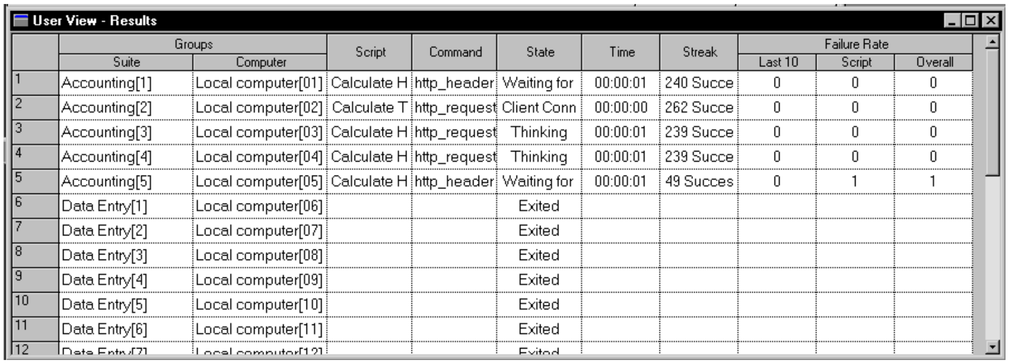

User View - Compact

	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	

For information about each column in this view, see *User View - Compact* or *Computer View - Compact* in the TestManager Help Index.

User/Computer View - Results

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



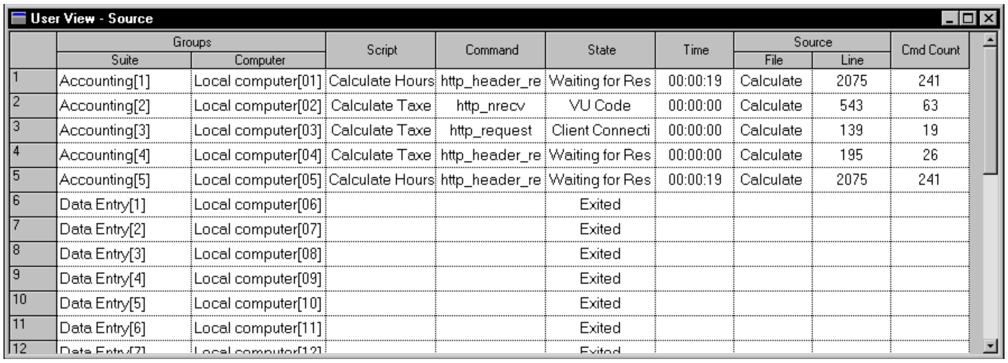
User View - Results

	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					

For information about each column in this view, see *User View - Results* or *Computer View - Results* in the TestManager Help Index.

User/Computer View - Source

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



User View - Source

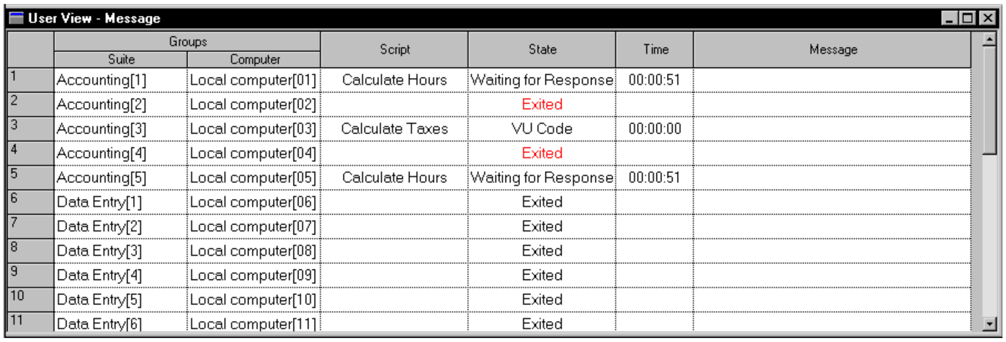
	Groups		Script	Command	State	Time	Source		Cmd Count
	Suite	Computer					File	Line	
1	Accounting[1]	Local computer[01]	Calculate Hours	http_header_re	Waiting for Res	00:00:19	Calculate	2075	241
2	Accounting[2]	Local computer[02]	Calculate Taxe	http_nrecv	VU Code	00:00:00	Calculate	543	63
3	Accounting[3]	Local computer[03]	Calculate Taxe	http_request	Client Connecti	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				

For information about each column in this view, see *User View - Source* or *Computer View - Source* in the TestManager Help Index.

User/Computer View - Message

This view is similar to the User/Computer Compact view, but it also displays messages from the first 20 letters of text from the TSS display function. If you have added a display routine to a test script, you may want to show this view.

The following figure shows an example of a User View - Message:


User View - Message

	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		

For information about each column in this view, see *User View - Message* or *Computer View - Message* in the TestManager Help Index.

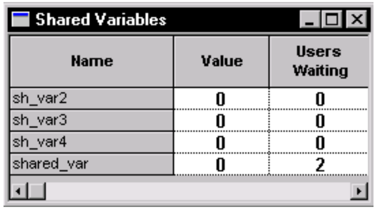
Displaying the Shared Variables View

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

To display the Shared Variables view:

- During a suite run, click **Monitor > Shared Variable**.

The following figure shows a Shared Variables view:


Shared Variables

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.

Changing the Value of a Shared Variable

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

1 During a suite run, click **Monitor > Shared Variable.**

2 Double-click the variable name, or right-click in the view and then click **Change Value**.

3 If the shared variable is read-only, type a new value in the **Value of** box.

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

   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.

5 Click **OK.**

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:

1 In the Shared Variables view, double-click the variable name, or right-click in the test script.

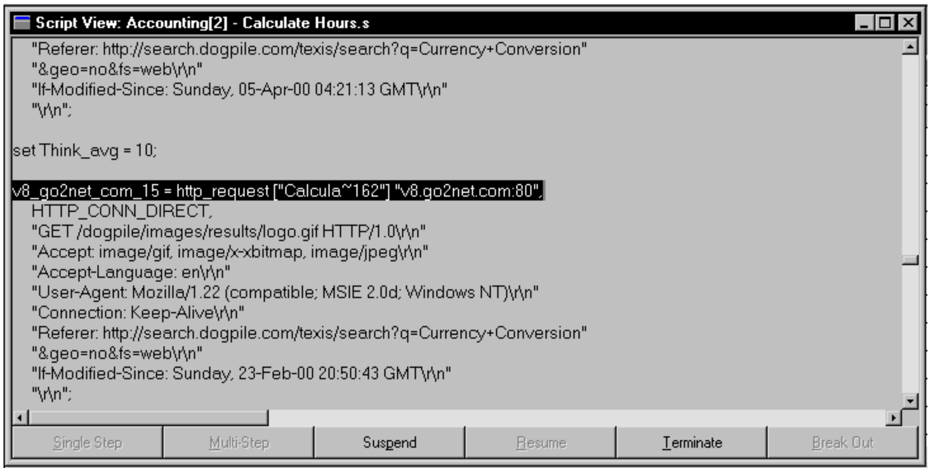
2 Click **See Users**.

Displaying the Test Script View

The Test script view displays the test script that is running and highlights the line of code that a virtual tester is executing. This view is useful for watching the progress of a virtual tester through a test script.

To display the Test script view:

- During a suite run, click **Monitor > Test Script**, and click the virtual tester whose progress you want to check.



```
   "Referer: http://search.dogpile.com/texis/search?q=Currency+Conversion"
   "&geo=no&fs=web\r\n"
   "If-Modified-Since: Sunday, 05-Apr-00 04:21:13 GMT\r\n"
   "\r\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\r\n"
   "Accept: image/gif, image/x-xbitmap, image/jpeg\r\n"
   "Accept-Language: en\r\n"
   "User-Agent: Mozilla/1.22 (compatible; MSIE 2.0d; Windows NT)\r\n"
   "Connection: Keep-Alive\r\n"
   "Referer: http://search.dogpile.com/texis/search?q=Currency+Conversion"
   "&geo=no&fs=web\r\n"
   "If-Modified-Since: Sunday, 23-Feb-00 20:50:43 GMT\r\n"
   "\r\n";
```

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

For information about the options available in this view, see *Test Script view* in the TestManager Help Index.

Debugging a Test Script

You may encounter problems when you are monitoring a suite. TestManager provides 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:

- During a suite run, click **Monitor > Test Script** and select the virtual tester running the test script that you want to debug.

Select one of the following the following debugging options:

- **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 option, 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 option, 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
  - TSS delay function

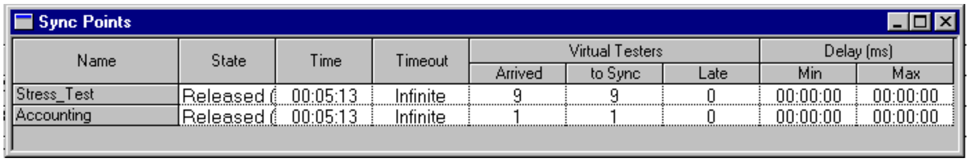
Displaying the Sync Points View

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.

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

To display the Sync Points view:

- During a suite run, click **Monitor > Sync Points**.



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

For information about each column in this view, see *Sync Points view* in the TestManager Help Index.

Displaying Virtual Testers Waiting on a Synchronization Point

To display the virtual testers waiting on a synchronization point:

1. During a suite run, click **Monitor > Sync Points**.
2. Right-click the name of the synchronization point, and then click **See Users**.

Releasing a Synchronization Point

You might decide to release a synchronization point, even though the required number of virtual testers has not yet been reached. Subsequent virtual 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 virtual 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:

1 During a suite run, click **Monitor > Sync Points**.

2 Right-click the name of the synchronization point, and then click **Release**.

3 A confirmation message appears, asking you to confirm the release. Click **Yes** or **No**.

Displaying the Computer View

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

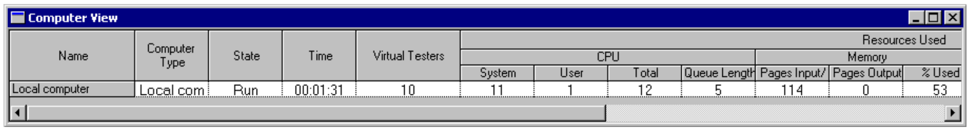
For information about each column in this view, see *Computer view* in the TestManager Help Index.

Viewing Resource Usage During a Run

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

To check computer resources used during a run:

- During a suite run, click **Monitor > Computers.**



Computer View

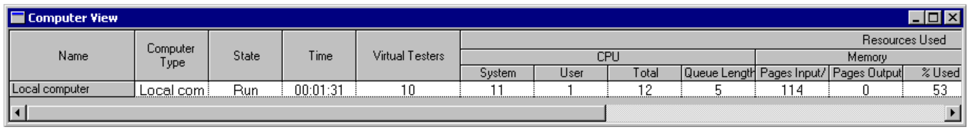
Name	Computer Type	State	Time	Virtual Testers	Resources Used						
					CPU				Memory		
					System	User	Total	Queue Length	Pages Input/	Pages Output	% Used
Local computer	Local com	Run	00:01:31	10	11	1	12	5	114	0	53

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:

- During a suite run, click **Monitor > Computers.**


Computer View

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 that you can watch 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.

For information about each column in this view, see *Computer view* in the TestManager Help Index.

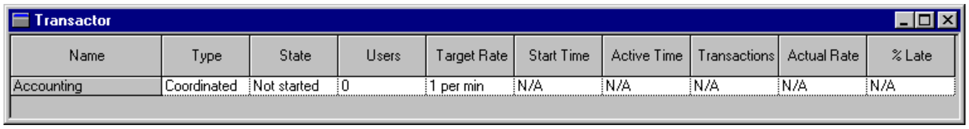
Displaying the Transactor View

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

Note: This view pertains primarily to performance testing.

To display a Transactor view:

- During a suite run, click **Monitor > Transactors**.


Transactor

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

For information about each column in this view, see *Transactor view* in the TestManager Help Index.

Displaying the Group Views

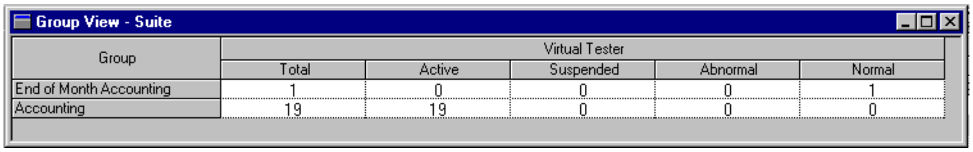
The Group views show the status of all user groups that you defined in the suite. Both Group views show the same information, but the Suite view shows the information by user group, and the Computer view shows the information by computer.

To display a Group view:

- During a suite run, click **Monitor > Groups**.

The two Group views are:

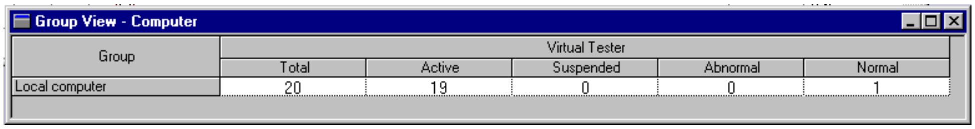
- **Group View - Suite** – A list of the user groups in a suite. The following figure shows this view:



Group View - Suite

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

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



Group View - Computer

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

For information about each column in this view, see *Group View - Suite* or *Group View - Computer* in the TestManager Help Index.

To display the virtual testers in the groups, right-click the 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 in a User or Computer View

While displaying a User or 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.

To change the order in which the virtual testers are displayed:

- Right-click in a column under the **Suite** or **Computer** heading from an open user or computer view to view the shortcut menu.

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.

Filtering a View

You can filter virtual testers in a User or Computer view so that only certain virtual 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 virtual testers.

Filtering Virtual Testers

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

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

To filter on virtual testers:

1. Select the virtual testers that you want to filter from an open user or computer view and right-click to display the shortcut menu.
2. Click **Filter Virtual Testers**, and then click **Include** or **Exclude**.

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:

1. Right-click in any cell under the Suite, Group, or Type headings of an open User or Computer view.
2. Click **Filter Virtual Testers**, and then click **By Value**.

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:

1 Right-click in the Group or Type heading of an open User or Computer view.

2 Click **Filter By Value**.

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:

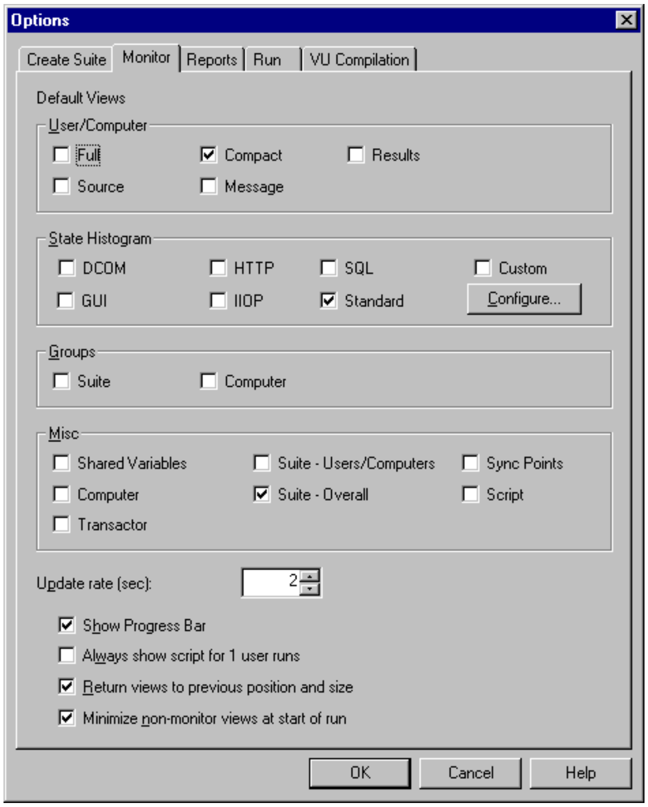
- From the view that you want to restore, click **View > Reset**.

Changing Monitor Defaults

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

To change monitor defaults:

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



Default Views

User/Computer

- [ ] Full
- [x] Compact
- [ ] Results
- [ ] Source
- [ ] Message

State Histogram

- [ ] DCOM
- [ ] HTTP
- [ ] SQL
- [ ] Custom
- [ ] GUI
- [ ] IIOP
- [x] Standard

Configure...

Groups

- [ ] Suite
- [ ] Computer

Misc

- [ ] Shared Variables
- [ ] Suite - Users/Computers
- [ ] Sync Points
- [ ] Computer
- [x] Suite - Overall
- [ ] Script
- [ ] Transactor

Update rate (sec): 2

- [x] Show Progress Bar
- [ ] Always show script for 1 user runs
- [x] Return views to previous position and size
- [x] Minimize non-monitor views at start of run

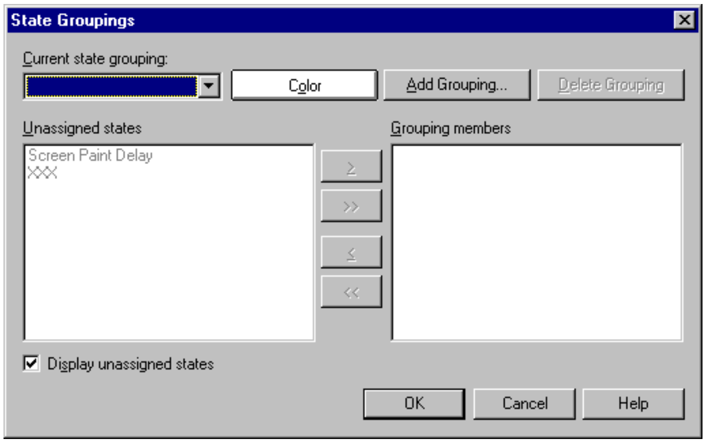
OK Cancel Help

Configuring Custom Histograms

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

To configure the State Histogram - Custom:

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

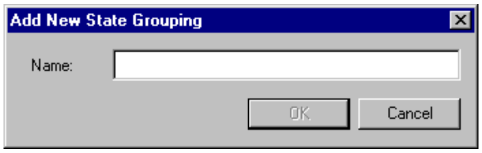


To assign or remove a state in a Custom Histogram group:

- 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 **Add Grouping**.



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, or you can suspend and resume individual virtual testers. This is useful for investigating any problems that occur during the run.

To suspend or resume all virtual testers:

- 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.

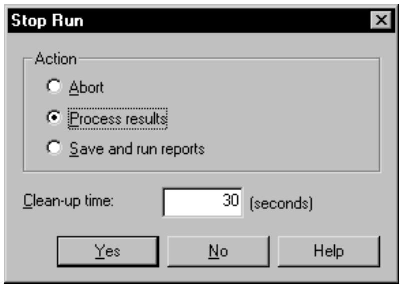
Stopping Suites

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

- In the Messages from running suite window, click **Cancel**.

To stop a suite when it is running:

- Click **Monitor > Stop**.



Stop a suite in one of these ways:

- **Abort** – Stops the run and does not save the 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 by using Task Manager, or all at once by logging off.

When the suite finishes running—whether normally or by manual termination—TestManager displays any log data in the Test Log window. For more information about the Test Log window, see *Evaluating Tests* on page 127.

Evaluating Tests 6

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

After you run a suite, test case, or test script, TestManager writes the results to a test log. You use the Test Log window of Rational TestManager to view the test logs created after you run a suite, test case, or test script.

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 test passed or 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.

You can use the Test Log window to:

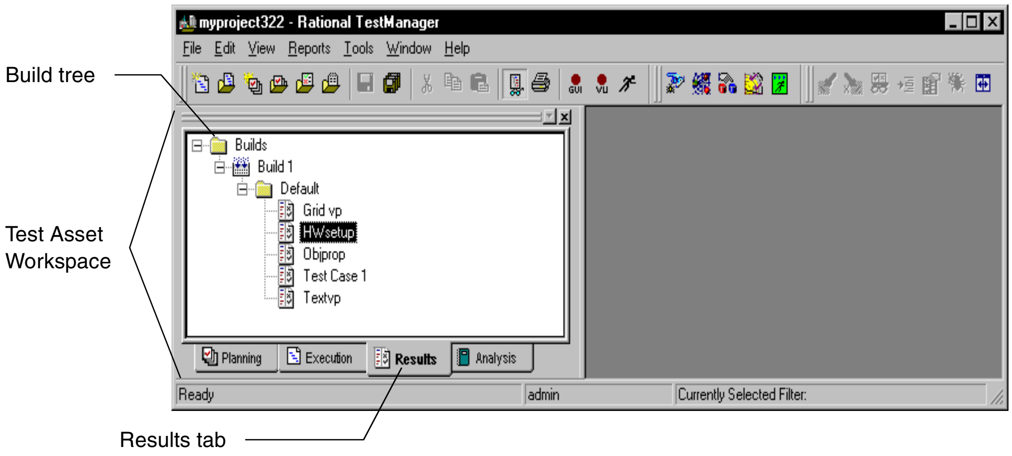
- Open a test log to view a result.
- Filter the data of a test log to view only the information you need.
- View all test cases with an unevaluated result in the Test Case Results tab of the Test Log window. This is particularly useful in evaluating the results of performance test cases. You can sort 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 test script information in the Rational ClearQuest defect form. For information about submitting a defect, see *About Submitting and Modifying Defects* on page 138.

- Open the test script of a script-based log event in the appropriate test script development tool. For example, if you create a manual test script, Rational ManualTest opens and displays the test script. If you create a custom test script type, TestManager opens the test script with the editor that you specify. You use a Test Script Console Adapter (TCSA) to specify what editor opens a test script and to manage custom test script types. For information about using a custom test script types, see *Defining Custom Test Script Types* on page 13.
- Preview or print data displayed in the active test log in the Test Log window.
- If you use Rational Robot to record test scripts, you can analyze the results in a Comparator to determine why a test may have failed. If you use Rational Quality Architect to generate test scripts from Rose models, you use the Grid Comparator to analyze results. For information about using the Comparators, see *About the Four Comparators* on page 181.

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.

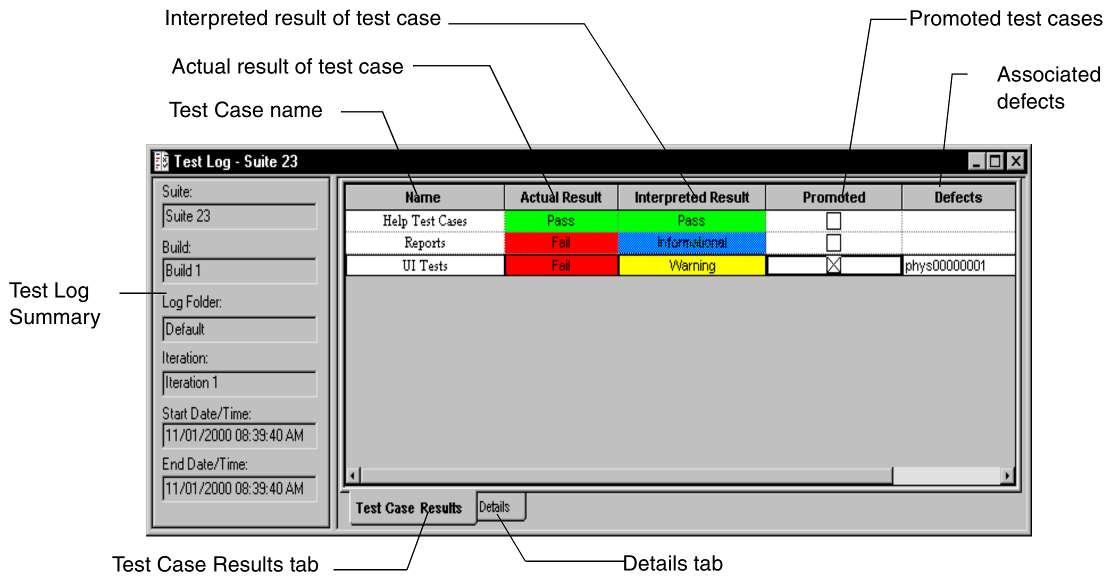
To control when the test log opens after running a test case, test script, or suite:

- 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 Window

The Test Log window of TestManager contains the Test Log Summary area, the Test Case Results tab, and the Details tab. The iteration that appears in the Test Log Summary area is associated with the build



Suite: Suite 23

Build: Build 1

Log Folder: Default

Iteration: Iteration 1

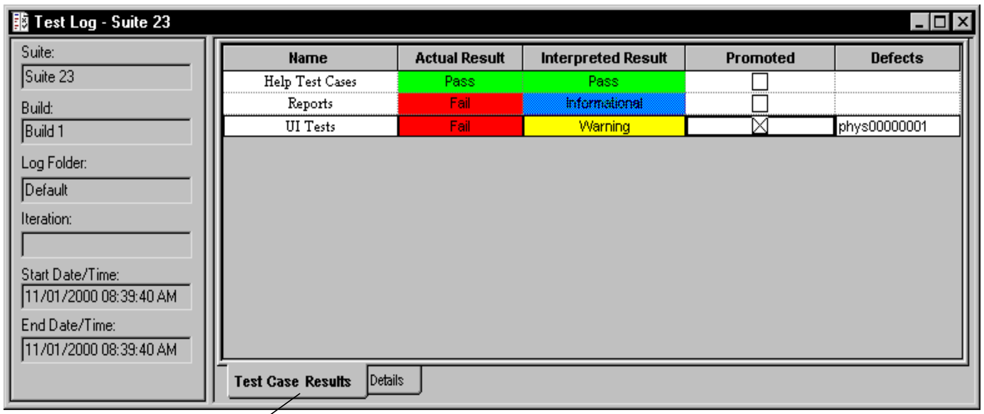
Start Date/Time: 11/01/2000 08:39:40 AM

End Date/Time: 11/01/2000 08:39:40 AM

Name	Actual Result	Interpreted Result	Promoted	Defects
Help Test Cases	Pass	Pass	☐	
Reports	Fail	Informational	☐	
UI Tests	Fail	Warning	☒	phys00000001

Test Case Results Tab

From the Test Log window, you can click the Test Case Results tab to get the overall results of each test case—did it pass or fail? The Test Case Results tab displays the results of running a test case or a suite that contains one or more test cases. If you run a test script from TestManager, ManualTest, or from Robot, even if the test script is an implementation of a test case, the Test Case Results tab will be empty. You must run a test case to get results in the Test Case Results tab.



Suite:	Suite 23
Build:	Build 1
Log Folder:	Default
Iteration:	
Start Date/Time:	11/01/2000 08:39:40 AM
End Date/Time:	11/01/2000 08:39:40 AM

Name	Actual Result	Interpreted Result	Promoted	Defects
Help Test Cases	Pass	Pass	☐	
Reports	Fail	Informational	☐	
UI Tests	Fail	Warning	☒	phys00000001

Interpreting Test Case Results

When you first open a test log and click the **Test Case Results** tab, it displays the same values for actual results and interpreted results: pass, fail, informational, or warning. An actual result is the test case result returned and logged when the test case was run. The actual result also appears in the Interpreted Result column as the default interpreted result. You may want to *interpret* a test case result if you possess additional knowledge about it and want to correct it.

For example, a test case may fail because there is a defect in the software. In this situation, the failure is valid, and you do not need to interpret the results. But a test case may fail under other situations, including:

- The application-under-test has changed, and you realize that you need to modify the related test scripts.
- There is a problem with the test automation—for example, a test script ran in the wrong order.

In either situation, the failure is misleading, and you may want to change the Interpreted Result column in the Test Case Results tab to Pass or Informational.

Promoting Test Case Results

When you *promote* a test case result, you indicate that the result is useful to your project and should be made visible to others. Before you promote a result, it appears only in the test log.

For example, assume that the required number of test cases passed your testing criteria for a build to be shipped to beta customers. You consider the test results "official" and want to include them in Test Case Results Distribution or Test Case Trend report. Therefore, you promote the result for each test case that is important for your testing criteria.

On the other hand, assume that TestManager reports a test case as failed. You discover that the computer is not plugged in, so the test case could not run—which is why TestManager reported the failure. A previous test case on that computer passed, so you promote that test case result because you want to include it in a Test Case Results Distribution or Test Case Trend report. Therefore, you would not promote the result of the failed test case, because the result is not significant—and, in fact, is misleading.

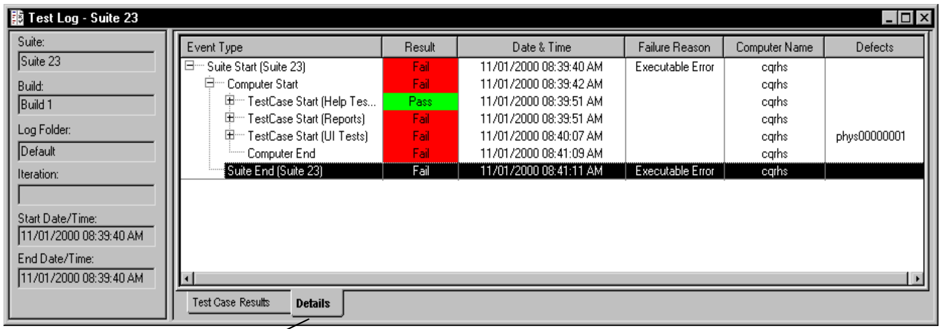
Promotion does not affect the result (pass, fail, informational, or warning). It merely indicates that the result is significant enough to appear in a Test Case Results Distribution or Test Case Trend report.

Note: You must also save the test case results when you close a test log to make the result appear in a Test Case Results Distribution report after *promoting* it.

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.

Note: Detailed test results in the Details tab cannot be promoted.



Suite: Suite 23
Build: Build 1
Log Folder: Default
Iteration:
Start Date/Time: 11/01/2000 08:39:40 AM
End Date/Time: 11/01/2000 08:39:40 AM

Event Type	Result	Date & Time	Failure Reason	Computer Name	Defects
Suite Start (Suite 23)	Fail	11/01/2000 08:39:40 AM	Executable Error	cqrhs	
Computer Start	Fail	11/01/2000 08:39:42 AM		cqrhs	
TestCase Start (Help Tes...	Pass	11/01/2000 08:39:51 AM		cqrhs	
TestCase Start (Reports)	Fail	11/01/2000 08:39:51 AM		cqrhs	
TestCase Start (UI Tests)	Fail	11/01/2000 08:40:07 AM		cqrhs	phys00000001
Computer End	Fail	11/01/2000 08:41:09 AM		cqrhs	
Suite End (Suite 23)	Fail	11/01/2000 08:41:11 AM	Executable Error	cqrhs	

Details tab

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.
  - Double-click the column heading.
- Show test cases by the following criteria:
  - Actual result with pass, fail, warning, or other.
  - Interpreted with pass, fail, warning, or other.
  - 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 window.

To view a particular event:

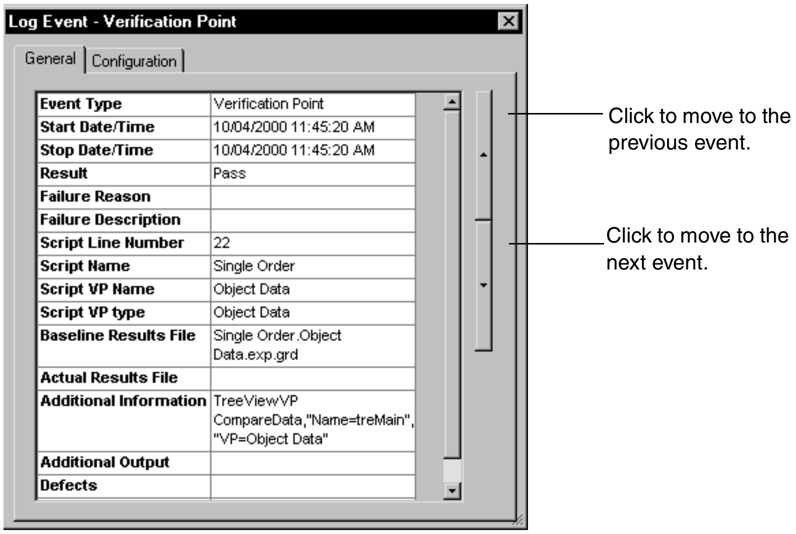
1 Click the **Details** tab in the Test Log window.

2 Click **View > Properties**.

You can keep the Log Event 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.

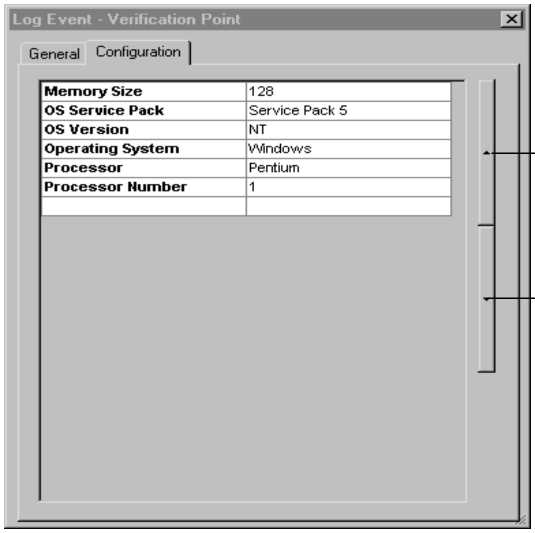
Note: If you previously used Rational Suite PerformanceStudio, the information found in the Trace and Analog reports is now available in the Log Event window.

The **General** tab of the Log Event window displays the type of event, the date and time the event was recorded, the test script name, result information (if any), and other information about a log event.


Log Event - Verification Point
General
Configuration
Click to move to the previous event.
Click to move to the next event.

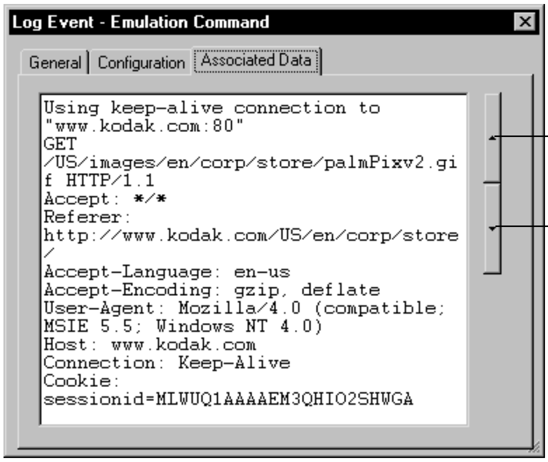
Event Type	Verification Point
Start Date/Time	10/04/2000 11:45:20 AM
Stop Date/Time	10/04/2000 11:45:20 AM
Result	Pass
Failure Reason	
Failure Description	
Script Line Number	22
Script Name	Single Order
Script VP Name	Object Data
Script VP type	Object Data
Baseline Results File	Single Order.Object Data.exp.grd
Actual Results File	
Additional Information	TreeViewVP CompareData,"Name=treMain","VP=Object Data"
Additional Output	
Defects	

The **Configuration** tab of the Log Event window displays the configuration information of the computer on which you ran the test script.


Log Event - Verification Point
General
Configuration
Click to move to the previous event.
Click to move to the next event.

Memory Size	128
OS Service Pack	Service Pack 5
OS Version	NT
Operating System	Windows
Processor	Pentium
Processor Number	1

When you add a new log event property type, a new tab appears in the Log Event window. The data, depending on how it is defined, displays in a separate tab in the Log Event window as text or as an HTML file, or in a separate application. For example, the **Associated Data** tab displays data generated by all http_request commands, TSS logging methods (such as `TSSLog.Message` in Visual Basic), and SQABasic timing statements.


Log Event - Emulation Command
General
Configuration
Associated Data
Click to move to the previous event.
Click to move to the next event.

```
Using keep-alive connection to
"www.kodak.com:80"
GET
/US/images/en/corp/store/palmPixv2.gi
f HTTP/1.1
Accept: */*
Referer:
http://www.kodak.com/US/en/corp/store
/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;
MSIE 5.5; Windows NT 4.0)
Host: www.kodak.com
Connection: Keep-Alive
Cookie:
sessionid=MLWUQ1AAAAEM3QHIO2SHWGA
```

About Log Filters

You can create a log filter to narrow down the amount of data displayed in the active Test Log window. For example, you can create a filter to display only verification points or computer starts. A log filter can make it easier to view large test logs.

Note: TestManager stores log filters in a project. All log filters are available to all users of a project but, when you apply a log filter to an active log, the results are only visible on your local system.

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 that appears in an active 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.

Note: When you apply a test log filter, the active Test Log window generates a new log and then displays the new filtered log information, which may take some time.

Viewing a Test Script

You can select any log event that is associated with a test script and view it in the tool that you used to create the test script. For example, if you create a GUI test script and open the test script from the Test Log window, the test script opens in Robot. If you create a manual test script and open the test script from the Test Log window, it opens in Rational ManualTest. If you create a custom test script type, TestManager opens the test script with the editor that you specify. You use a Test Script Console Adapter (TCSA) to specify what editor opens a test script and to manage custom test script types. For information about using a custom test script type, see *Defining Custom Test Script Types* on page 13.

Note: When you double-click an event in an open log generated from Robot under Rational Purify, Quantify, or PureCoverage, the test 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 test script:

1 Open a test log.

2 Click the **Details** tab.

3 Right-click a test script start or test 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 In the Test Asset Workspace, click the **Results** tab.

2 Right-click a test log and 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 141.

About Test Logs

Test logs record everything that happens during a test script, test case, or suite run from the time the test script, test case, or test suite 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 in a whole file.

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 then 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, 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 then 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 tester 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, 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 then click **View Virtual Tester Output File**.

To print an output file:

- From an open output file, click **File > Print**.

About Submitting 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 submit defects for any verification points that fail during playback of a recorded test script. When you submit a defect from the test log, TestManager opens a special defect form, the TestStudio defect form, and fills in several fields for you with information from the log. If you associate a test case with a test input, the test input information appears automatically in the defect form. (When you submit defects this way, TestManager does not actually start ClearQuest; it opens the defect form, which is part of ClearQuest.) You can also submit defects manually using ClearQuest, but none of the fields will be automatically filled in for you.

Note: To use ClearQuest to store defects, an administrator must first set up the ClearQuest schema, and then create or attach a ClearQuest user database as part of a Rational project. For information, see the *Administering Rational ClearQuest* manual. If you use ClearQuest Multisite, you must have ClearQuest Mastership privileges to submit or modify a defect from the Test Log window. If you do not have ClearQuest Mastership, you can modify defects manually using ClearQuest.

For your convenience, a specially designed schema, the TestStudio schema, is included with your software for defect tracking. In ClearQuest, the term *schema* refers to all attributes associated with a change-request database. This includes field definitions, field behaviors, the state transition tables, actions, and forms. For more information about ClearQuest schemas, see the Rational ClearQuest Help.

About the Rational TestStudio Schema

Th*e TestStudio schema* includes two TestStudio defect forms: one for submitting new defects, and one for modifying and tracking defect information.

Note: To use the TestStudio schema, you must select it when you create a ClearQuest user database as part of a Rational project. You can also select an existing ClearQuest user database to a project and use the TestStudio schema. If you want to attach an existing ClearQuest database that has a different schema than the TestStudio schema, you need to upgrade the schema. For information, see the *Using the Rational Administrator* manual.

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.

How to Submit and Modify a Defect

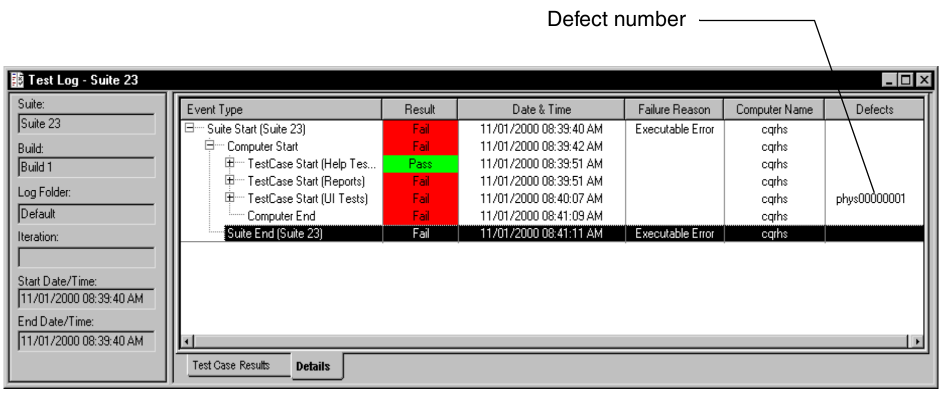
You can submit or modify defects from the Test Log window of TestManager or from ClearQuest. If you open a test log in TestManager, TestManager fills in many of the fields in the defect form. If you use ClearQuest, you must enter the fields manually.

To submit or modify 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 submit the defect.

If you submit a defect from the test log, the number of the new defect appears in the **Defect** column of the test log.



Suite: Suite 23
Build: Build 1
Log Folder: Default
Iteration:
Start Date/Time: 11/01/2000 08:39:40 AM
End Date/Time: 11/01/2000 08:39:40 AM

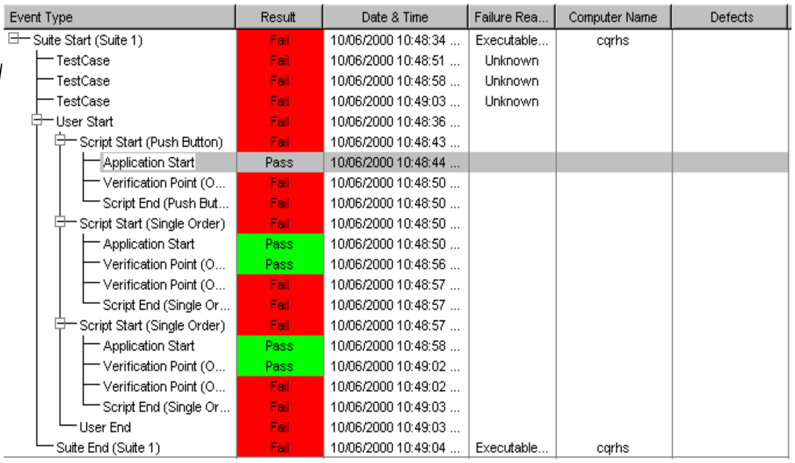
Event Type	Result	Date & Time	Failure Reason	Computer Name	Defects
Suite Start (Suite 23)	Fail	11/01/2000 08:39:40 AM	Executable Error	cqrhs	
Computer Start	Fail	11/01/2000 08:39:42 AM		cqrhs	
TestCase Start (Help Tes...	Pass	11/01/2000 08:39:51 AM		cqrhs	
TestCase Start (Reports)	Fail	11/01/2000 08:39:51 AM		cqrhs	
TestCase Start (UI Tests)	Fail	11/01/2000 08:40:07 AM		cqrhs	phys00000001
Computer End	Fail	11/01/2000 08:41:09 AM		cqrhs	
Suite End (Suite 23)	Fail	11/01/2000 08:41:11 AM	Executable Error	cqrhs	

Note: You can also submit 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	Defects
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	

When you print an active test log, your printed document will match what you see on the screen, so be sure to display the level of detail on your screen that you want in the printed document. 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.

To print an active test log:

1 Display the log that you want to print in the Test Log window.

2 Click **File > Print**.

Managing Log Event Property Types

You can manage log event property types to register additional log event properties for display 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. For more information about the information logged by this property type, see *Viewing Events Details* on page 133.

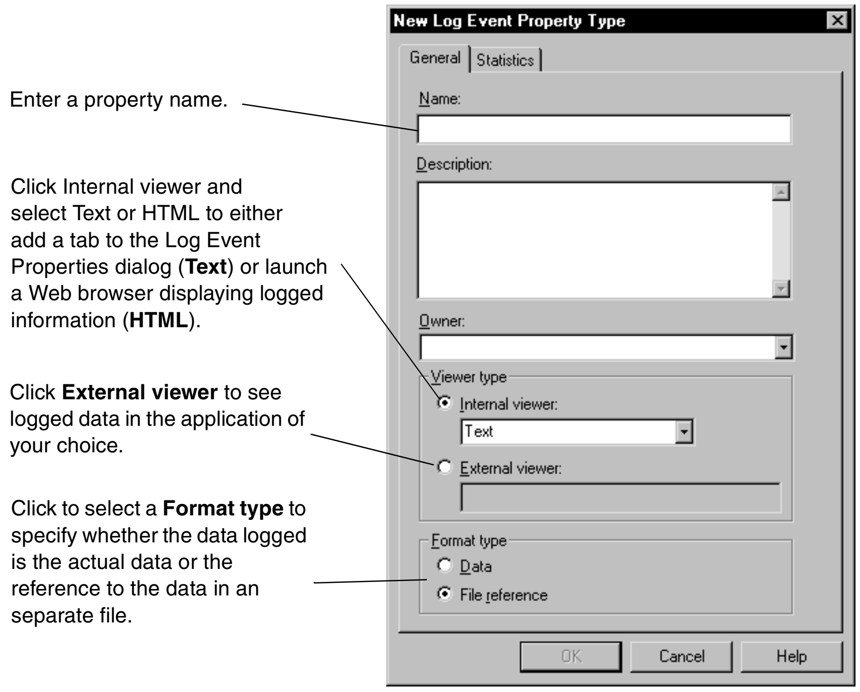
Note: Log event property types are related to the extensibility of TestManager. When integrating a new test script type, you can define additional log event properties. For more information, see the *Rational Test Extensibility Reference* manual.

TestManager supplies a default log event property type **Virtual Tester Associated Data**. This only applies to data generated by all http_request commands, TSS logging methods (such as `TSSLog.Message` in Visual Basic), and SQABasic timing statements.

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.

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 an separate file.

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 test 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 *Using the Comparators* on page 181.

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 create a file using Notepad in the recorded environment, when you play back the test script, the file already exists and the test log shows 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 test script playback or an unrelated window.

Intentional Changes to an Application Build

Revisions to the application-under-test can generate failure indications in test 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 *Using the Comparators* on page 181.

Reporting Results

TestManager provides you with a set of standard reports that you can use to analyze performance and test case results.

TestManager provides three types of reports to help you in your testing efforts:

- Test case reports — use to track the progress of planning, implementation, and execution of test cases.
- Listing reports — use to display the test assets stored in a Rational project.
- Performance testing reports — use to analyze the performance of a server under specified conditions.

About Test Case Reports

Test case reports help you track the progress of planning, implementation, and execution of test cases. These reports have multiple display formats including bar chart, stack chart, area chart, line chart, pie chart, and tree view.

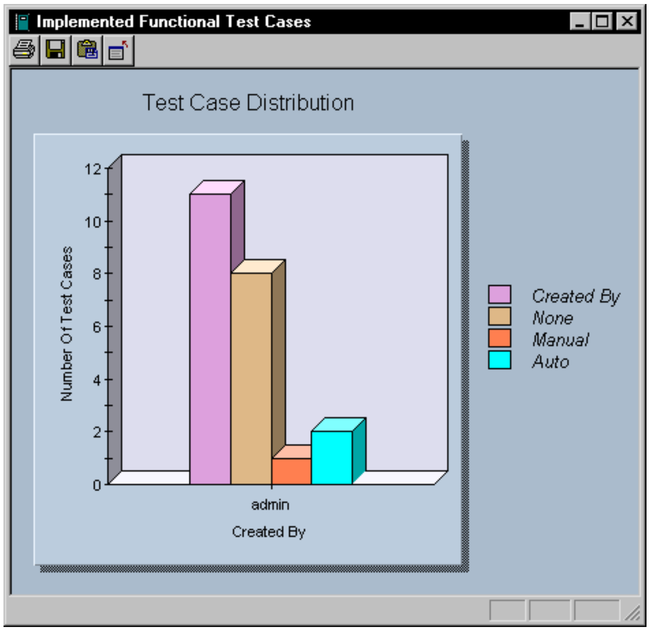
There are three types of test case reports:

- **Test Case Distribution** reports can be useful during the planning and implementation phases of a project. A test case distribution report can help you determine:
  - The number of test cases planned.
  - The number of test cases implemented with test scripts.
  - The number of test cases that have not been implemented.
  - The number of test cases implemented with a manual or automated test script.

  The Test Case Distribution reports come with two coverage reports already created for you. These test coverage reports help you track the progress of your test planning, test development, and test execution efforts.

  - **Test Planning Coverage** — a Test Case Distribution report that displays the percentage of test inputs planned and the number of test cases planned
  - **Test Development Coverage** — a Test Case Distribution report that facilitates test planning by showing you what percentage and number of test inputs have test cases with test scripts implemented that are ready to be run. This report shows you not only whether you have planned a test script for each test input, but also whether you have actually recorded the test script.

The following Test Case Distribution report shows the implemented test cases with the number of manual and automated test scripts.

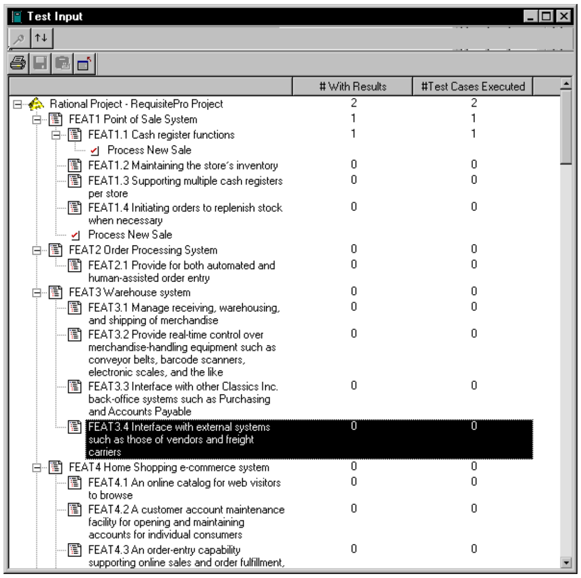


- **Test Case Results Distribution** reports can be useful during the execution phase of a project. These reports provide crucial information about the results of running a test script, test case, or test suite. With the result information, you can evaluate the quality of a specific build and the progress of the testing of that build.

  The Test Case Results Distribution reports come with one coverage report already created for you. This test coverage report helps you track the progress of your test execution efforts.

  - **Test Execution Coverage** - a Test Case Results Distribution report that displays the number of test cases planned, the number and percentage of test cases executed, the number of test cases passed, and the number of test case that failed.

The following Test Case Results report shows the number of test cases with results and the number of test cases executed for each test input.

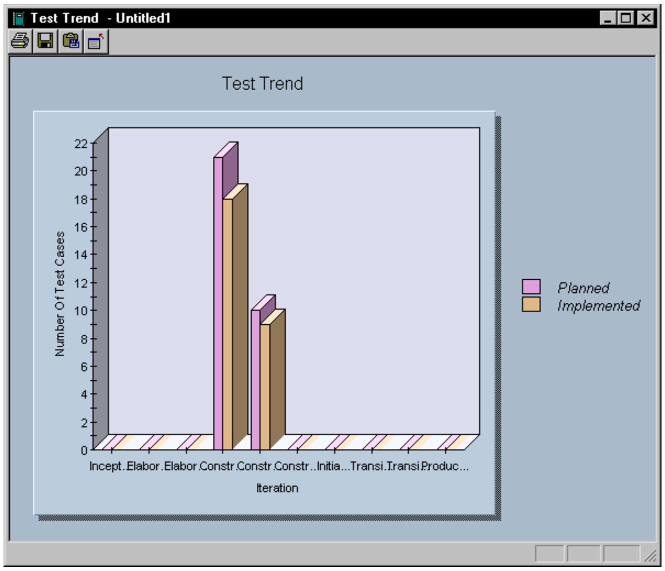


Test Input

	# With Results	#Test Cases Executed
Rational Project - RequisitePro Project	2	2
FEAT1 Point of Sale System	1	1
FEAT1.1 Cash register functions	1	1
Process New Sale		
FEAT1.2 Maintaining the store's inventory	0	0
FEAT1.3 Supporting multiple cash registers per store	0	0
FEAT1.4 Initiating orders to replenish stock when necessary	0	0
Process New Sale		
FEAT2 Order Processing System	0	0
FEAT2.1 Provide for both automated and human-assisted order entry	0	0
FEAT3 Warehouse system	0	0
FEAT3.1 Manage receiving, warehousing, and shipping of merchandise	0	0
FEAT3.2 Provide real-time control over merchandise-handling equipment such as conveyor belts, barcode scanners, electronic scales, and the like	0	0
FEAT3.3 Interface with other Classics Inc. back-office systems such as Purchasing and Accounts Payable	0	0
FEAT3.4 Interface with external systems such as those of vendors and freight carriers	0	0
FEAT4 Home Shopping e-commerce system	0	0
FEAT4.1 An online catalog for web visitors to browse	0	0
FEAT4.2 A customer account maintenance facility for opening and maintaining accounts for individual consumers	0	0
FEAT4.3 An order-entry capability supporting online sales and order fulfillment,	0	0

- **Test Case Trend** reports provide information about the number of test inputs and test cases that have been planned, developed, executed, or that have met the testing criteria over a period of time, with intervals specified by builds, iterations, or dates.

The following Test Case Trend report shows the number of test cases planned and implemented over all iterations.

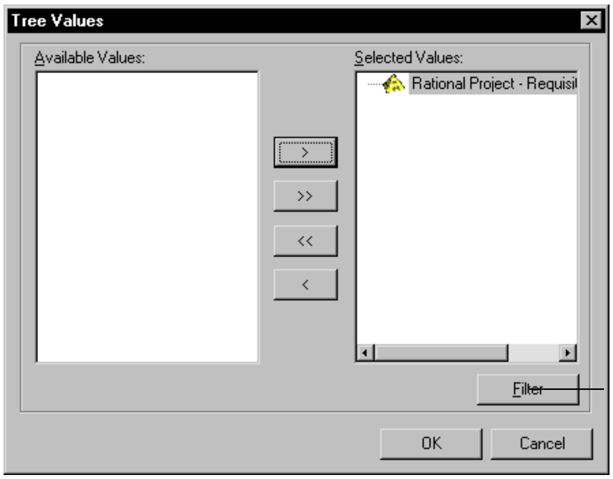


Filtering Test Input Source Information

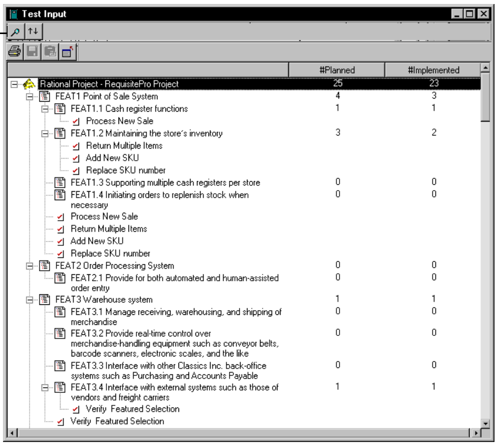
There are two ways that you can filter test input source information in Test Case Distribution or Test Case Results Distribution reports.

Note: When you filter test input source information, it may take some time for TestManager to generate a new report.

- Filter before you run a report - You can filter out unnecessary information in order to narrow down the amount of data displayed in a Test Case report.


Tree Values
Available Values:
Selected Values:
Rational Project - Requisit
>
>>
<<
<
Filter
OK
Cancel
Click to filter test input source information.

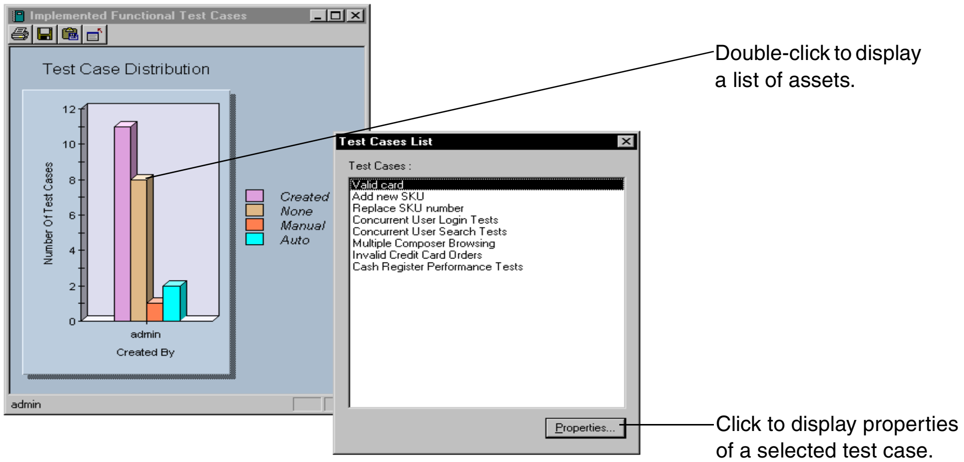
- Filter after you run a report - After you run a report, if you still need to eliminate unnecessary information, you can filter out the unnecessary information.



	#Planned	#Implemented
Rational Project - RequisitePro Project	25	23
FEAT1 Point of Sale System	4	3
FEAT1.1 Cash register functions	1	1
Process New Sale		
FEAT1.2 Maintaining the store's inventory	3	2
Return Multiple Items		
Add New SKU		
Replace SKU number		
FEAT1.3 Supporting multiple cash registers per store	0	0
FEAT1.4 Initiating orders to replenish stock when necessary	0	0
Process New Sale		
Return Multiple Items		
Add New SKU		
Replace SKU number		
FEAT2 Order Processing System	0	0
FEAT2.1 Provide for both automated and human-assisted order entry	0	0
FEAT3 Warehouse system	1	1
FEAT3.1 Manage receiving, warehousing, and shipping of merchandise	0	0
FEAT3.2 Provide real-time control over merchandise-handling equipment such as conveyor belts, barcode scanners, electronic scales, and the like	0	0
FEAT3.3 Interface with other Classics Inc. back-office systems such as Purchasing and Accounts Payable	0	0
FEAT3.4 Interface with external systems such as those of vendors and freight carriers	1	1
Verify Featured Selection		
Verify Featured Selection		

Viewing Properties of Assets in a Test Case Report

You can view the properties of an asset in any type of test case report. Double click the asset in a report to display a list of test cases associated with the particular asset as shown in the following figure:

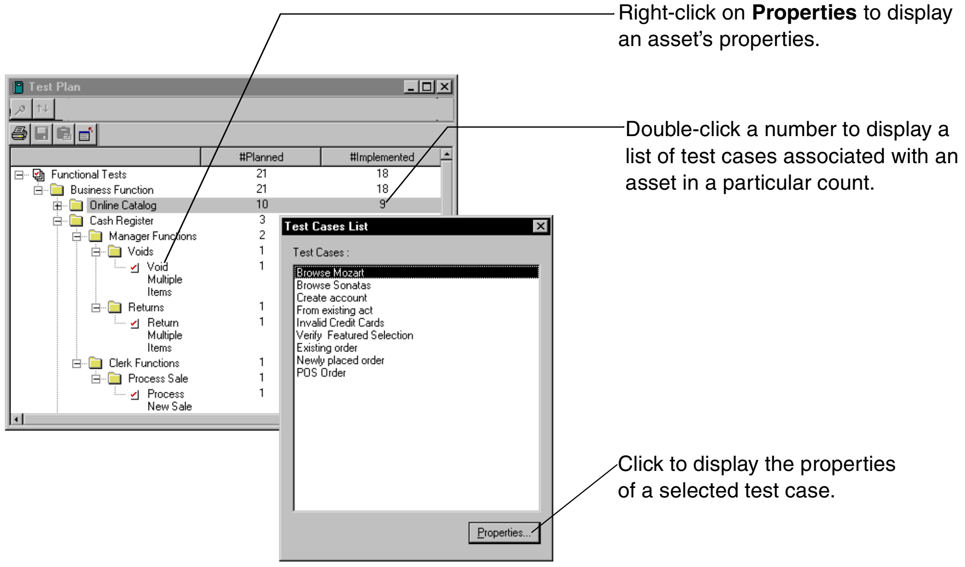


Valid card
Add new SKU
Replace SKU number
Concurrent User Login Tests
Concurrent User Search Tests
Multiple Composer Browsing
Invalid Credit Card Orders
Cash Register Performance Tests

Note: When you save a report, it creates a JPEG image. You cannot double-click a JPEG image to display a list of assets.

When you double-click the number of a counted asset in a tree view type of test case report (a Test Case Distribution or Test Case Results report created when you distribute over a test input or test plan), the assets in that particular count display.

For example, in the following figure, when you double-click **9** in the Implemented column, a list of the nine test cases that have been implemented appears in the Test Cases List window. To see the properties of any asset, select an asset, and then right-click **Properties**.


Right-click on **Properties** to display an asset's properties.

Double-click a number to display a list of test cases associated with an asset in a particular count.

Click to display the properties of a selected test case.

Test Plan

	#Planned	#Implemented
Functional Tests	21	18
Business Function	21	18
Online Catalog	10	9
Cash Register	3	
Manager Functions	2	
Voids	1	
Void Multiple Items	1	
Returns	1	
Return Multiple Items	1	
Clerk Functions	1	
Process Sale	1	
Process New Sale	1	

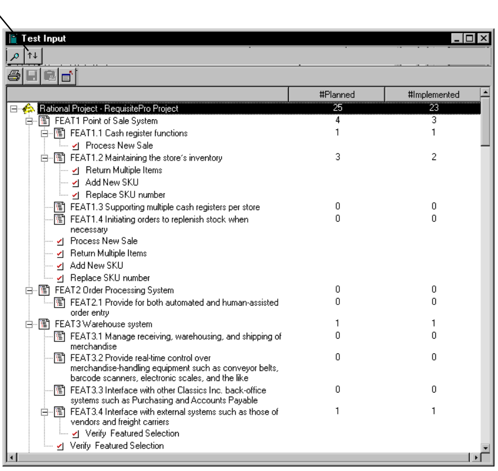
Test Cases List

Test Cases :

Browse Mozart
Browse Sonatas
Create account
From existing act
Invalid Credit Cards
Verify Featured Selection
Existing order
Newly placed order
POS Order

You can view the total number of test cases associated with an asset in a tree view type of test case report. When you roll-up a report, the total number of test cases associated with an asset appears. When you roll-down a report, the number of test cases associated with an asset appears beside each asset.

Click to roll-up or roll-down the number of test cases associated with an asset.


Test Input

	#Planned	#Implemented
Rational Project - RequisitePro Project	25	23
FEAT1 Point of Sale System	4	3
FEAT1.1 Cash register functions	1	1
Process New Sale		
FEAT1.2 Maintaining the store's inventory	3	2
Return Multiple Items		
Add New SKU		
Replace SKU number		
FEAT1.3 Supporting multiple cash registers per store	0	0
FEAT1.4 Initiating orders to replenish stock when necessary	0	0
Process New Sale		
Return Multiple Items		
Add New SKU		
Replace SKU number		
FEAT2 Order Processing System	0	0
FEAT2.1 Provide for both automated and human-assisted order entry	0	0
FEAT3 Warehouse system	1	1
FEAT3.1 Manage receiving, warehousing, and shipping of merchandise	0	0
FEAT3.2 Provide real-time control over merchandise-handling equipment such as conveyor belts, barcode scanners, electronic scales, and the like	0	0
FEAT3.3 Interface with other Classics Inc. back-office systems such as Purchasing and Accounts Payable	0	0
FEAT3.4 Interface with external systems such as those of vendors and freight carriers	1	1
Verify Featured Selection		
Verify Featured Selection		

About Listing Reports

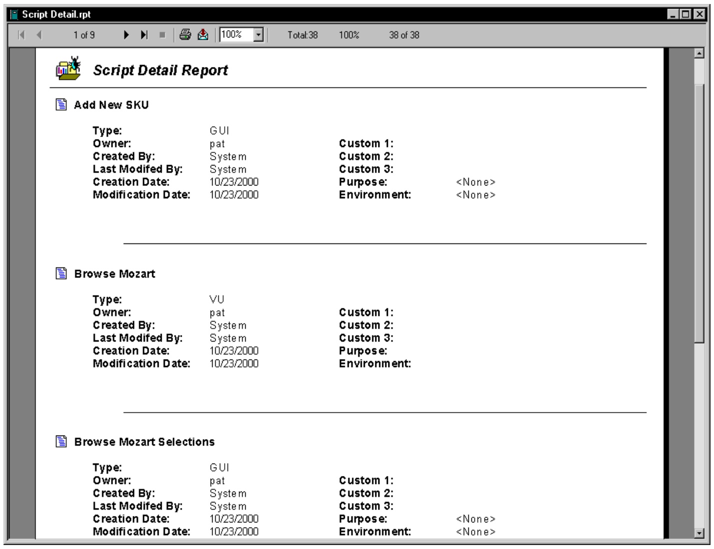
Listing reports display lists of the different test assets stored in a Rational project. TestManager includes listing reports for builds, computers, computer lists, configurations, iterations, sessions, suites, test logs, test plans, test scripts, and users.

Each listing report comes with one or more design layouts that you can use. 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 153.

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.



Script Detail.rpt

1 of 9 | 100% | Total:38 | 100% | 38 of 38

Script Detail Report

Add New SKU

```
Type:               GUI
Owner:              pat             Custom 1:
Created By:         System          Custom 2:
Last Modifed By:    System          Custom 3:
Creation Date:      10/23/2000      Purpose:        <None>
Modification Date:  10/23/2000      Environment:    <None>
```

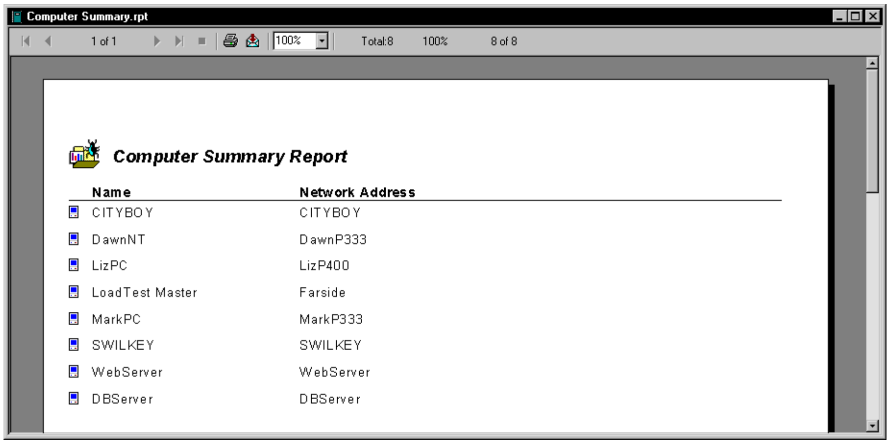
Browse Mozart

```
Type:               VU
Owner:              pat             Custom 1:
Created By:         System          Custom 2:
Last Modifed By:    System          Custom 3:
Creation Date:      10/23/2000      Purpose:
Modification Date:  10/23/2000      Environment:
```

Browse Mozart Selections

```
Type:               GUI
Owner:              pat             Custom 1:
Created By:         System          Custom 2:
Last Modifed By:    System          Custom 3:
Creation Date:      10/23/2000      Purpose:        <None>
Modification Date:  10/23/2000      Environment:    <None>
```

- Summarizes all of the computers in a project.



Computer Summary Report

Name	Network Address
CITYBOY	CITYBOY
DawnNT	DawnP333
LizPC	LizP400
LoadTest Master	Farside
MarkPC	MarkP333
SWILKEY	SWILKEY
WebServer	WebServer
DBServer	DBServer

You can also 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 listing 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 performance of a 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 customize reports.

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 *Reporting Performance Testing Results* on page 291.

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	*About Test Case Reports* on page 145
Determine the number of test cases that meet your test criteria.	Test Case Results Distribution	*About Test Case Reports* on page 145
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	*About Test Case Reports* on page 145
List the builds in your project.	Build Listing	*About Listing Reports* on page 152
List the computers in your project.	Computer Listing	*About Listing Reports* on page 152
List the computer lists in your project.	Computer List Listing	*About Listing Reports* on page 152
List the configurations in your project.	Configuration Listing	*About Listing Reports* on page 152
List the iterations in your project.	Iteration Listing	*About Listing Reports* on page 152
List the sessions in your project.	Session Listing	*About Listing Reports* on page 152

List the suites in your project.	Suite Listing	*About Listing Reports* on page 152
List the test logs in your project.	Test Log Listing	*About Listing Reports* on page 152
List the test plans in your project.	Test Plan listing	*About Listing Reports* on page 152
List the test scripts in your project.	Test Script Listing	*About Listing Reports* on page 152
List the users in your project.	User Listing	*About Listing Reports* on page 152
Display the response times, and calculate the mean, standard deviation, and percentiles for each command in a suite.	Performance	*Performance Reports* on page 308
Compare the response times measured by several Performance reports.	Compare Performance	*Compare Performance Reports* on page 312
Display individual response times and whether a response has passed or failed.	Response vs. Time	*Response vs. Time Reports* on page 317
Obtain a quick summary of which commands passed or failed.	Command Status	*Command Status Reports* on page 320
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	*Command Usage Reports* on page 322

Designing Your Own Reports

If you are an experienced Crystal Reports user, you can create your own custom reports in addition to listing reports to meet the needs of your testing team. For information see the Crystal Reports Help.

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**.

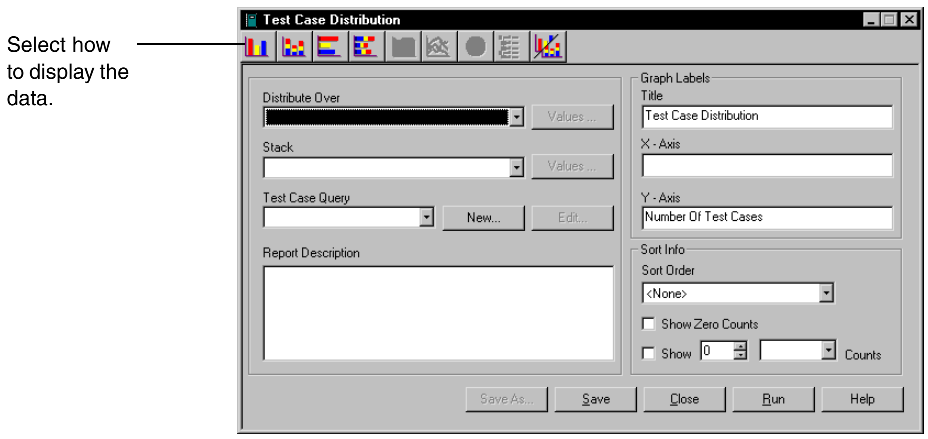
Creating Reports

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 report type, depending on the type of report you select.



Distribute Over

Stack

Test Case Query

Report Description

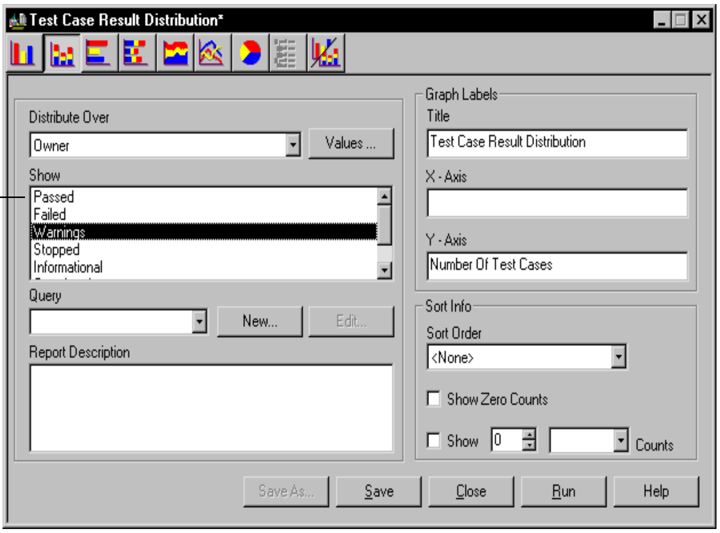
Graph Labels
Title: Test Case Distribution
X - Axis:
Y - Axis: Number Of Test Cases

Sort Info
Sort Order: <None>
☐ Show Zero Counts
☐ Show 0 Counts

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:

Select the result.



Distribute Over: Owner

Show:
Passed
Failed
Warnings
Stopped
Informational

Query:

Report Description:

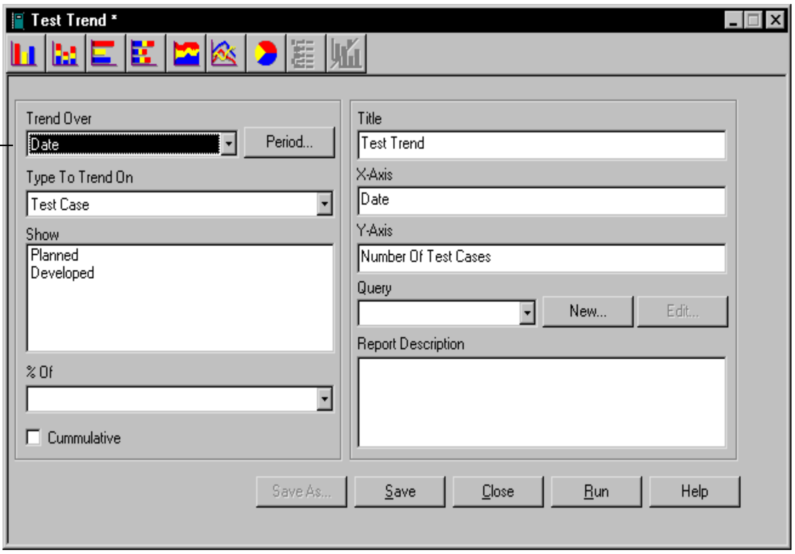
Graph Labels
Title: Test Case Result Distribution
X - Axis:
Y - Axis: Number Of Test Cases

Sort Info
Sort Order: <None>
☐ Show Zero Counts
☐ Show 0 Counts

Creating a Test Case Trend Report

When you create a test case trend report, you select the information about test cases or test inputs over several builds, iterations, or dates that you want to appear in a report.

Select the date.



Trend Over: Date

Type To Trend On: Test Case

Show:
Planned
Developed

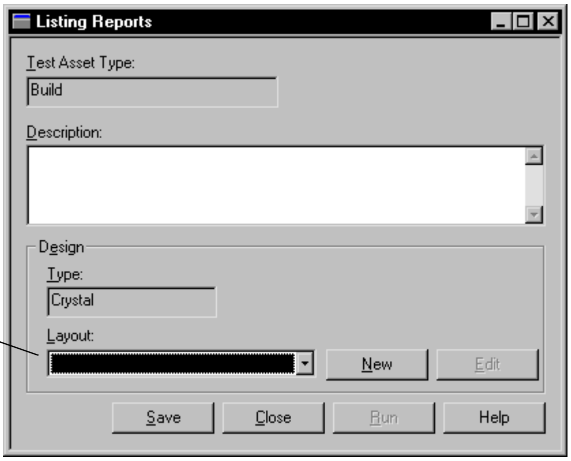
% Of:

☐ Cummulative

Title: Test Trend
X-Axis: Date
Y-Axis: Number Of Test Cases
Query:
Report Description:

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 153.



Test Asset Type: Build

Type: Crystal

Layout:

Select the design layout.

Creating Performance Testing Reports

When you create performance testing 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 testing reports, see *Reporting Performance Testing Results* on page 291.

Opening a Report

After creating and saving 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.
- Open the report from the Report bar (performance testing reports only.)

For more details about opening a report, see the TestManager Help.

Running Reports

You can run reports from:

- The Test Asset Workspace.
- The Report menu.
- The Report bar (performance testing reports only). For information, see *Running a Report from the Report Bar* on page 294.

Running a Report from the Test Asset Workspace

To run a report from the Test Asset Workspace:

- In the **Analysis** tab of the Test Asset Workspace, expand the type of report to run. Right-click the particular report and click **Run**.

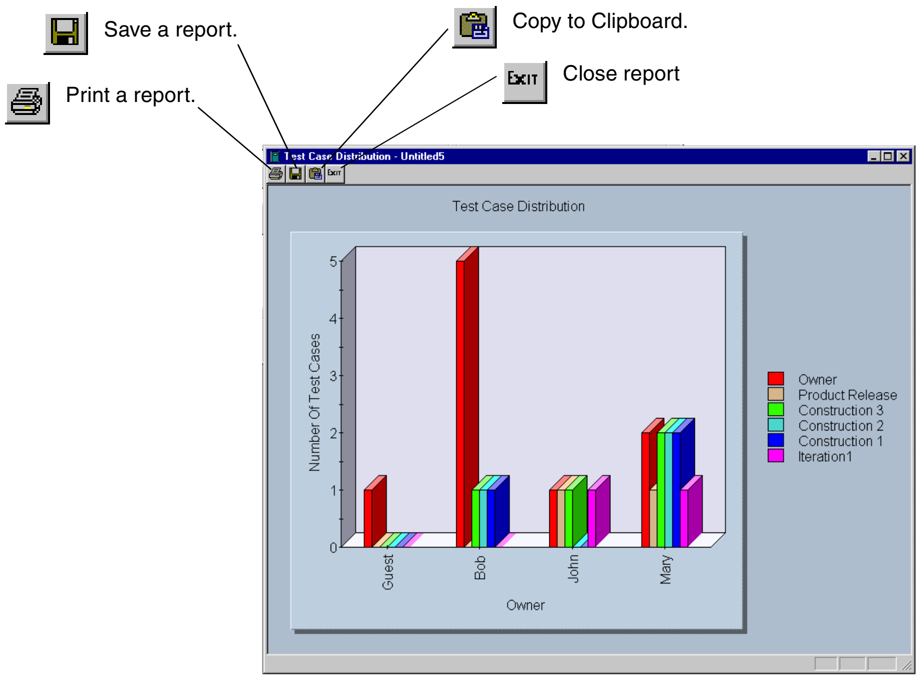
Running a Report from the Menu

To run a report from the menu:

- Click **Reports > Run**, and select the type of report to run. Select a particular report and click **OK**.

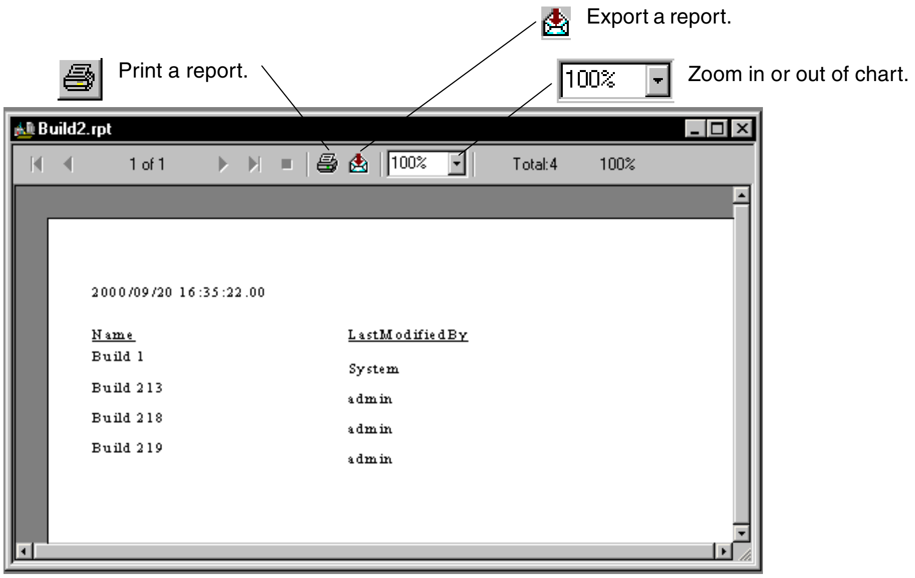
Printing, Saving, or Copying a Test Case Report

After you run a test case report, you can print, save, or copy it to the Clipboard.



Printing, Exporting, or Zooming in on 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.



2000/09/20 16:35:22.00

Name	LastModifiedBy
Build 1	System
Build 213	admin
Build 218	admin
Build 219	admin

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 Testing Report

TestManager provides pre-defined queries to narrow down the data in Test Case Distribution, Test Case Results Distribution, Test Case Trend, and Performance Testing 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 then 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

Creating Functional Testing Suites 7

This chapter describes how to create functional testing suites. It includes the following topics:

- About suites
- Inserting a computer group into a suite
- Inserting a test script into a suite
- Inserting a test case into a suite
- Inserting a suite into a suite
- Setting a precondition on a test script, test case, or suite
- Inserting a selector into a suite
- Inserting other items into a suite
- Running tests on a specific computer
- Distributing tests among different computers
- Executing suites

About Suites

A suite shows a hierarchical representation of the tasks that you want to test. It shows such items as the computers that run the test, the test script that run, and how many times each test script runs.

Through a suite, you can:

- Assign test cases to computers and rerun the test cases without having to reassign them.
- Run test scripts and test cases on the next available computer, thus speeding up your testing process.
- Set preconditions on items in a suite, which require that they complete successfully before the next item in the suite runs.

- Synchronize virtual testers.

Note: The suites in this chapter contain GUI test scripts, which are generally used for functional testing. A suite, however, can also contain VU scripts, VB scripts, or other user-defined test script types.

Inserting a Computer Group into a Suite

When you create a suite for functional testing, you first set up *computer groups*. A computer group contains the test scripts that the suite runs and declares which computers are available to the suite.

Your test can run on any Agent computer that you have defined in TestManager. If you have not defined any computers, TestManager runs your test on the Local computer. For information about defining computers, see *Defining Agent Computers and Computer Lists* on page 69.

If you simply insert a computer group and accept the defaults, TestManager creates one computer group. Use multiple computer groups only if you want to:

- Assign certain items to run on certain sets of computers. The items assigned to a group will use only the computers assigned to that group.
- Mix GUI and VU test scripts in a suite. GUI and VU test scripts must be in different computer groups.

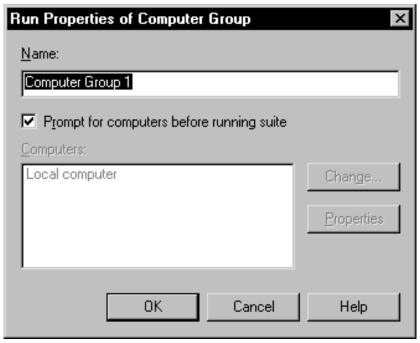
When you insert a computer group into a suite, you must decide when you will assign the computers. The method that you use applies to the entire suite. You can:

- Assign specific computers when you insert the computer group. The suite will run if any these computers are available at runtime.
- Wait until runtime to assign specific computers. TestManager prompts you for the computers when you run the suite. The suite will run if any of these computers are available at runtime.

  When you assign computers at runtime, you limit the suite to one computer group.
- Run the test on any computer that is free to run a test script. This is called *distributed functional testing*. For more information about distributed functional testing, see *Distributing Tests Among Different Computers* on page 178.

To insert a computer group into a suite:

- Click **Suite > Insert > Computer Group**.



Run Properties of Computer Group

Name: Computer Group 1

☑ Prompt for computers before running suite

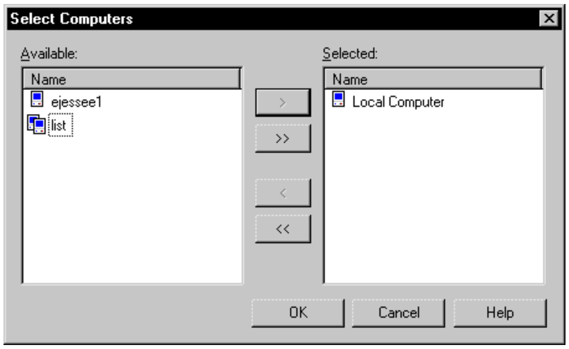
Computers:

Local computer

At this point, the virtual testers in the computer group are assigned to the Local computer.

To assign the virtual testers in the group to Agent computers:

- Clear **Prompt for computers before running suite**, and then click **Change**.



Select Computers

Available: Name
ejessee1
list

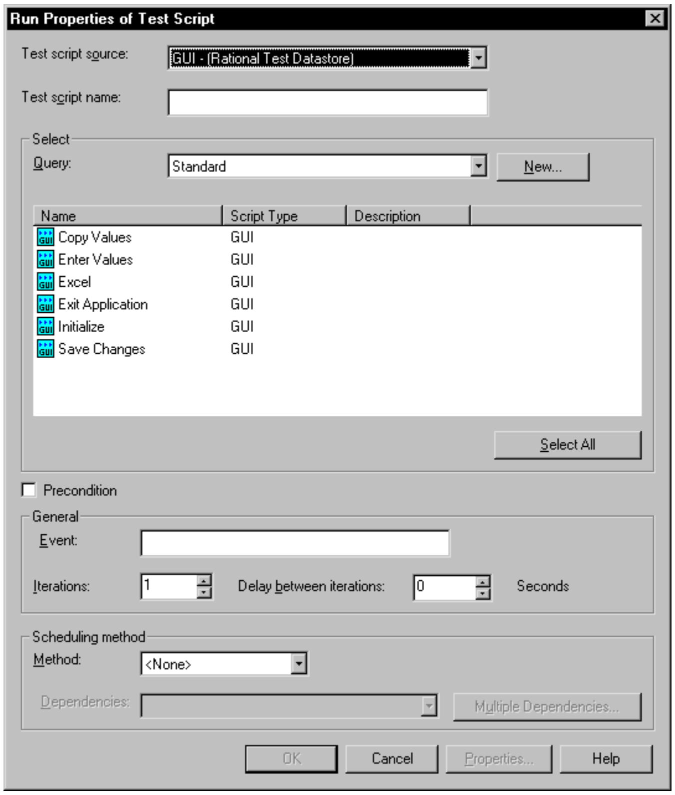
Selected: Name
Local Computer

Inserting a Test Script into a Suite

After you insert a computer group into a suite, you insert the test scripts that the computer group will run. One test script runs on one computer at a time.

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**.


Run Properties of Test Script

Test script source: GUI - (Rational Test Datastore)

Test script name:

Select

Query: Standard

Name	Script Type	Description
Copy Values	GUI	
Enter Values	GUI	
Excel	GUI	
Exit Application	GUI	
Initialize	GUI	
Save Changes	GUI	

Precondition

General

Event:

Iterations: 1 Delay between iterations: 0 Seconds

Scheduling method

Method: <None>

Dependencies:

You can set a precondition on a test script. When you set a precondition, the test script must successfully complete in order for other suite items with the same parent to run.

For example, a test script might establish a certain state in the software. You can run the test script to establish the state, and then perform a series of tests that depend on the system state.

For information about preconditions, see *Setting a Precondition on a Test Script, Test Case, or Suite* on page 174.

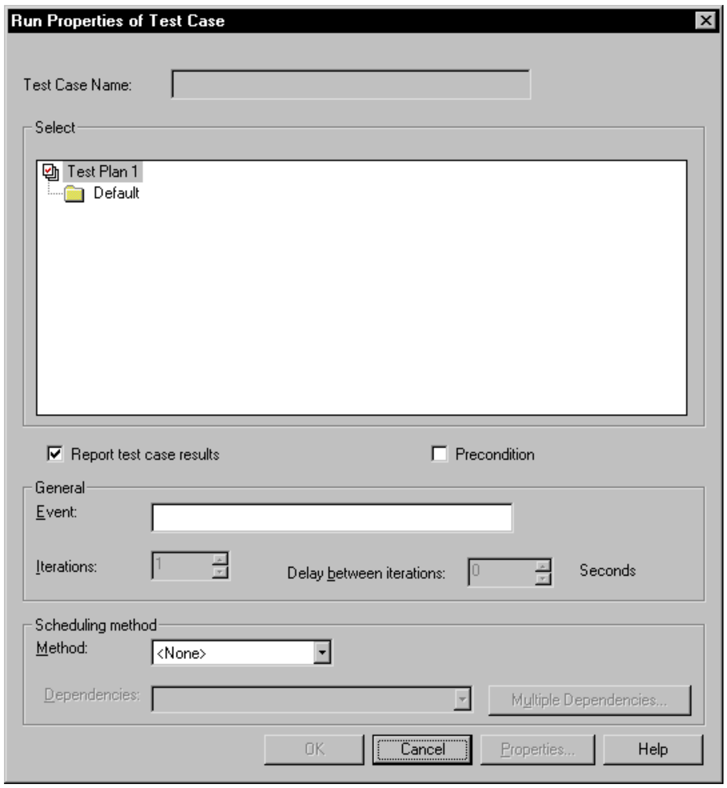
Inserting a Test Case into a Suite

Test cases let you:

- Define a test without being concerned about its implementation. Over time, the implementation can be changed, but the test case remains the same. The benefit is that you can create a suite with a test case and change the implementation (test script) without updating or maintaining the suite.
- Insert test cases into suites so that you can run multiple test cases at one time and save the set of test cases that are running together.
- Insert configured test cases to verify that a test case succeeds in multiple different environments. When you insert configured test cases in suites, TestManager automatically assigns the test cases to the appropriately configured computers.

To insert a test case into a suite:

- Click **Suite > Insert > Test Case**.



You can set a precondition on a test case. When you set a precondition, the test case must successfully complete in order for other suite items with the same parent to run. For information about preconditions, see *Setting a Precondition on a Test Script, Test Case, or Suite* on page 174.

To set a precondition on a test case:

- Right-click the test case to which to apply the precondition and select **Run Properties**.

Inserting Suites and Scenarios into Suites

Inserting suites or scenarios into a suite enable you to maintain a hierarchy of suite items. When you insert a suite or scenario into a suite, you can:

- Reuse suite items without having to duplicate them in multiple areas of a suite.
- Group suite items together so they can be shared by more than one computer group.
- Maintain your suite more easily. This is especially true if you have a complicated suite that uses many test scripts. Grouping the suite items under a suite or a scenario has the added advantage of making your suite easier to read and maintain.

In functional testing, you typically insert a suite into a suite, because inserting a suite provides more flexibility. In general, insert a suite into a suite when:

- You want to reuse a series of items in *multiple* suites. You can insert a suite into different suites.
- You want any change that you make to a suite replicated in every instance of that suite.

Insert a scenario into a suite when:

- You want to reuse a series of items in *one* suite. You cannot insert a scenario into different suites.
- You want to see the hierarchy of the suite items when you open a suite. A scenario lets you see this structure. If you insert a suite into a suite, you will have to open the child suite to see the suite items.

For example, you could create three suites, each testing a different aspect of an accounting application:

- One suite opens and edits the spreadsheets.

- One suite tests all the menus.
- One suite tests complex formulas within the spreadsheet.

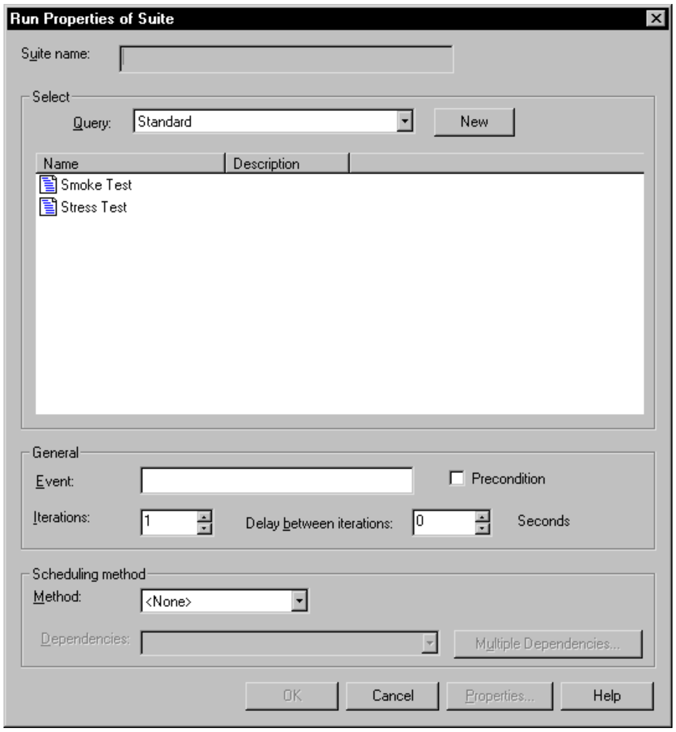
All three suites need virtual testers to open the accounting application. Yet within each suite, unique tasks need to be repeated. You can create a separate suite for opening the application and insert that suite into each of the three suites. You can then insert a scenario into each suite to represent the tasks that are unique to the suite.

Inserting a Suite into a Suite

If a suite contains computer groups, you can insert it into other suites. Inserting a suite into a suite is useful when you are creating a complex test, or when you are creating multiple tests that perform duplicate functions. You can create and check a suite, and then insert it into larger suite. You save time by not having to redefine the same test assets in each 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**.


Run Properties of Suite

Suite name:

Select

Query: Standard

Name	Description
Smoke Test	
Stress Test	

General

Event:

☐ Precondition

Iterations: 1

Delay between iterations: 0 Seconds

Scheduling method

Method: <None>

Dependencies:

You can set a precondition on a suite. When you set a precondition, the suite must successfully complete in order for other suite items with the same parent to run. For information about preconditions, see *Setting a Precondition on a Test Script, Test Case, or Suite* on page 174.

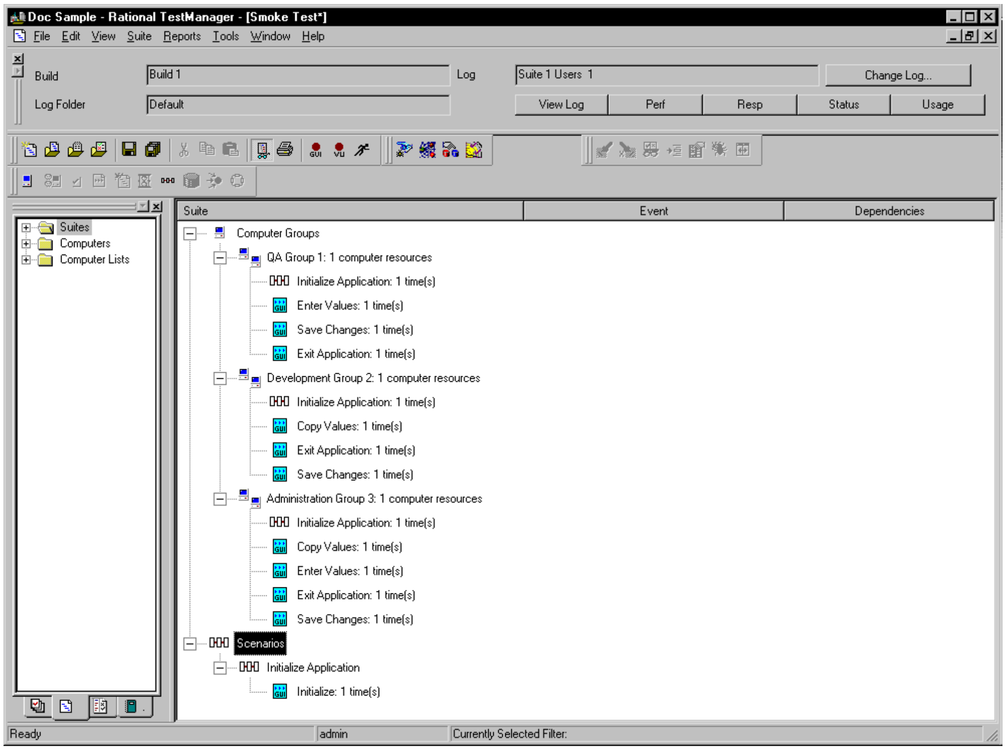
To set a precondition on a suite:

- Right-click the suite to which to set the precondition, and then select **Run Properties**.

Inserting a Scenario

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.


Doc Sample - Rational TestManager - [Smoke Test*]

File Edit View Suite Reports Tools Window Help

Build: Build 1 | Log: Suite 1 Users 1 | Change Log...

Log Folder: Default | View Log | Perf | Resp | Status | Usage

Suites
Computers
Computer Lists

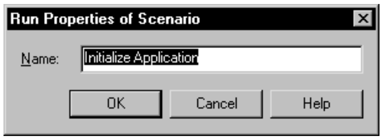
Suite	Event	Dependencies

```
Computer Groups
    QA Group 1: 1 computer resources
        Initialize Application: 1 time(s)
        Enter Values: 1 time(s)
        Save Changes: 1 time(s)
        Exit Application: 1 time(s)
    Development Group 2: 1 computer resources
        Initialize Application: 1 time(s)
        Copy Values: 1 time(s)
        Exit Application: 1 time(s)
        Save Changes: 1 time(s)
    Administration Group 3: 1 computer resources
        Initialize Application: 1 time(s)
        Copy Values: 1 time(s)
        Enter Values: 1 time(s)
        Exit Application: 1 time(s)
        Save Changes: 1 time(s)
Scenarios
    Initialize Application
        Initialize: 1 time(s)
```

Ready | admin | Currently Selected Filter:

To create a new scenario:

- From the Scenarios section of the suite, click **Suite > Insert > Scenario**.

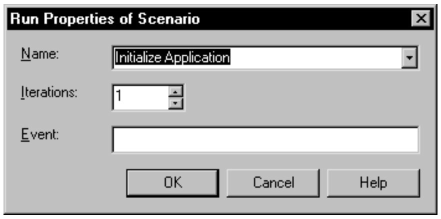

Run Properties of Scenario

Name: Initialize Application

OK | Cancel | Help

To insert a scenario into a suite:

- Click where you want to place the scenario, then click **Suite > Insert > Scenario**.


Run Properties of Scenario

Name: Initialize Application

Iterations: 1

Event:

OK | Cancel | Help

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 test cases, suites, and selectors.

Setting a Precondition on a Test Script, Test Case, or Suite

When you insert a test script, test case, or suite into a suite, you can specify that successful completion of that item is a *precondition* for the remainder of that suite sequence. The item must pass for the remaining suite items at the same level to run.

For example, suppose a suite includes two suites, each of which contains an initialization test script and several test cases. If you set a precondition on the initialization test script and the test script fails, TestManager skips all remaining test cases within that suite *only*. The suite run resumes at the beginning of the second suite.

To set a precondition:

- Right-click the test script, suite, or test case to which to set the precondition, and select **Run Properties**.

Preconditions apply only to the specific instance of the test script, test case, or suite. For example, if you insert a test script multiple times, and you want to set a precondition on all instances of the test script, you must set the precondition for each test script.

Inserting a Selector into a Suite

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 to execute, 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.

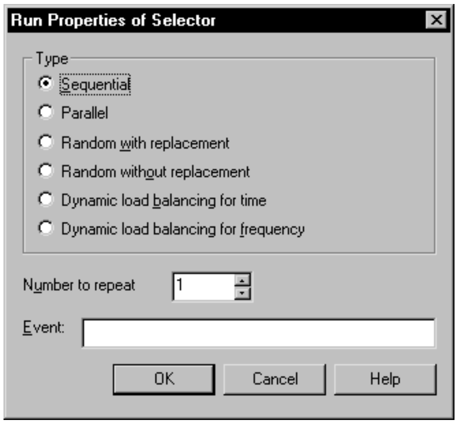
The following list explains the types of selectors that are used in functional testing. The other types of selectors are used in performance testing.

- **Sequential** – Runs each suite item in the order in which it appears in the suite. This is the default.
- **Parallel** – Distributes each suite item to any computer that is available. This selector is used in distributed functional testing. The suite items are parceled out in order, based on which computers 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. For example, assume Script A runs for 10 iterations, and Script B runs for only one iteration. The number of iterations does not affect the way the scripts are distributed.

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**.


Run Properties of Selector

Type
(● Sequential) Parallel / Random with replacement / Random without replacement / Dynamic load balancing for time / Dynamic load balancing for frequency

Number to repeat: 1

Event:

Inserting Other Items into a Suite

The items described in the following sections are generally used in performance tests. You may occasionally use these items in functional tests.

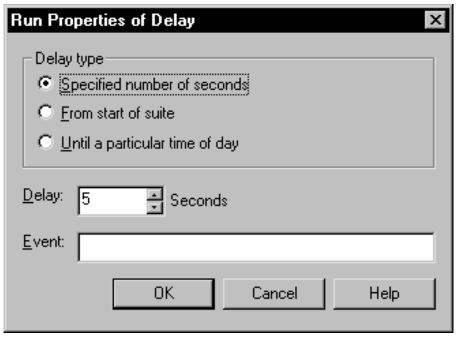
Inserting a Delay

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

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.

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**.



Delay type: [✓ Specified number of seconds] From start of suite / Until a particular time of day

Delay: 5 Seconds

Event:

Inserting a Synchronization Point

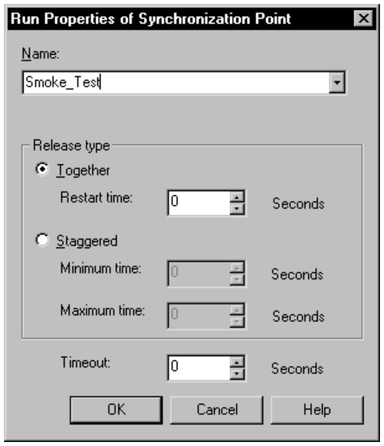
A *synchronization point* lets you coordinate the activities of a number of virtual testers by pausing the execution of each virtual tester at a particular point. Synchronization points are used primarily in performance testing suites. However, you might use synchronization points in a functional testing suite to test what happens when two virtual testers access a file at the same time.

A synchronization point is in effect until one of the following events occurs:

- All virtual testers associated with the synchronization point arrive at the synchronization point.
- A timeout period is reached before all virtual testers arrive at the synchronization point.
- You manually release the virtual testers while monitoring the suite.

To insert a synchronization point into a suite:

- Click **Suite > Insert > Synchronization Point**.



Name: Smoke_Test

Release type:
- (●) Together — Restart time: 0 Seconds
- ( ) Staggered — Minimum time: 0 Seconds; Maximum time: 0 Seconds

Timeout: 0 Seconds

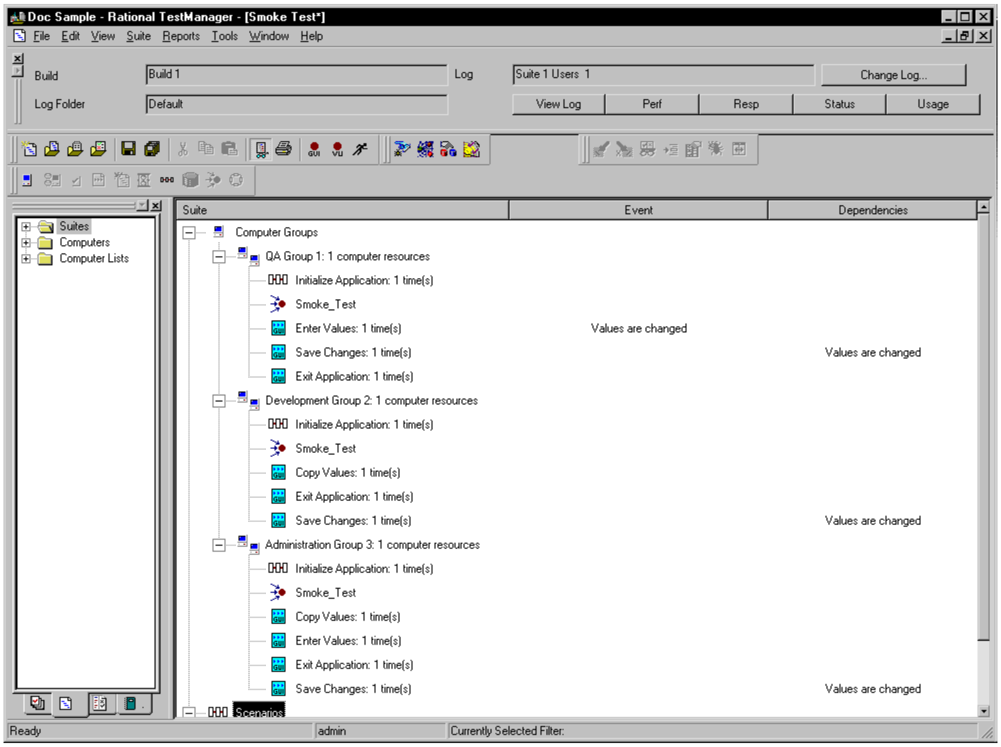
For more information about how synchronization points work, see *How Synchronization Points Work* on page 249.

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 changed. 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 an event.

The following suite shows virtual testers waiting until the first virtual tester changes values:


Doc Sample - Rational TestManager - [Smoke Test*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Ready
admin
Currently Selected Filter:

Build: Build 1
Log Folder: Default
Log: Suite 1 Users 1

Suite	Event	Dependencies
Computer Groups		
QA Group 1: 1 computer resources		
Initialize Application: 1 time(s)		
Smoke_Test		
Enter Values: 1 time(s)	Values are changed	
Save Changes: 1 time(s)		Values are changed
Exit Application: 1 time(s)		
Development Group 2: 1 computer resources		
Initialize Application: 1 time(s)		
Smoke_Test		
Copy Values: 1 time(s)		
Exit Application: 1 time(s)		
Save Changes: 1 time(s)		Values are changed
Administration Group 3: 1 computer resources		
Initialize Application: 1 time(s)		
Smoke_Test		
Copy Values: 1 time(s)		
Enter Values: 1 time(s)		
Exit Application: 1 time(s)		
Save Changes: 1 time(s)		Values are changed
Scenarios		

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: This 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.

Distributing Tests Among Different Computers

You may want to distribute your test scripts among different computers. For example, your tests may not be designed for a specific computer. Or you may want to run your tests on a group of computers, so that they can complete as fast as possible. With TestManager, you can run many computers concurrently and distribute your tests among these computers. This enables you to speed up the testing process.

To distribute your tests among different computers, follow these steps:

- When you insert computer groups into a suite, click **Change** 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 a Computer Group into a Suite* on page 166.
- After you have inserted your computer groups, insert a Parallel selector. The test scripts that you insert under the selector will be continuously sent out to the next available computer. Of course, the test 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 into a Suite* on page 174.

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 virtual tester in. A modular script for each virtual tester task. A test script to perform any cleanup work and then shut down the application.	One computer group that logs the users in. One computer group that contains a Parallel selector and modular scripts that run on any computer. One computer group that shuts down the application.	Test log report to show whether all virtual testers 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 test scripts should be modular.

Executing Suites

After you have created and saved your suite, you can:

- Check the suite for errors. To do this, open the suite, and then click **Suite > Check Suite**.
- Check the status of Agent computers. To do this, open the suite, and then click **Suite > Check Agents**.
- Control the runtime information of the suite. To do this, open the suite, and then click **Suite > Edit Runtime**.
- Control how the suite terminates. To do this, open the suite, and then click **Suite > Edit Termination**.
- Run the suite. To do this, open the suite, and then click **File > Run Suite**.
- Monitor the progress of a suite that is running. For information about monitoring suites, see *Monitoring Suites* on page 103.

Using the Comparators 8

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

After you play back a test case, test script, or suite, TestManager writes the results to a test log that appears in the Test Log window of TestManager. The test log tells you whether each test case, test script, or suite passed or failed.

You can get further details by clicking the Details tab of the test log. When you double-click a failed verification point in the test log, the appropriate Comparator for that verification point appears. You can view and compare data captured using the Comparators to pinpoint the exact reason that a verification point failed.

Note: You can use the Comparators only with test scripts containing verification points created with Rational Robot.

When you record a test script that includes a verification point, Robot creates a *Baseline file* that contains the data that you captured.

When you play back a test script, Robot compares the properties in the Baseline file with the properties in the application-under-test. If the comparison fails, Robot saves the data that caused the failure to an *Actual file*. The results of the verification point appear in a test log.

The four Comparators are as follows:

- 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:

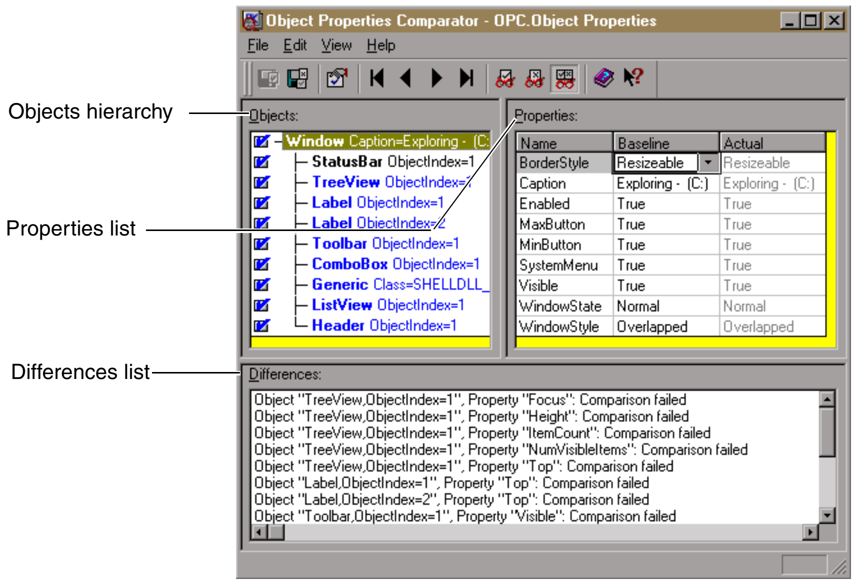
- Review, compare, and analyze the differences between the Baseline file and the Actual file.
- View or edit the Baseline file for an Object Properties verification point.

To start the Object Properties Comparator from the test log window, see *Starting a Comparator* on page 182.

The Main Window

The main window of the Object Properties Comparator contains the Objects hierarchy, the Properties list, and the Differences list.

ž



Objects:

- Window Caption=Exploring - (C:
  - StatusBar ObjectIndex=1
  - TreeView ObjectIndex=1
  - Label ObjectIndex=1
  - Label ObjectIndex=2
  - Toolbar ObjectIndex=1
  - ComboBox ObjectIndex=1
  - Generic Class=SHELLDLL_
  - ListView ObjectIndex=1
  - Header ObjectIndex=1

Properties:

Name	Baseline	Actual
BorderStyle	Resizeable	Resizeable
Caption	Exploring - (C:)	Exploring - (C:)
Enabled	True	True
MaxButton	True	True
MinButton	True	True
SystemMenu	True	True
Visible	True	True
WindowState	Normal	Normal
WindowStyle	Overlapped	Overlapped

Differences:

```
Object "TreeView,ObjectIndex=1", Property "Focus": Comparison failed
Object "TreeView,ObjectIndex=1", Property "Height": Comparison failed
Object "TreeView,ObjectIndex=1", Property "ItemCount": Comparison failed
Object "TreeView,ObjectIndex=1", Property "NumVisibleItems": Comparison failed
Object "TreeView,ObjectIndex=1", Property "Top": Comparison failed
Object "Label,ObjectIndex=1", Property "Top": Comparison failed
Object "Label,ObjectIndex=2", Property "Top": Comparison failed
Object "Toolbar,ObjectIndex=1", Property "Visible": Comparison failed
```

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 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 columns, 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 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. Since only the Current Baseline can 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.

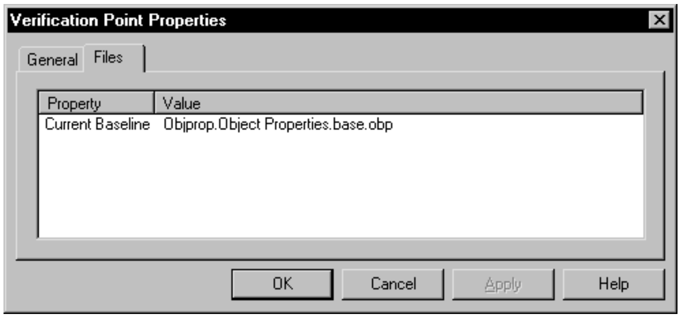
You can also select a description in the Differences list to highlight that failure in the Properties list.

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.



Property	Value
Current Baseline	Objprop.Object Properties.base.obp

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 then 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 remove the property 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 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.

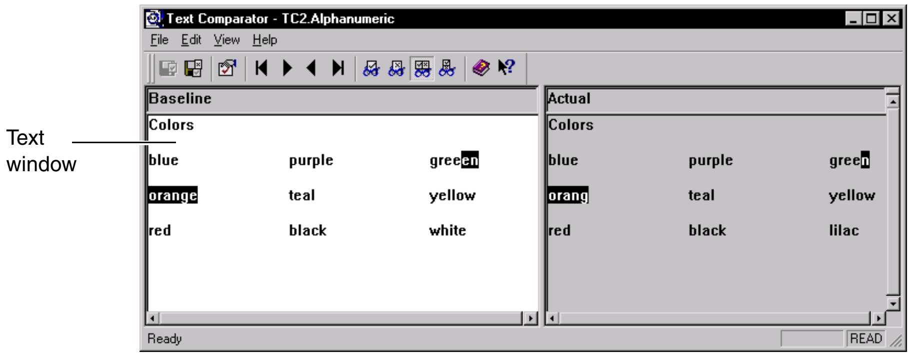
You can use the Text Comparator to:

- Review, compare, and analyze the differences between the Baseline file and the Actual file.
- View or edit the Baseline file for an Alphanumeric verification point.

To start the Text Comparator, see *Starting a Comparator* on page 182.

The Main Window

The main window of the Text Comparator contains the Text window.



Colors

blue	purple	green
orange	teal	yellow
red	black	white

Colors

blue	purple	green
orang	teal	yellow
red	black	lilac

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 for 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 the comparator finds a difference using the **View** commands, the difference between the Baseline file and the Actual file appears in reverse color.

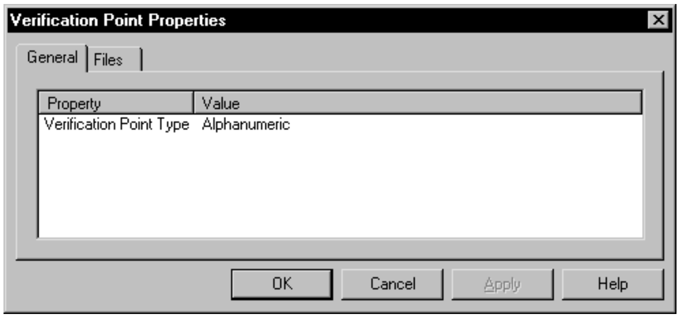
The Alphanumeric verification point stores the specified verification method as part of the test script command. For data files created by the Alphanumeric verification point, the Comparator assumes a case-sensitive comparison, regardless of how it was recorded. For numeric data, the Comparator assumes 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.



Property	Value
Verification Point Type	Alphanumeric

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 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

Rational Quality Architect also uses the grid comparator to display verification point information.

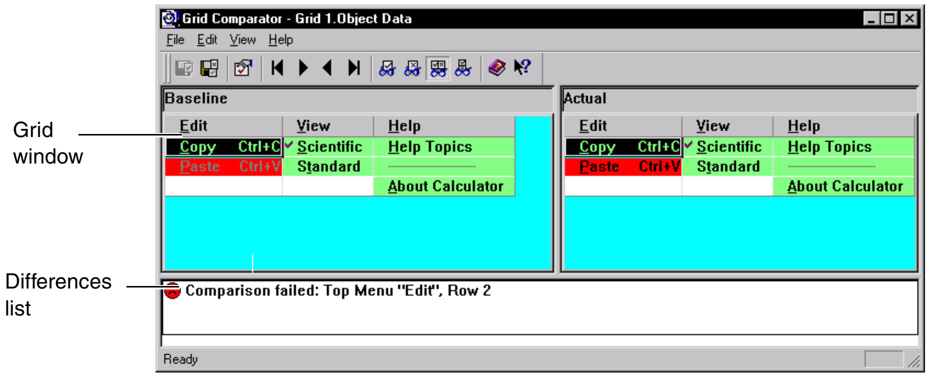
You can use the Grid Comparator to:

- Review, compare, and analyze the differences between the Baseline file and the Actual file.
- View or edit the Baseline file for a verification point.

To start the Grid Comparator, see *Starting a Comparator* on page 182.

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.



Baseline

Edit	View	Help
Copy Ctrl+C	✓ Scientific	Help Topics
Paste Ctrl+V	Standard	
		About Calculator

Actual

Edit	View	Help
Copy Ctrl+C	✓ Scientific	Help Topics
Paste Ctrl+V	Standard	
		About Calculator

Comparison failed: Top Menu "Edit", Row 2

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 194.

Differences List

The Differences list displays the Actual items that failed during playback. This list shows the reasons why a verification point failed and displays icons to graphically illustrate the type of failure. 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, 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.

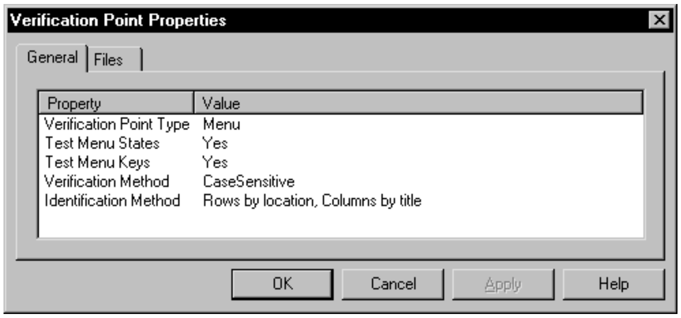
If the data displayed in the grid is larger than the window, you can use the scroll bars to view other areas of the data, or you can resize the window.

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**.



Property	Value
Verification Point Type	Menu
Test Menu States	Yes
Test Menu Keys	Yes
Verification Method	CaseSensitive
Identification Method	Rows by location, Columns by title

The Verification Point Properties dialog box shows:

- Verification point type
- Test menu states
- Test menu keys
- Verification method
- Identification method
- Name of the baseline file
- Name of the actual file

Using Keys to Compare Data Files

You can use 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 might compare the selected record to a record that contains similar values but that is not the record 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 file.

For step-by-step instructions, 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

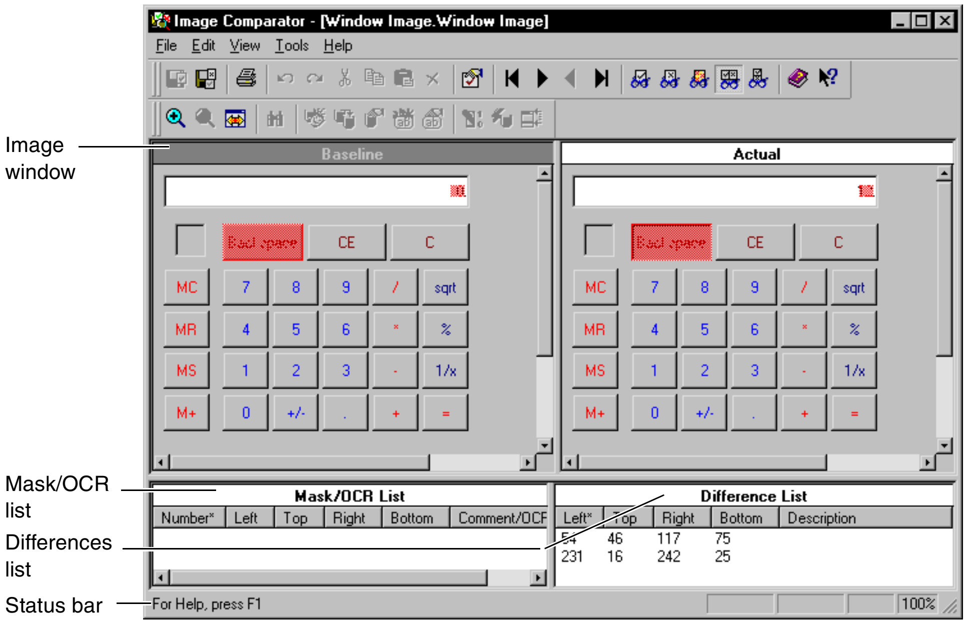
You can use the Image Comparator to:

- 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 182.

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.



Mask/OCR List

Number*	Left	Top	Right	Bottom	Comment/OCR

Difference List

Left*	Top	Right	Bottom	Description
54	46	117	75	
231	16	242	25	

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 202.

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 that are 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, Differences List. The Mask/OCR List part of the Image Comparator 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.

These measurements work in the same way as they work 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 o f the status bar, 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, the Comparator 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 determine whether more or fewer difference regions are 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 useful 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

Planning Performance Tests 9

This chapter introduces performance testing using Rational TestManager. It includes the following topics:

- About performance testing
- Rational TestManager and performance testing
- Planning performance tests
- Implementing performance tests
- Examples of performance tests
- Analyzing performance results

About Performance Testing

A *performance test* helps you determine whether a multi-client system is performing to defined standards under varying workloads 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.

Early in the development lifecycle—in the elaboration iterations—performance tests focus on identifying and eliminating architecture-related performance bottlenecks. Later in the lifecycle—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 workload 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 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.

You can also use virtual testers to measure client response times to get an end-to-end response. This is more indicative of what a real user experiences when significant client processing or screen-painting time is associated with the user activity that you are measuring.

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** – 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).

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 a 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

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 to 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 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.

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.

Contention Tests

Contention testing involves executing virtual testers on one or more computers to simulate an actual user environment. For example, you might have virtual 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. With TestManager, you can conduct multi-computer tests in which virtual testers wait for conditions to be satisfied on their own computers, or on other computers, before they continue running. For example, you can have one virtual tester on a computer add a record to a database, and have a second virtual tester on another computer pause until the first virtual tester finishes adding the record. The second virtual tester can then read the record.

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 workload 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 a 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.

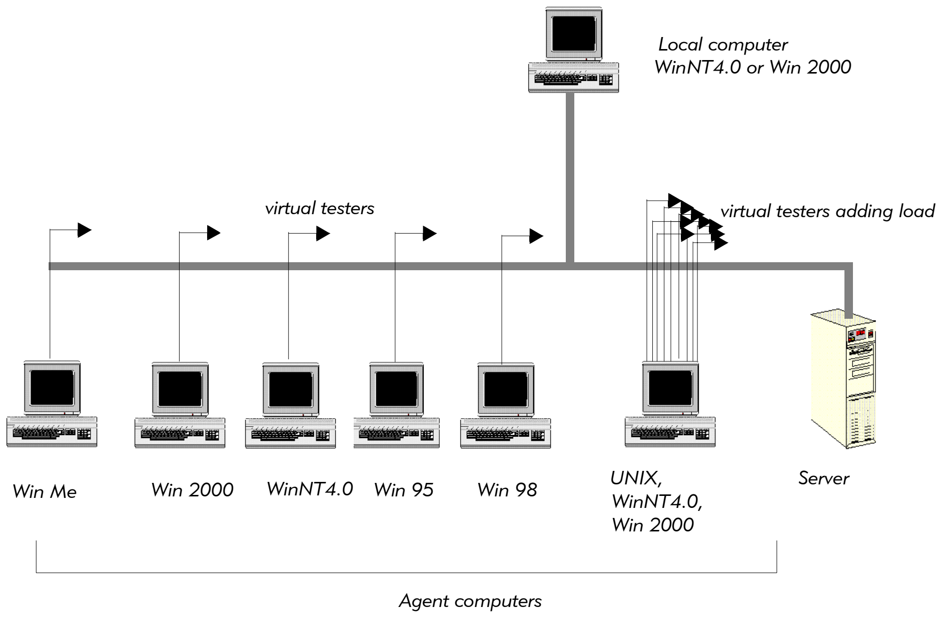
Because TestManager lets you play back the activities of multiple virtual testers on a single computer, you can run tests involving hundreds or thousands of virtual testers on just a few computers—or on one computer.

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

The following figure illustrates a typical TestManager configuration:



Recording and Playing Back Test Scripts

Performance tests that are run from TestManager play back test scripts. A 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.

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 workload.

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 workload 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 *Implementing Tests as Suites* on page 67.

- **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 227.

- **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 accesses 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 241. 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 98.

- **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 299.

- **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 workload 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
A test script to initialize the database. A test script to log virtual testers in. A test script for each virtual tester task: ▪ adding records ▪ deleting records ▪ querying the database ▪ running payroll reports	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 many virtual testers. This group logs in and waits until the event is set. It then executes the scenario. A scenario that contains: ▪ a selector to randomly select a test script ▪ a test script for each virtual tester task	A test log to show whether all virtual testers in the suite successfully ran to completion. A Command Status report to show whether the server completed its requests successfully. Performance reports for each suite run: 300, 400, 500, 600, and 700 virtual testers. A Compare Performance report comparing the output of all five Performance reports.

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 workload.

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 workload, 300 virtual testers perform transactions repeatedly.

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

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: ▪ adding records ▪ deleting records ▪ querying the database ▪ running payroll reports	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. 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. One scenario that contains: ▪ a selector to randomly select a test script ▪ a test script for each virtual tester task	A test log to show whether all virtual testers in the suite successfully ran to completion. A Command Status report to show whether the server completed its requests successfully. Two Performance reports: ▪ One report on the time period from the start of the run until 2 hours have passed. ▪ One report on the time period from 2 hours until the end of the run. A Compare Performance report comparing the output of both Performance reports.

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: ▪ adding records ▪ deleting records ▪ querying the database ▪ running payroll reports	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: ▪ a selector to randomly select a test script ▪ a test script for each virtual tester task	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: ▪ adding records ▪ deleting records ▪ querying the database ▪ running payroll reports	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: ▪ a selector to randomly select a test script ▪ a test script for each virtual tester task	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 successful suite run, 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 workload. 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 *Reporting Performance Testing Results* on page 291.

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 301.

Designing Performance Testing Suites 10

This chapter describes how to design performance testing suites. It includes the following topics:

- About suites
- Creating a suite from a Robot session
- Inserting user groups into a suite
- Inserting test scripts into a suite
- Setting a precondition on a test script, test case, or suite
- Inserting other items into a suite
- Using events and dependencies to coordinate execution
- Executing suites

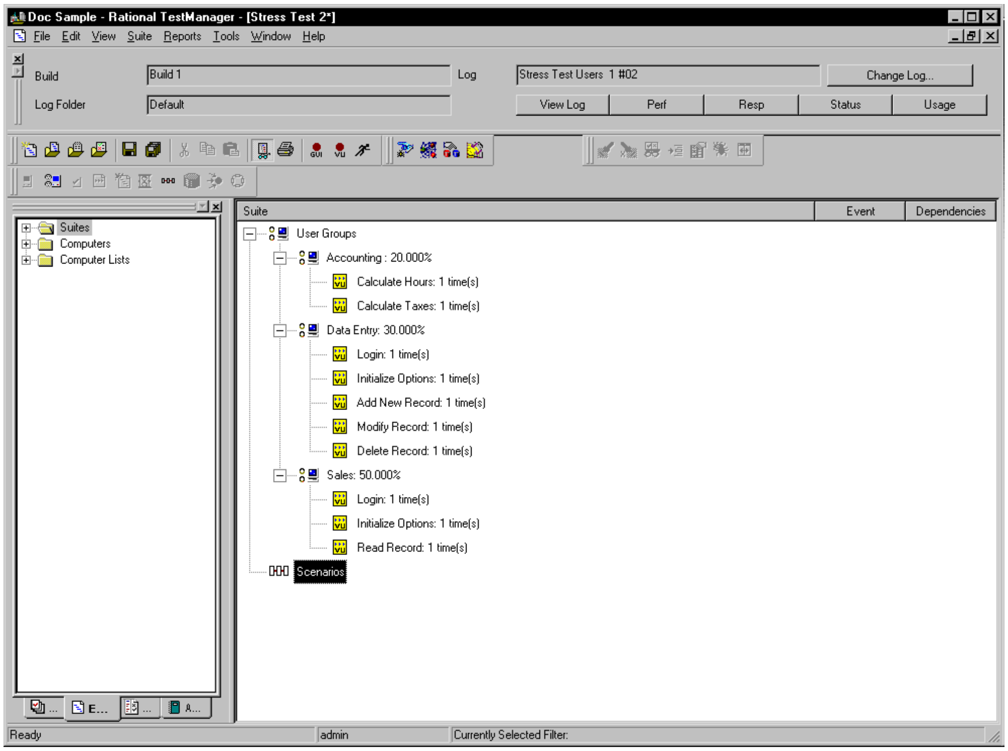
About Suites

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.


Doc Sample - Rational TestManager - [Stress Test 2*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suite
Event
Dependencies
Ready

Build: Build 1
Log Folder: Default
Log: Stress Test Users 1 #02

```
Suites
Computers
Computer Lists
```

```
User Groups
    Accounting : 20.000%
        Calculate Hours: 1 time(s)
        Calculate Taxes: 1 time(s)
    Data Entry: 30.000%
        Login: 1 time(s)
        Initialize Options: 1 time(s)
        Add New Record: 1 time(s)
        Modify Record: 1 time(s)
        Delete Record: 1 time(s)
    Sales: 50.000%
        Login: 1 time(s)
        Initialize Options: 1 time(s)
        Read Record: 1 time(s)
Scenarios
```

admin | Currently Selected Filter:

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.

Note: The suites in this chapter contain VU test scripts, which are generally used for performance testing. A suite, however, can also contain GUI scripts, VB scripts, or other user-defined test script types.

Creating a Suite from a Robot 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 maintains the test 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.

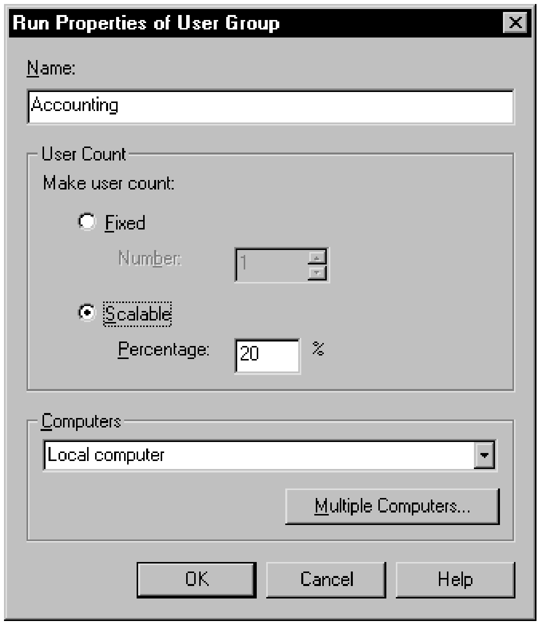
You can also create a suite based on an existing suite or from scratch. For more information about creating suites, see *Creating a Suite* on page 73.

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 226 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**.



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.

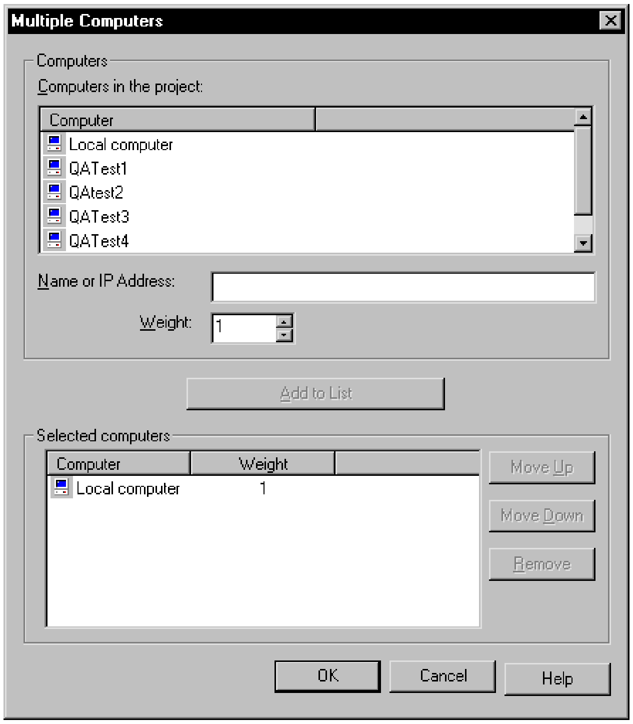
Note: Copy any custom-created external C libraries, Java class files, or COM components necessary for the test to the Agent computer.

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**.



Computer
Local computer
QATest1
QAtest2
QATest3
QATest4

Weight: 1

Computer	Weight
Local computer	1

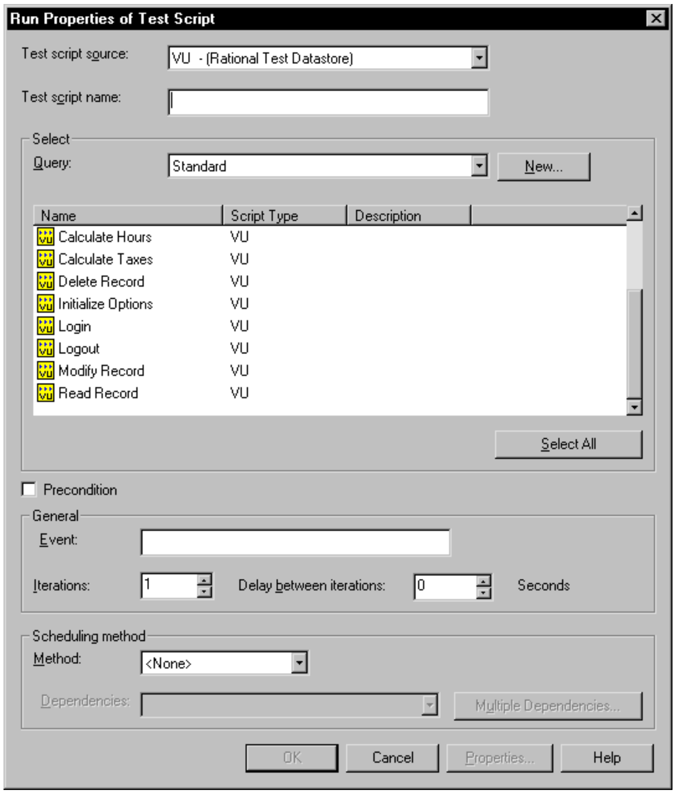
Inserting Test Scripts into a Suite

After you insert user groups into a suite, you add the test scripts that the user groups run. The suite on page 226 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**.


Run Properties of Test Script

Test script source: VU - (Rational Test Datastore)

Test script name:

Select

Query: Standard

Name	Script Type	Description
Calculate Hours	VU	
Calculate Taxes	VU	
Delete Record	VU	
Initialize Options	VU	
Login	VU	
Logout	VU	
Modify Record	VU	
Read Record	VU	

Precondition

General

Event:

Iterations: 1 Delay between iterations: 0 Seconds

Scheduling method

Method: <None>

Dependencies:

Setting a Precondition on a Test Script, Test Case, or Suite

When you insert a test script, test case, or suite into a suite, you can specify that successful completion of that item is a *precondition* for the remainder of that suite sequence. The item must pass for the remaining suite items at the same level to run.

For example, suppose a suite includes two suites, each of which contains an initialization test script and several test cases. If you set a precondition on the initialization test script and the test script fails, TestManager skips all remaining test cases within that suite *only*. The suite run resumes at the beginning of the second suite.

To set a precondition:

- Right-click the test script, suite, or test case to which to set the precondition, and select **Run Properties**.

Preconditions apply only to the specific instance of the test script, test case, or suite. For example, if you insert a test script multiple times, and you want to set a precondition on all instances of the test script, you must set the precondition for each test script.

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.

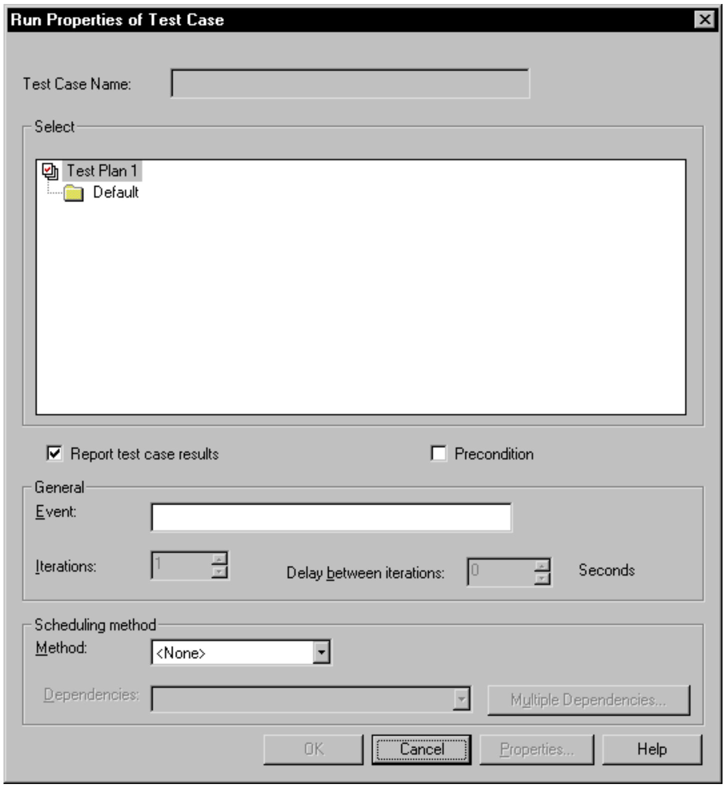
Inserting a Test Case into a Suite

You insert test cases into suites so that you can run multiple test cases at one time and save the set of test cases that are running together.

You insert configured test cases to verify that a test case succeeds in different environments. When you insert configured test cases in suites, TestManager automatically assigns the test cases to the appropriately configured computers.

To insert a test case into a suite:

- From an open suite, click **Suite > Insert > Test Case**.



You can set a precondition on a test case. When you set a precondition, the test case must successfully complete in order for other suite items with the same parent to run. For information about preconditions, see *Setting a Precondition on a Test Script, Test Case, or Suite* on page 231.

To set a precondition on a test case:

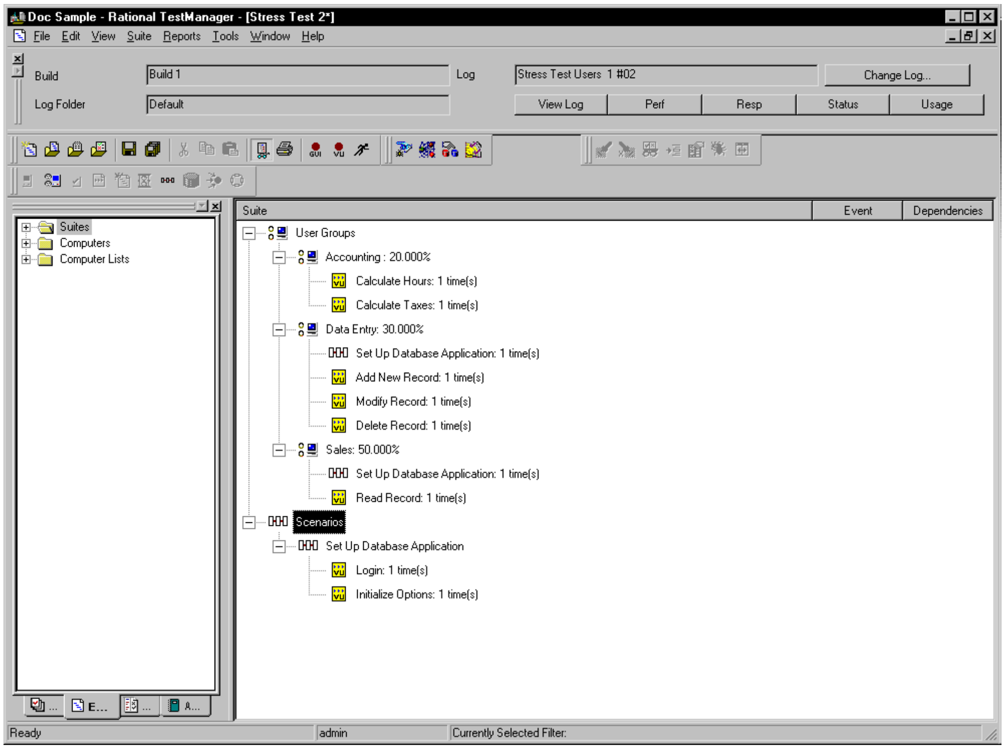
- Right-click the test case, and then select **Run Properties**.

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 226, 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:


Doc Sample - Rational TestManager - [Stress Test 2*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Suite
Event
Dependencies
Ready

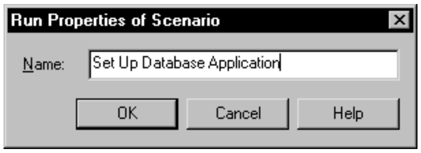
Build: Build 1
Log: Stress Test Users 1 #02
Log Folder: Default

- User Groups
  - Accounting : 20.000%
    - Calculate Hours: 1 time(s)
    - Calculate Taxes: 1 time(s)
  - Data Entry: 30.000%
    - Set Up Database Application: 1 time(s)
    - Add New Record: 1 time(s)
    - Modify Record: 1 time(s)
    - Delete Record: 1 time(s)
  - Sales: 50.000%
    - Set Up Database Application: 1 time(s)
    - Read Record: 1 time(s)
- Scenarios
  - Set Up Database Application
    - Login: 1 time(s)
    - Initialize Options: 1 time(s)

admin
Currently Selected Filter:

To create a new scenario:

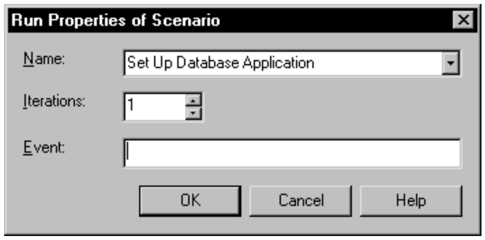
- From the Scenarios section of the suite, click **Suite > Insert > Scenario**.



Name: Set Up Database Application

To insert a scenario into a suite:

- Click where you want to place the scenario, and then click **Suite > Insert > Scenario**.



Name: Set Up Database Application
Iterations: 1
Event:

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.

Inserting a Suite into a Suite

Although you generally use suites that contain user (rather than computer) groups in performance testing, you can insert a suite that contains computer groups into another suite. The advantage of inserting a suite into a suite is that changes that you make to the child suite are persistent. So, for example, you can insert a group of test scripts into a computer-based suite, use the group in many different suites, but update the suite only once.

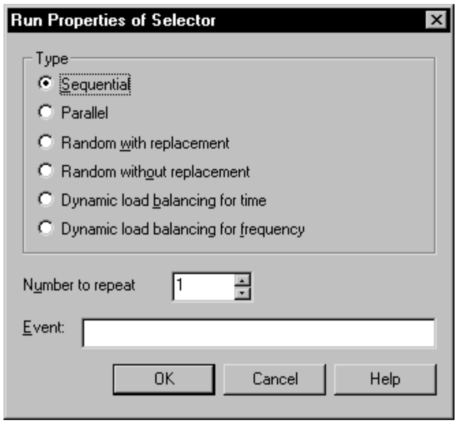
For more information, see *Inserting a Suite into a Suite* on page 171.

Inserting a Selector

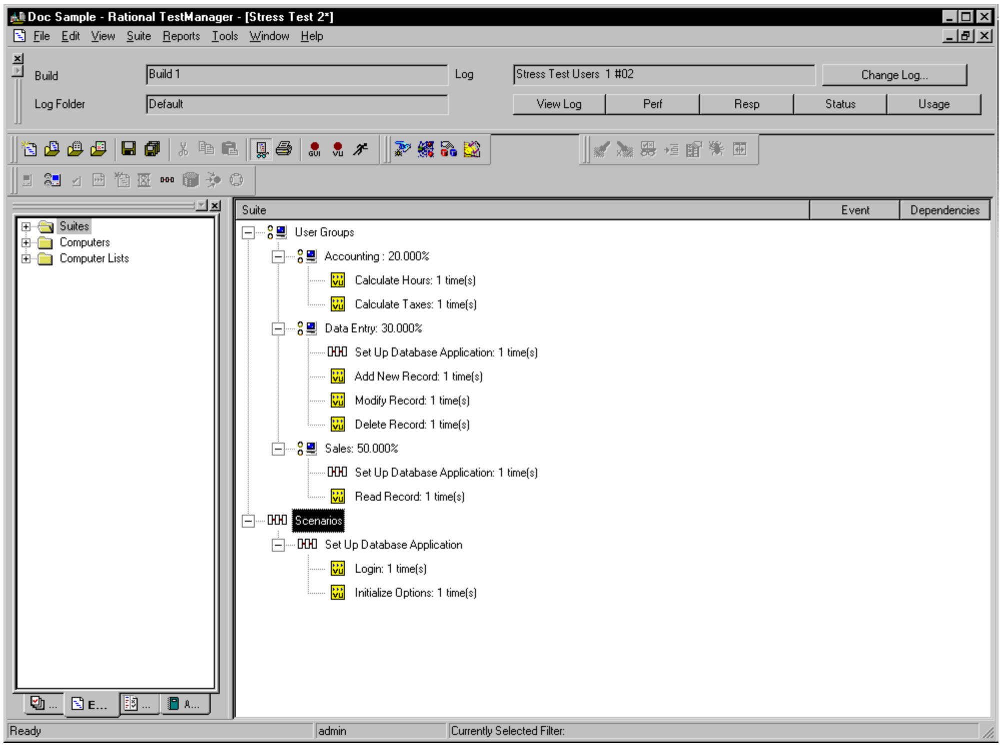
TestManager allows you to set suite items to run in different sequences by setting a *selector*. A *selector* provides more sophisticated control than running a simple sequence of consecutive items in a suite. A selector tells TestManager which items each virtual tester executes, and in what sequence. 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**.


Run Properties of Selector
Type
[✓ Sequential]
Parallel
Random with replacement
Random without replacement
Dynamic load balancing for time
Dynamic load balancing for frequency
Number to repeat 1
Event:
OK
Cancel
Help

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


Doc Sample - Rational TestManager - [Stress Test 2*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Suite
Event
Dependencies
Ready
admin
Currently Selected Filter:

Build: Build 1
Log Folder: Default
Log: Stress Test Users 1 #02

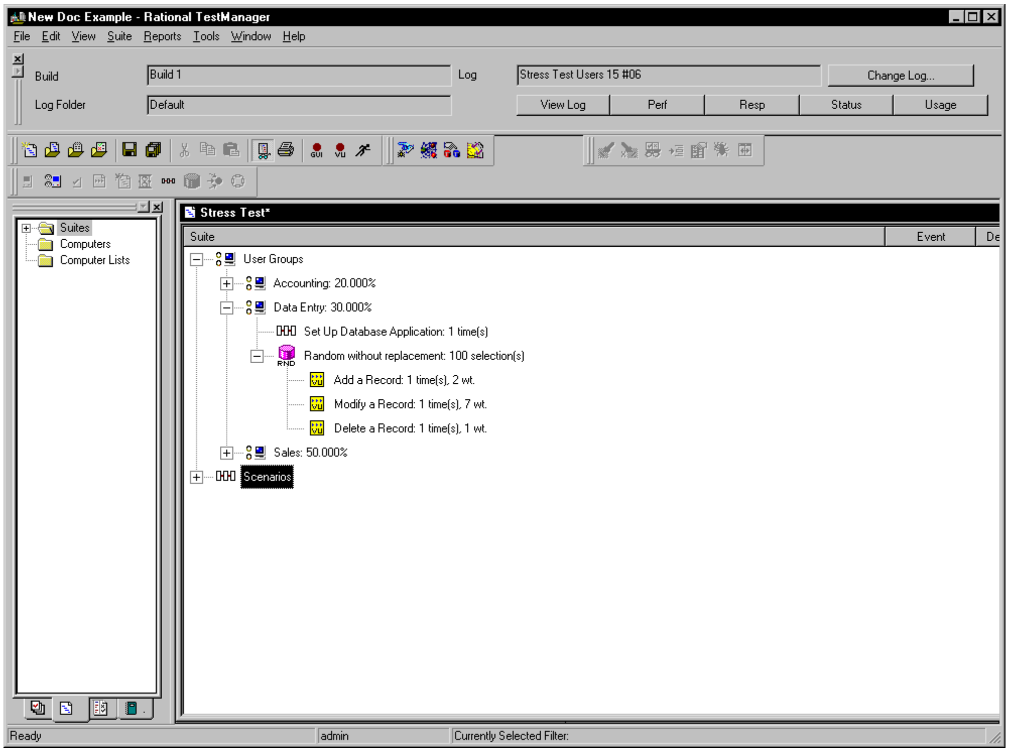
```
User Groups
    Accounting : 20.000%
        Calculate Hours: 1 time(s)
        Calculate Taxes: 1 time(s)
    Data Entry: 30.000%
        Set Up Database Application: 1 time(s)
        Add New Record: 1 time(s)
        Modify Record: 1 time(s)
        Delete Record: 1 time(s)
    Sales: 50.000%
        Set Up Database Application: 1 time(s)
        Read Record: 1 time(s)
Scenarios
    Set Up Database Application
        Login: 1 time(s)
        Initialize Options: 1 time(s)
```

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.



Build: Build 1
Log Folder: Default
Log: Stress Test Users 15 #06

Stress Test*

```
User Groups
  Accounting: 20.000%
  Data Entry: 30.000%
    Set Up Database Application: 1 time(s)
    Random without replacement: 100 selection(s)
      Add a Record: 1 time(s), 2 wt.
      Modify a Record: 1 time(s), 7 wt.
      Delete a Record: 1 time(s), 1 wt.
  Sales: 50.000%
Scenarios
```

admin | Currently Selected Filter:

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.

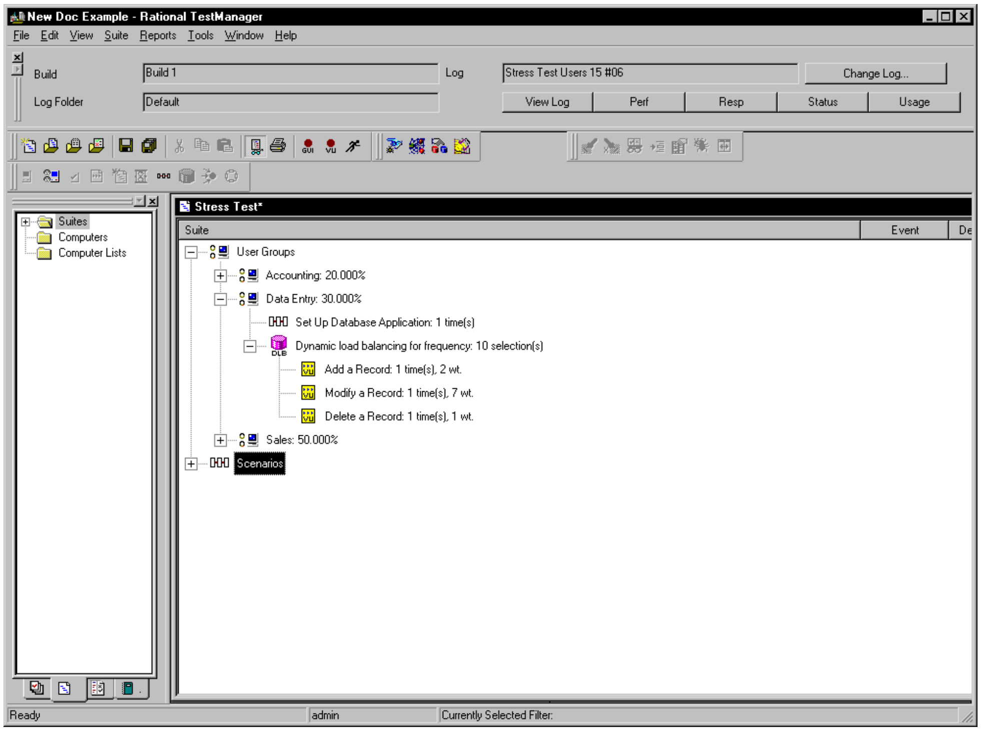
- **Dynamic load balancing** – With dynamic load balancing, items are not selected randomly. Think again of the bucket that contains red and green balls. You have assigned an equal "weight" to each ball. If the first ball that is selected is red, the second ball selected is always green. This is because 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 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 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.



Build: Build 1
Log: Stress Test Users 15 #06
Log Folder: Default

Stress Test*

User Groups
Accounting: 20.000%
Data Entry: 30.000%
Set Up Database Application: 1 time(s)
Dynamic load balancing for frequency: 10 selection(s)
Add a Record: 1 time(s), 2 wt.
Modify a Record: 1 time(s), 7 wt.
Delete a Record: 1 time(s), 1 wt.
Sales: 50.000%
Scenarios

admin
Currently Selected Filter:

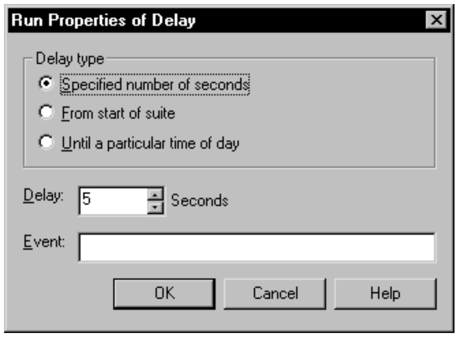
When you run the suite, the first Data Entry virtual tester selects the Modify Record script, because it has the largest weight. But because the workload is balanced across *all* Data Entry virtual testers, after the first virtual tester exits, TestManager recalculates the weights to reflect the fact 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**.

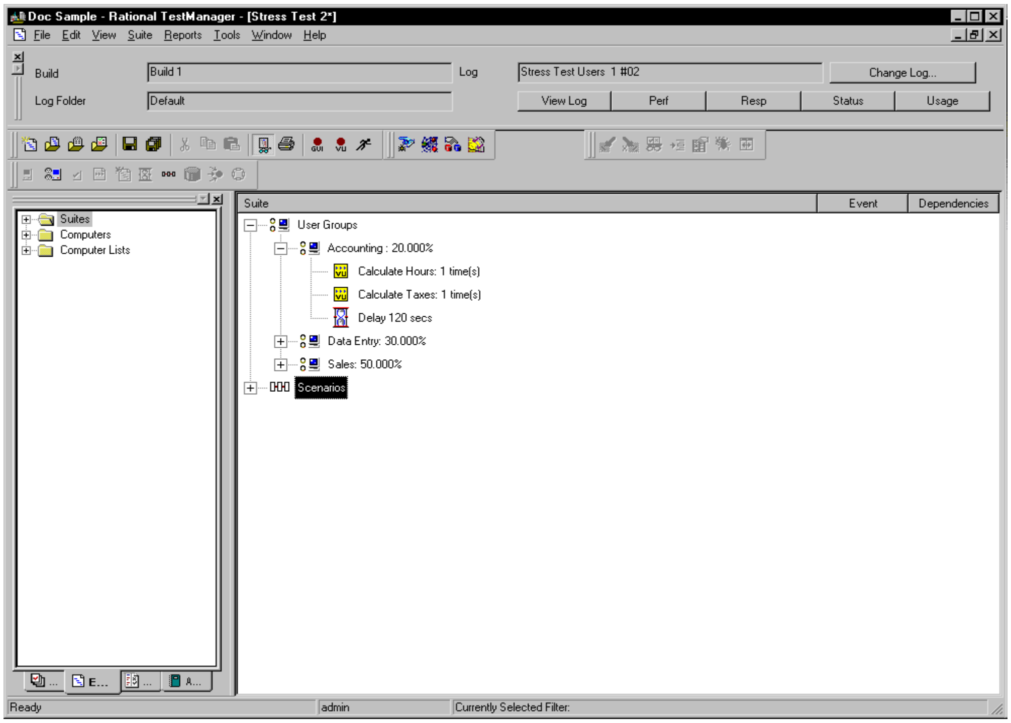

Run Properties of Delay

Delay type
(●) Specified number of seconds
( ) From start of suite
( ) Until a particular time of day

Delay: 5 Seconds

Event:

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.



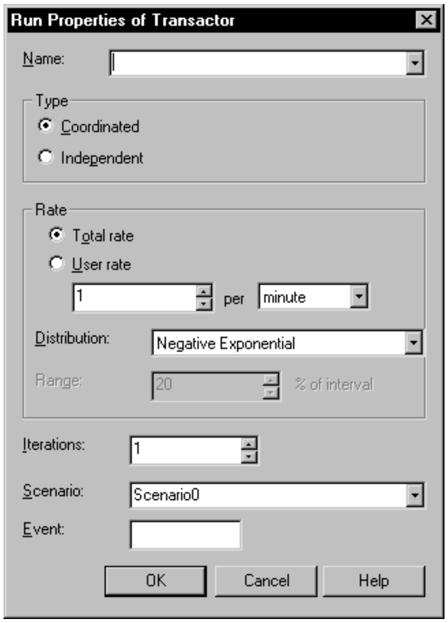
You can insert a delay into a suite or a test script. The advantages of inserting a delay into a suite are that the delay is visible in the suite and the delay is easy to change without editing the test script.

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.**


Run Properties of Transactor

Name:

Type
(●) Coordinated
( ) Independent

Rate
(●) Total rate
( ) User rate
1 per minute

Distribution: Negative Exponential

Range: 20 % of interval

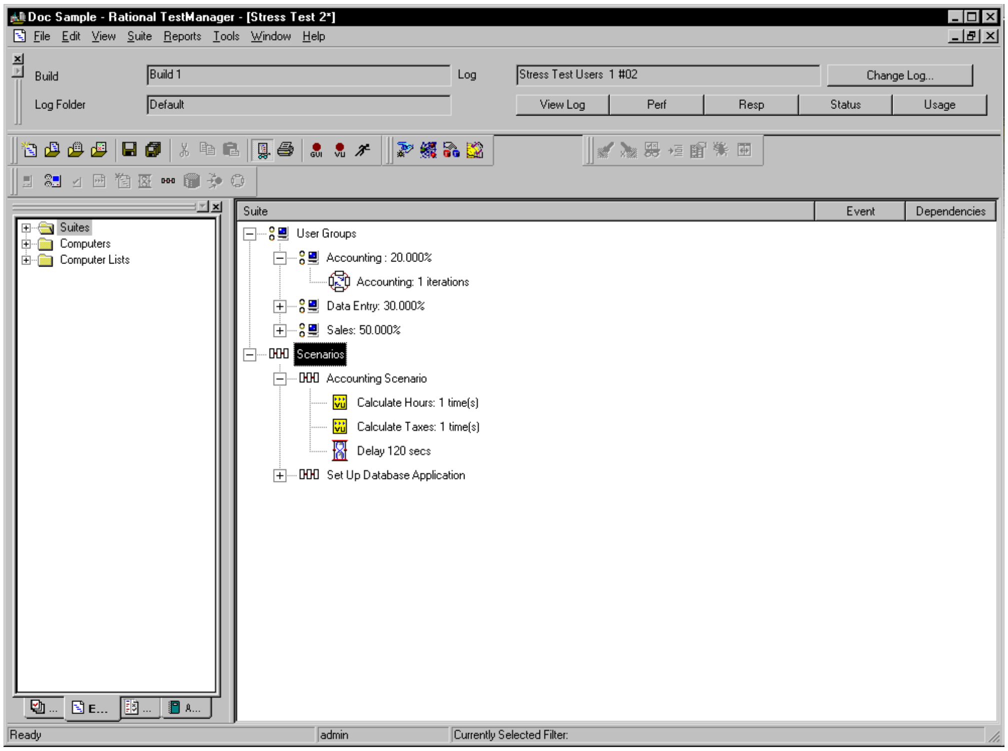
Iterations: 1

Scenario: Scenario0

Event:

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 242.

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:


Doc Sample - Rational TestManager - [Stress Test 2*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Suite
Event
Dependencies

Build: Build 1
Log Folder: Default
Log: Stress Test Users 1 #02

```
User Groups
  Accounting : 20.000%
    Accounting: 1 iterations
  Data Entry: 30.000%
  Sales: 50.000%
Scenarios
  Accounting Scenario
    Calculate Hours: 1 time(s)
    Calculate Taxes: 1 time(s)
    Delay 120 secs
  Set Up Database Application
```

Ready | admin | Currently Selected Filter:

This suite is identical to the one on page 242, 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 user 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 user 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/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 238.

A transactor can set an event. For information about events, see *Using Events and Dependencies to Coordinate Execution* on page 254.

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.

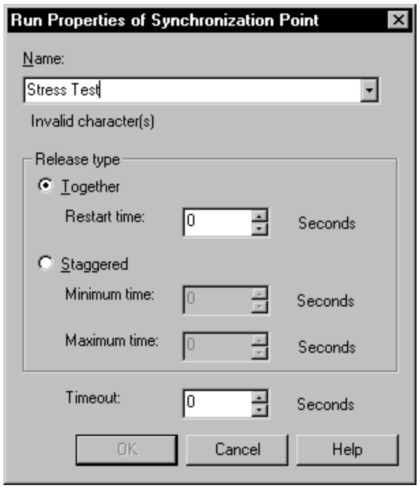
- Into a suite – Insert a synchronization point into a suite through TestManager.

  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 be released.
  - You can specify a timeout period.

To insert a synchronization point into a suite:

- Click **Suite > Insert > Synchronization Point**.



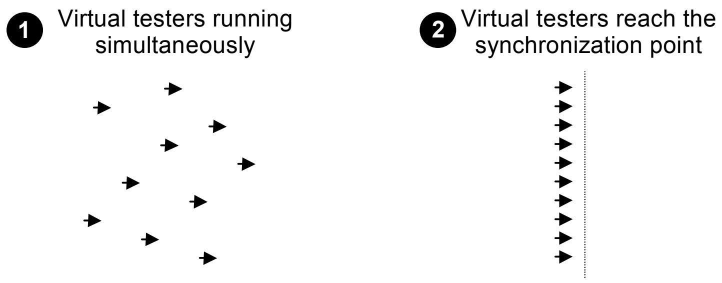
For example, when you run a stress test (an attempt to run your applications under extreme conditions to see if they or the server "break"), your suite might contain virtual testers that perform the certain operations continuously and repeatedly for hours on end. To most effectively run a stress test, 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

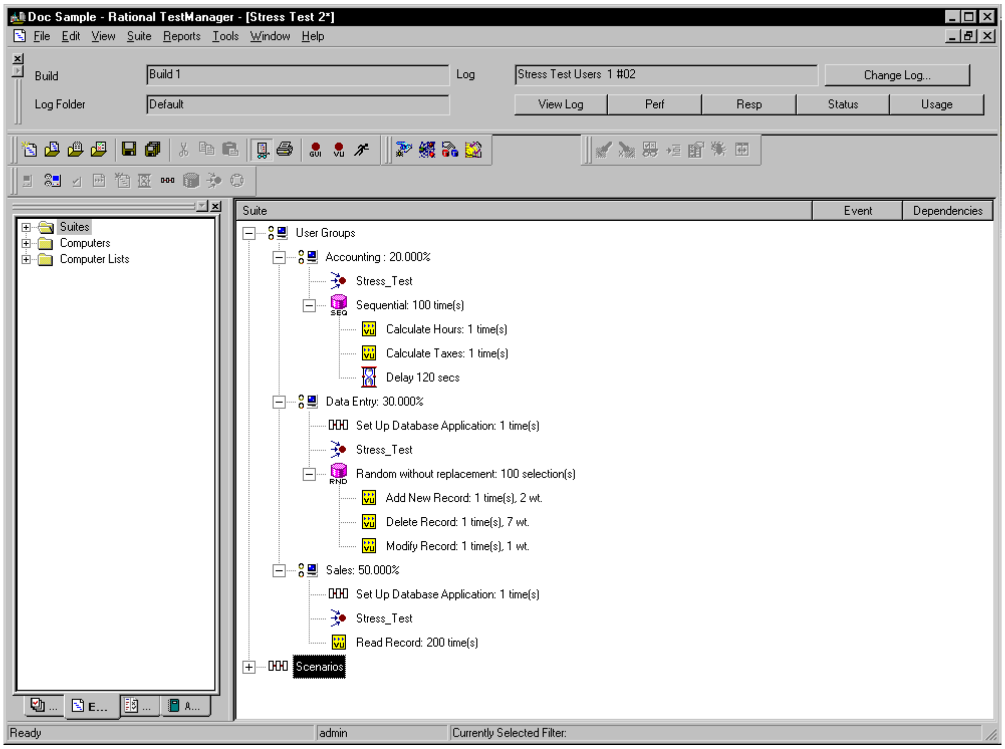
At the start of a test, all virtual testers begin executing their assigned test scripts. They continue to run until they reach the synchronization point. When specified in a test script, a synchronization point is a programmatic command (`sync_point` in a VU test script, `SQASyncPointWait` in a SQABasic 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.


1 Virtual testers running simultaneously
2 Virtual testers reach the synchronization point

The following suite shows synchronization points called Stress Test:


Doc Sample - Rational TestManager - [Stress Test 2*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Suite
Event
Dependencies
Ready

Build: Build 1
Log: Stress Test Users 1 #02
Log Folder: Default

```
User Groups
  Accounting : 20.000%
    Stress_Test
    Sequential: 100 time(s)
      Calculate Hours: 1 time(s)
      Calculate Taxes: 1 time(s)
      Delay 120 secs
  Data Entry: 30.000%
    Set Up Database Application: 1 time(s)
    Stress_Test
    Random without replacement: 100 selection(s)
      Add New Record: 1 time(s), 2 wt.
      Delete Record: 1 time(s), 7 wt.
      Modify Record: 1 time(s), 1 wt.
  Sales: 50.000%
    Set Up Database Application: 1 time(s)
    Stress_Test
    Read Record: 200 time(s)
Scenarios
```

admin | Currently Selected Filter:

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 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 that provides the load, 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.

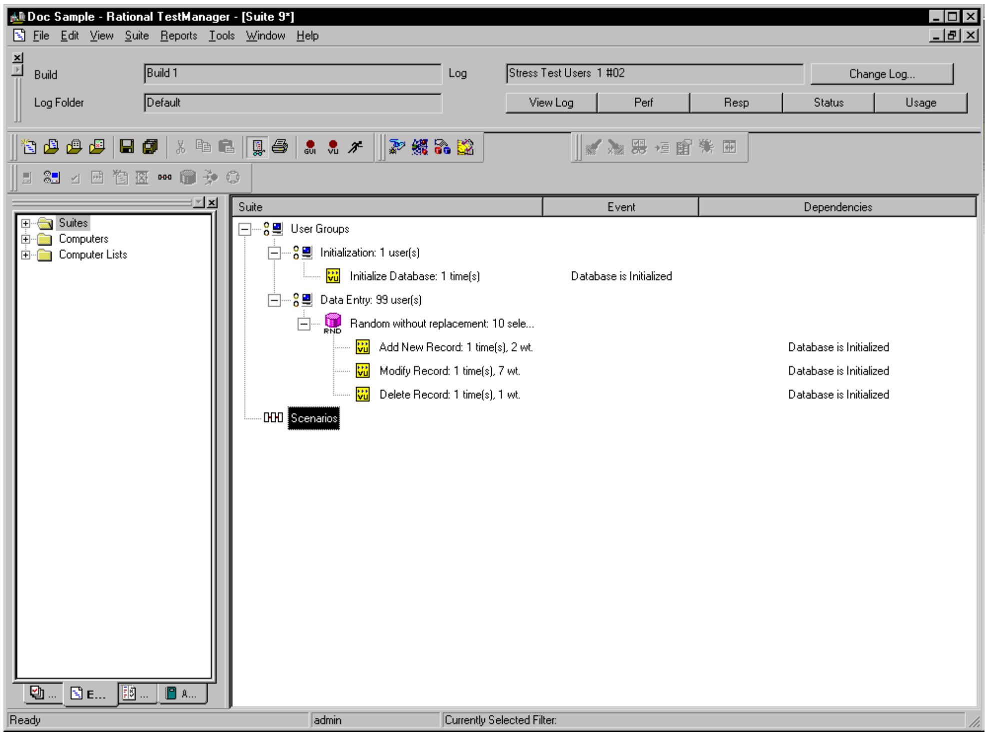
Using Events and Dependencies to Coordinate Execution

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 *Setting a Precondition on a Test Script, Test Case, or Suite* on page 231.

The following suite shows 99 virtual testers waiting until the first virtual tester initializes a database:


Doc Sample - Rational TestManager - [Suite 9*]
File Edit View Suite Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Scenarios

Build: Build 1
Log Folder: Default
Log: Stress Test Users 1 #02

Suite	Event	Dependencies
User Groups		
Initialization: 1 user(s)		
Initialize Database: 1 time(s)	Database is Initialized	
Data Entry: 99 user(s)		
Random without replacement: 10 sele...		
Add New Record: 1 time(s), 2 wt.		Database is Initialized
Modify Record: 1 time(s), 7 wt.		Database is Initialized
Delete Record: 1 time(s), 1 wt.		Database is Initialized
Scenarios		

Ready | admin | Currently Selected Filter:

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 *Executing Tests* on page 87.

Working with Datapools 11

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 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 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. The following table shows which activities you can perform with each product:

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 about using datapools in Rational Robot, see the *Using Rational Robot* manual.

Managing Datapool Files

A datapool consists of two files: a text file and a specification file.

- 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 278 and *Creating a Datapool Outside Rational Test* on page 284.

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 that you provided during recording or hand-coded into the test script—you might encounter problems at suite runtime.

The following are some examples of problems that datapools can solve:

- *Problem:* During recording, you create a personnel file for a new employee using the employee's unique social security number. Each time the 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 of stages shown is the typical order for planning and creating a datapool for test scripts.

Plan the Datapool

- What datapool columns do you need?
- What data type should you assign each column?
- Do you need to create data types?

Generate the Code

VU Test Scripts

- Select the **Use datapools** recording option.
- Record the transaction(s), and then stop recording.
- Robot automatically generates datapool commands.
- Robot automatically matches up test script variable names with datapool column names.

GUI Test Scripts

- Manually add datapool commands to the test script.
- Match up test script variable names with datapool columns.

Create and Populate the Datapool

VU Test Scripts

- In Robot, click **Edit > Datapool Information.**
- Modify DATAPOOL_CONFIG or accept the defaults.

VU and GUI Test Scripts

- In Robot or TestManager, define datapool columns (including assigning a data type to each datapool column).
- Generate the data.

The following steps provide details about these activities.

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 that 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 266.

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 that **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 264.

- Generating the data. Once you configure the datapool and define its columns, you populate the datapool by clicking **Generate Data**.

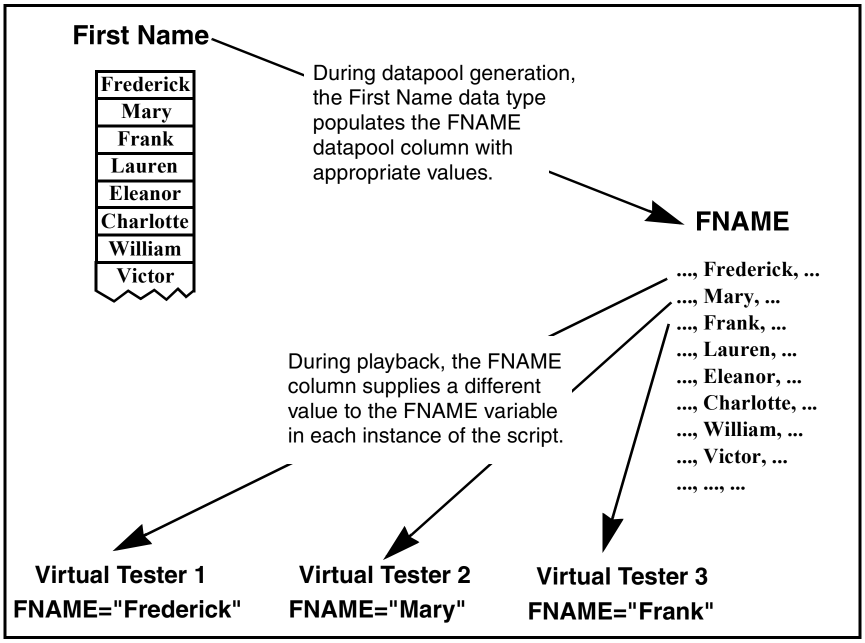
Note: You can also create and populate a datapool file manually and import it into the datastore. For more information, see *Creating a Datapool Outside Rational Test* on page 284.

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 following figure shows the relationship between data types, datapool columns, and the values assigned to test script variables during playback:



Standard and User-Defined Data Types

There are two kinds of datapool data types, as follows:

- **Standard data types** that are included with Rational Test. These data types include commonly used, realistic sets of data in categories such as first and last names, company names, cities, and numbers.

  For a list of the standard data types, see *Standard Datapool Data Types* on page 341.

- **User-defined data types** that you create. You must create a data type if none of the standard data types contain the kind of values you want to supply to a test script variable.

  User-defined data types are useful in situations such as the following:

  - When a field accepts a limited number of valid values. For example, suppose a datapool column supplies data to a 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 268.

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"
  ```

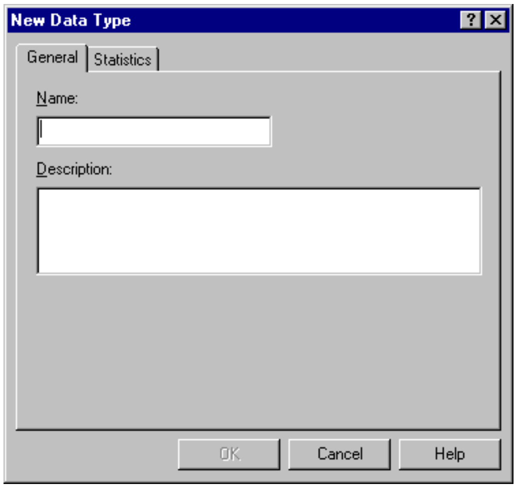
  For more information about the `DATAPOOL_CONFIG` section of a test script, see the *Using Rational Robot* manual.

Creating User-Defined Data Types

If none of the standard data types can provide the correct kind of values to a test script variable, you can 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 280.

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 that the **No. of records to generate** value does not exceed the number of unique values in your user-defined data type.

For information about the values you set in the Datapool Specification dialog box, see *Defining Datapool Columns* on page 271.

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 the following ways:

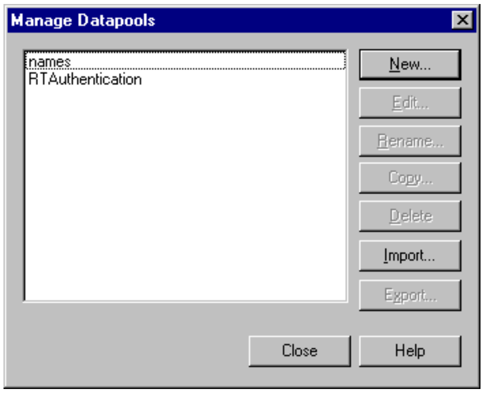
- Through a user-defined data type. For information, see *Creating User-Defined Data Types* on page 266.

  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. The IME is included in the Japanese version of Microsoft Windows.

- Through the Read From File data type. For information, see *Creating a Column of Values Outside Rational Test* on page 288.

Managing Datapools

You manage datapools from 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 278.



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.

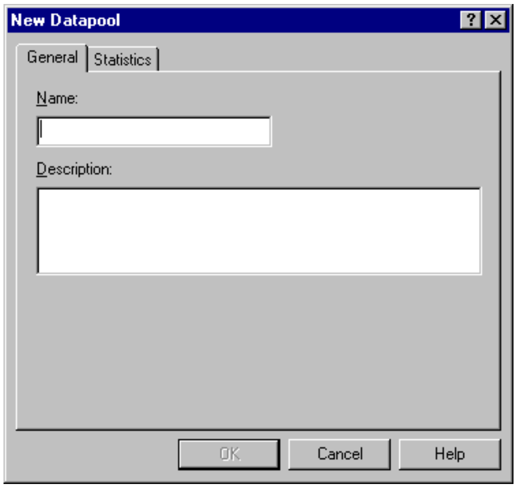
- Data type and properties for each datapool column.

  For information about the properties you can define for a datapool column, see *Defining Datapool Columns* on page 271.

- Number of records to generate.

To create and automatically populate a datapool:

- Click **Tools > Manage > Datapools**, and then click **New**.



If There Are Errors

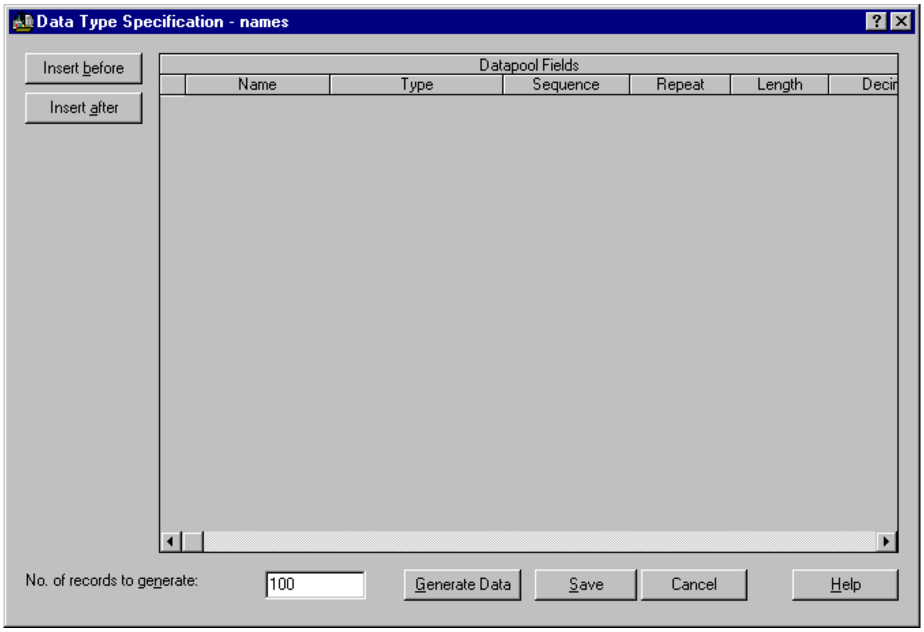
If the datapool values are not successfully generated, TestManager asks if you want to see an error report rather than a summary of the generated data. Viewing this report can help you identify where to make corrections in the datapool fields. To view an error report, click **Yes** when TestManager asks 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
Name	The name of a datapool column (and its corresponding test script variable). If you change the name of a datapool column, be sure that the new name matches all instances of its corresponding test script variable. If you create a datapool outside of the Rational Test environment and then import it, TestManager automatically assigns default names to the datapool columns. Use **Name** to match the imported datapool column names with their corresponding test script variables. Names are case-sensitive. You can use an IME to type multi-byte characters in datapool field names.
Type	The standard or user-defined data type that supplies values to the datapool column in **Name**. User-defined data types are marked with an asterisk (*). Specify the data type to assign to the datapool column, as follows: ▪ 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: ▪ See Appendix B for a description of the standard data types. ▪ 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: `*MyData` ▪ To create a new user-defined data type, enter the data type name (without the asterisk) in the field, and then press RETURN. After you click **Yes** to confirm that you want to create a user-defined data type, the Data Type Properties - Edit dialog box appears. ▪ For information about creating a data type, see *Creating User-Defined Data Types* on page 266.

Grid column	Description
Sequence	The order in which the values in the data type specified in **Type** are written to the datapool column. Select one of these options from the drop-down list: ▪ **Random** – Writes numeric and alphanumeric values to the datapool column in any order. ▪ **Sequential** – Writes numeric values sequentially (for example, 0, 1, 2...). With decimal numbers, the sequence is based on the lowest possible decimal increment (for example, with a **Decimals** value of 2, the sequential values are 0.00, 0.01, 0.02, ...). ▪ **Sequential** is only supported for numeric values (including date and time values) and values generated from user-defined data types. When you select Sequential with numeric data types, and you specify a **Minimum** and **Maximum** range, **Interval** must be greater than 0. ▪ **Unique** – With data type Integers - Signed, ensures that numbers written to the datapool column are unique. Also, set **Repeat** to 1, and define a **Minimum** and **Maximum** range. Do not confuse the **Random** and **Sequential** settings in this grid with **Random** and **Sequential** access order in the Configure Datapool in Script dialog box.The **Random** and **Sequential** settings in this grid determine the order in which values are written to an individual datapool column at datapool creation time. **Random** and **Sequential** access order determine the order in which virtual testers access datapool rows at suite runtime.
Repeat	The number of times a given value can appear in a datapool column. **Repeat** cannot be set to 0. To make values unique with Integers - Signed data types and user-defined data types, set **Repeat** to 1. For unique Integers - Signed values, also set **Sequence** to either Sequential or Unique. When defining unique values, make sure that the number of rows you are generating is not higher than the range of possible unique values.
Length	The maximum number of characters that a value in the datapool column can have. If the datapool column contains numeric values, **Length** specifies the maximum number of characters a number can have, *including* a decimal point and minus sign, if any. For example, for decimal numbers as high as 999.99, set **Length** to 6. For decimal numbers as low as -999.99, set **Length** to 7. **Length** cannot be 0.
Decimals	Specifies the maximum number of decimal places that floating point values can have. Maximum setting is 6 decimal places.

Grid column	Description
Interval	Writes a sequence of numeric values to the datapool column. The sequence increments by the interval you set. For example, if **Interval** is 10, the datapool column contains 0, 10, 20, and so on. If **Interval** is 10 and **Decimal** is 2, the datapool column contains 0.00, 0.10, 0.20, and so on. Minimum interval is 1. Maximum interval is 999999. With numeric data types (including dates and times), when **Sequence** is set to Sequential and you specify a **Minimum** and **Maximum** range, **Interval** must be greater than 0. Use **Interval** only with numeric values (including dates and times).
Minimum	Specifies the lowest in a range of numeric values. For example, if the datapool column supplies order number values, and the lowest possible order number is 10000, set **Type** to Integer - Signed, **Minimum** to 10000, and **Maximum** to the highest possible order number. Use **Minimum** only with numeric values (including dates and times).
Maximum	Specifies the highest in a range of numeric values. For example, if the datapool column supplies values to a variable named *ounces*, set **Type** to Integer - Signed, **Minimum** to 0, and **Maximum** to 16. Use **Maximum** only with numeric values (including dates and times).
Seed	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.
Data File	The path to the user-defined data type file. The path is automatically inserted for you. This field is not modifiable. Data type files are stored in the Datatype directory of your project. You never have to modify these files directly.

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.

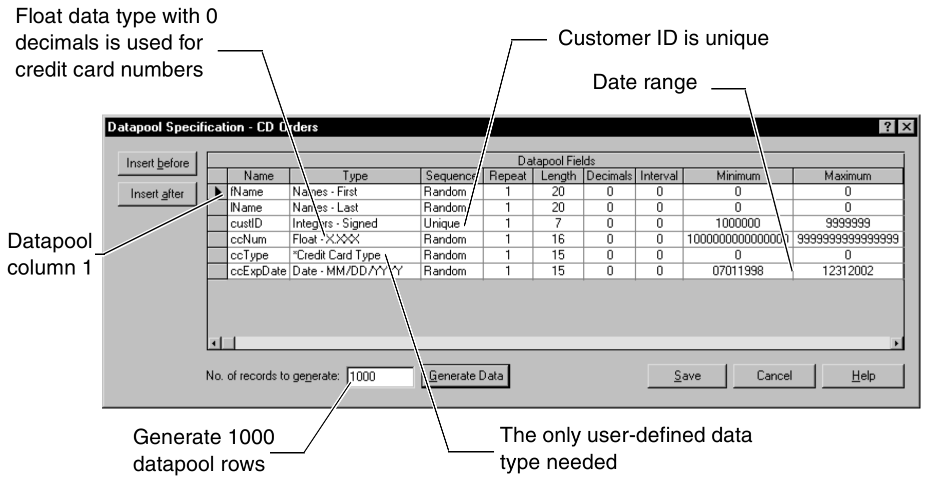
Note: You can use an IME to type multi-byte characters into the **Name** column only. The IME is automatically disabled when you are editing any other column.

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 in the following figure:


Float data type with 0 decimals is used for credit card numbers
Customer ID is unique
Date range
Datapool Specification - CD Orders
Insert before
Insert after
Datapool column 1
No. of records to generate: 1000
Generate Data
Save
Cancel
Help
Generate 1000 datapool rows
The only user-defined data type needed

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	Integers - Signed	Unique	1	7	0	0	1000000	9999999
ccNum	Float - X.XXX	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

The following are some highlights of the datapool column you have chosen:

- **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.

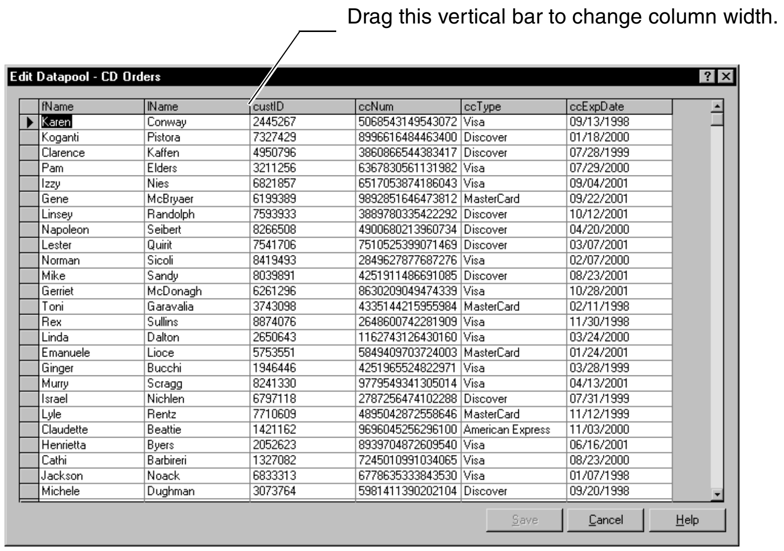
Example of Datapool Value Generation

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.
Edit Datapool - CD Orders
Save
Cancel
Help

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	3860866544383417	Discover	07/28/1999
Pam	Elders	3211256	6367830561131982	Visa	07/29/2000
Izzy	Nies	6821857	6517053874186043	Visa	09/04/2001
Gene	McBryaer	6199389	9892851646473812	MasterCard	09/22/2001
Linsey	Randolph	7593933	3889780335422292	Discover	10/12/2001
Napoleon	Seibert	8266508	4900680213960734	Discover	04/20/2000
Lester	Quirit	7541706	7510525399071469	Discover	03/07/2001
Norman	Sicoli	8419493	2849627877687276	Visa	02/07/2000
Mike	Sandy	8039891	4251911486691085	Discover	08/23/2001
Gerriet	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	Rentz	7710609	4895042872558646	MasterCard	11/12/1999
Claudette	Beattie	1421162	9696045256296100	American Express	11/03/2000
Henrietta	Byers	2052623	8939704872609540	Visa	06/16/2001
Cathi	Barbireri	1327082	7245010991034065	Visa	08/23/2000
Jackson	Noack	6833313	6778635333843530	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.

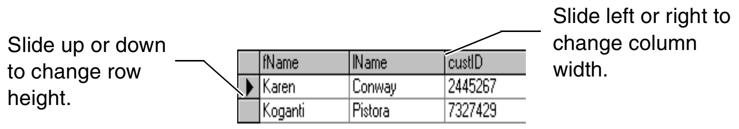
Editing Datapool Values

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 that 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.


Slide up or down to change row height.
Slide left or right to change column width.

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 275.

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

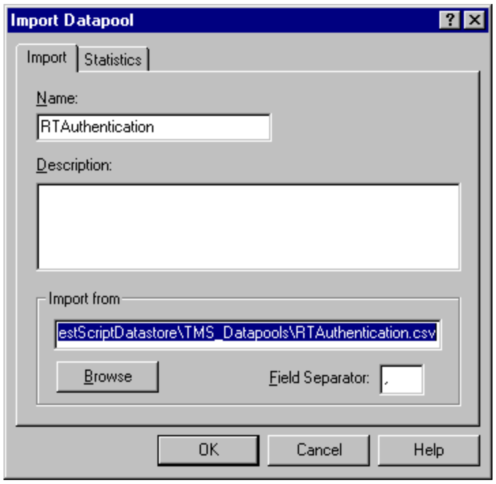
It is possible for you to create and populate a datapool yourself, using a tool such as Microsoft Excel. For example, you might want to export data from your database into a .csv file, and then use that file as your datapool.

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 287.

To import a datapool .csv file:

- Click **Tools > Manage > Datapools**, and then click **Import**.


Name: RTAuthentication
Description:
Import from: estScriptDatastore\TMS_Datapools\RTAuthentication.csv
Field Separator: ,

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 that 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 281.

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 278.

To export a datapool to a location on your computer:

- 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 264.

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 that contains 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**

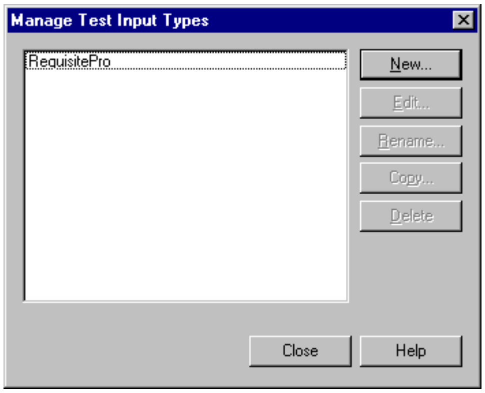
Note: You can also assign the standard data type Read From File to a user-defined data type. For information about using the Read From File data type, see *Creating a Column of Values Outside Rational Test* on page 288.

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.	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. With the Integers - Signed data type, take all of these actions: ▪ Set **Sequence** to Unique or Sequential. ▪ Set **Repeat** to 1. ▪ If **Sequence**=Unique, set an appropriate range in **Minimum** and **Maximum**. ▪ Make sure that the values of **Length** and **No. of records to generate** are appropriate for the set of numbers to generate. With the Read From File data type, see *Generating Unique Values* on page 289 for information. With user-defined data types, see *Generating Unique Values from User-Defined Data Types* on page 267 for information.
Generate enough datapool rows.	Generate at least as many unique datapool rows as the number of times the datapool will be accessed during a test. 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. You specify the number of rows to generate in the **No. of records to generate** field of the Datapool Specification dialog box.
Disable cursor wrapping.	If the datapool cursor wraps after the last row in the datapool has been accessed, previously fetched rows are fetched again. Disable cursor wrapping in any of these ways: ▪ When editing the `DATAPOOL_CONFIG` section of a VU test script in the Configure Datapool in Script dialog box, set **Wrap at end of file?** to **No**. ▪ When editing a VU test script in Robot, add `DP_NOWRAP` to the list of flags in the *`flags`* argument of the `DATAPOOL_CONFIG` statement or the `datapool_open` function. ▪ When editing a GUI test script in Robot, set the *`wrap`* argument of the `SQADatapoolOpen` command to **False**.

What to do	How to do it
Use sequential or shuffle access order.	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. With random access, rows can be referenced in the access order multiple times. Therefore, a given row can be retrieved multiple times. You can set row access order in any of these ways: ▪ When editing the `DATAPOOL_CONFIG` section of a VU test script in the Configure Datapool in Script dialog box, set **Access Order** to **Sequential** or **Shuffle**. ▪ When editing a VU test script in Robot, add `DP_SEQUENTIAL` or `DP_SHUFFLE` to the list of flags in the *`flags`* argument of the `DATAPOOL_CONFIG` statement or the `datapool_open` function. ▪ When editing a GUI test script in Robot, set the *`sequence`* argument of the `SQADatapoolOpen` command to `SQA_DP_SEQUENTIAL` or `SQA_DP_SHUFFLE`.
Do not rewind the cursor during a test.	If you rewind the datapool cursor during a test (through the VU `datapool_rewind` function or the SQABasic `SQADatapoolRewind` command), previously accessed rows will be fetched again.

Note: Rational Test can guarantee that a datapool contains unique rows only when you generate datapool data through 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 278.

Datapool Structure

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

- Each row contains one record.
- Each record contains datapool field values delimited by a field separator. Any character can be used for the field separator. Some common field separators are as follows:
  - Comma (,). This is typically the default in the U.S. and the U.K.
  - 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

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

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

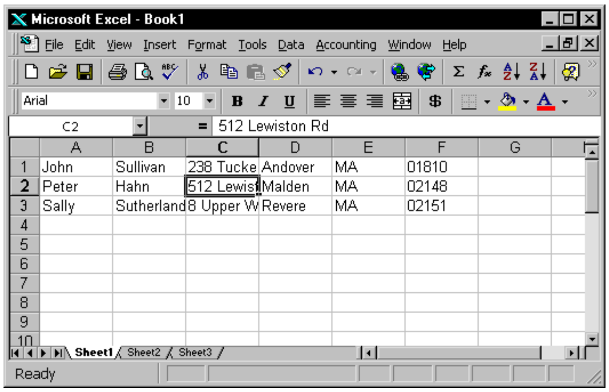
Using Microsoft Excel to Create Datapool Data

When you are using Microsoft Excel to populate a datapool, do not separate values with the Windows separator character on page 285. 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:


Microsoft Excel - Book1
File Edit View Insert Format Tools Data Accounting Window Help
Arial
10
Sheet1
Sheet2
Sheet3
Ready

C2 = 512 Lewiston Rd

	A	B	C	D	E	F	G
1	John	Sullivan	238 Tucke	Andover	MA	01810	
2	Peter	Hahn	512 Lewis	Malden	MA	02148	
3	Sally	Sutherland	8 Upper W	Revere	MA	02151	
4							
5							
6							
7							
8							
9							
10							

Note that:

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

Saving the Datapool in Excel

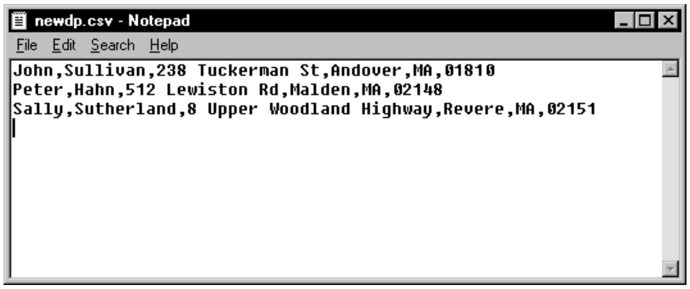
When you finish adding rows of values to the datapool, save the datapool to .csv format.

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:



```
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
```

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 when importing the datapool. 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 276.

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 that 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 271.

Reporting Performance Testing Results 12

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
- Exporting reports
- Changing report defaults
- Types of reports

About Reports

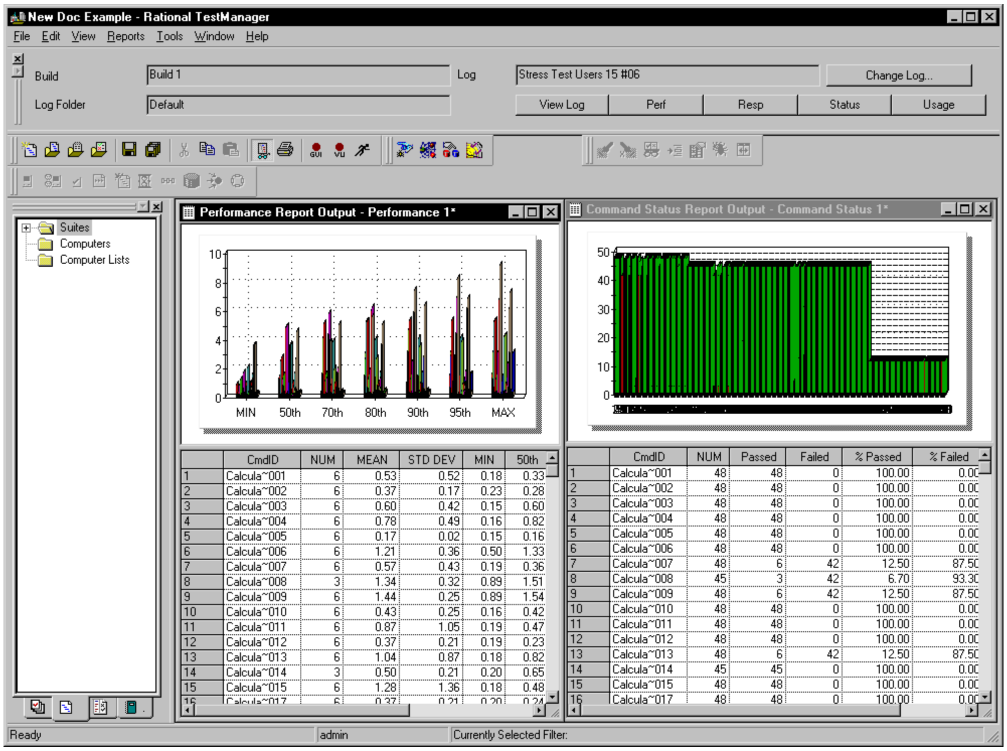
TestManager provides several types of reports that help you analyze the success or failure of a given suite run, as well as the performance of the server under specified conditions. For example, you can determine how long it took for a virtual tester to execute a command and how response times varied 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:


New Doc Example - Rational TestManager
File Edit View Reports Tools Window Help
Change Log...
View Log
Perf
Resp
Status
Usage
Suites
Computers
Computer Lists
Performance Report Output - Performance 1*
MIN
50th
70th
80th
90th
95th
MAX
Command Status Report Output - Command Status 1*
Ready
admin
Currently Selected Filter:

Build: Build 1
Log Folder: Default
Log: Stress Test Users 15 #06

	CmdID	NUM	MEAN	STD DEV	MIN	50th
1	Calcula~001	6	0.53	0.52	0.18	0.33
2	Calcula~002	6	0.37	0.17	0.23	0.28
3	Calcula~003	6	0.60	0.42	0.15	0.60
4	Calcula~004	6	0.78	0.49	0.16	0.82
5	Calcula~005	6	0.17	0.02	0.15	0.16
6	Calcula~006	6	1.21	0.36	0.50	1.33
7	Calcula~007	6	0.57	0.43	0.19	0.36
8	Calcula~008	3	1.34	0.32	0.89	1.51
9	Calcula~009	6	1.44	0.25	0.89	1.54
10	Calcula~010	6	0.43	0.25	0.16	0.42
11	Calcula~011	6	0.87	1.05	0.19	0.47
12	Calcula~012	6	0.37	0.21	0.19	0.23
13	Calcula~013	6	1.04	0.87	0.18	0.82
14	Calcula~014	3	0.50	0.21	0.20	0.65
15	Calcula~015	6	1.28	1.36	0.18	0.48
16	Calcula~017	6	0.37	0.21	0.20	0.24

	CmdID	NUM	Passed	Failed	% Passed	% Failed
1	Calcula~001	48	48	0	100.00	0.0
2	Calcula~002	48	48	0	100.00	0.0
3	Calcula~003	48	48	0	100.00	0.0
4	Calcula~004	48	48	0	100.00	0.0
5	Calcula~005	48	48	0	100.00	0.0
6	Calcula~006	48	48	0	100.00	0.0
7	Calcula~007	48	6	42	12.50	87.5
8	Calcula~008	45	3	42	6.70	93.3
9	Calcula~009	48	6	42	12.50	87.5
10	Calcula~010	48	48	0	100.00	0.0
11	Calcula~011	48	48	0	100.00	0.0
12	Calcula~012	48	48	0	100.00	0.0
13	Calcula~013	48	6	42	12.50	87.5
14	Calcula~014	45	45	0	100.00	0.0
15	Calcula~015	48	48	0	100.00	0.0
16	Calcula~017	48	48	0	100.00	0.0

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	Display the response times, and calculate the mean, standard deviation, and percentiles for each command in the suite run. 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.	*Performance Reports* on page 308
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.	*Compare Performance Reports* on page 312
Response vs. Time	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. 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. You can right-click on the report, select a computer that was in the run, and graph the resource monitoring statics for that computer. These are the same statistics that you display when you monitor resources during a suite run.	*Response vs. Time Reports* on page 317
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.	*Command Status Reports* on page 320
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.	*Command Usage Reports* on page 322

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 about viewing and using test logs, see *Evaluating Tests* on page 127.

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.

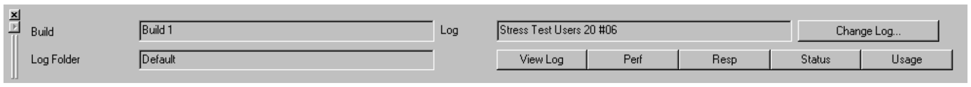
Note: You might also want to view the log files—the "raw" result files—before you run reports against them. For information about viewing log information in the Test Log window, see *Evaluating Tests* on page 127.

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 307.

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 292 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 304. 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 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

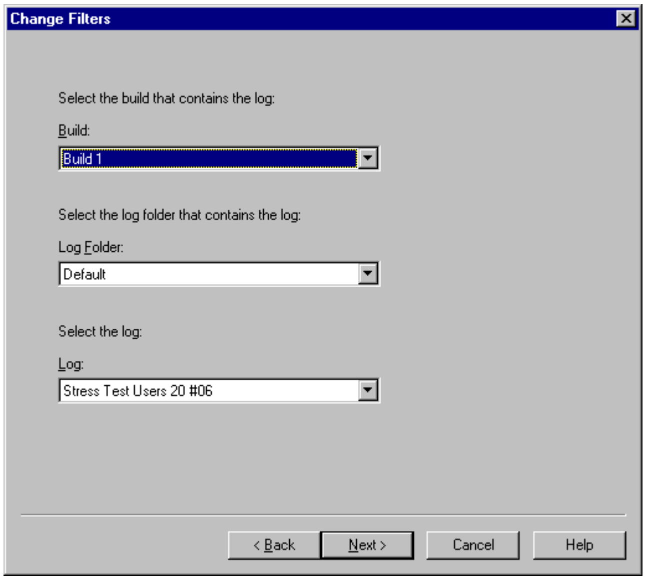
  The test scripts associated with the selected virtual testers on which to filter data.

- 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**.



Change Filters

Select the build that contains the log:

Build: Build 1

Select the log folder that contains the log:

Log Folder: Default

Select the log:

Log: Stress Test Users 20 #06

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.

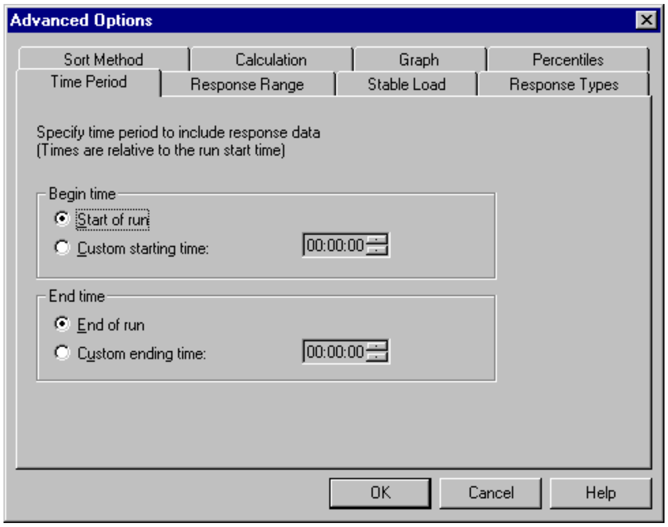
You can also filter the report *after* you run it. For more information, see *Filtering Command IDs that Appear in a Graph* on page 305.

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 **Change Options**.



Specify time period to include response data
(Times are relative to the run start time)

Begin time
- [✓ Start of run]
- Custom starting time: 00:00:00

End time
- [✓ End of run]
- Custom ending time: 00:00:00

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
Graph	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
Response Range	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 *Eliminating Outliers* on page 299.	Command Status, Performance, Response vs. Time, Compare Performance
Response Types	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

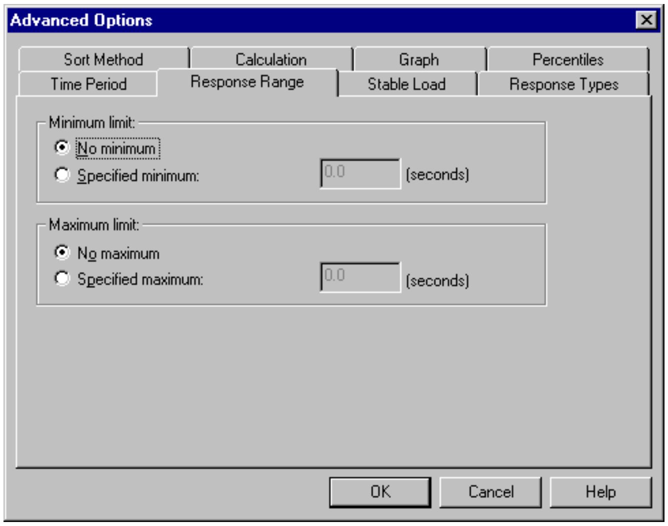
Option	Description	Reports
Sort Method	Sort command IDs numerically or in the order in which they were run. The default is to sort command IDs alphabetically.	Command Status, Performance, Response vs. Time
Stable Load	Specify a number of virtual testers that must be logged on before results are reported. 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 *Reporting on a Stable Load* on page 299. 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 *Reporting on a Dynamic Number of Virtual Testers* on page 300. If you change this option for one report, change the other reports, too, so that the reports reflect the same information.	Command Status, Performance, Response vs. Time
Time Period	Report on a specific portion of the suite run. The default is to report on the entire run.	Command Status, Performance, Response vs. Time
Calculation	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
Response Status	Include only passed responses, or only failed responses. The default is to include all responses.	Response vs. Time
Summary	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
Percentiles	Change how the response times are grouped. Generally, the defaults of 50, 70, 80, 90, and 95 are adequate.	Performance

Eliminating Outliers

Reports may contains some values, called *outliers*, that are completely out of the normal range. For example, suppose 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.



Minimum limit:
[✓ No minimum]
Specified minimum: 0.0 (seconds)

Maximum limit:
[✓ No maximum]
Specified maximum: 0.0 (seconds)

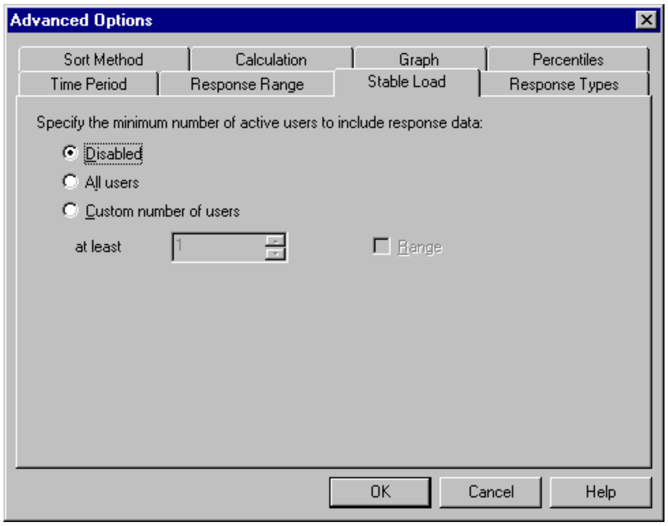
Note: Consider carefully whether to remove data points from graphs. While at times outlying data may non-representative, in other cases outliers could be indicative of other performance issues.

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 are probably 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:

- In the **Stable Load** tab of the Advanced Options dialog box, specify the minimum number of virtual testers that you consider to be a stable load.



Specify the minimum number of active users to include response data:

[✓ Disabled]
All users
Custom number of users

at least 1

Range

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, and 151–200.

To report on a dynamic number of virtual testers:

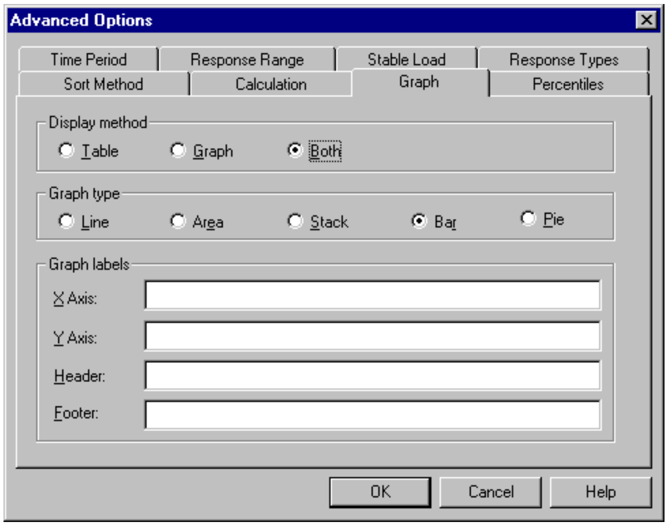
- In the **Stable Load** tab of the Advanced Options dialog box, specify the number of virtual testers.
- In the **Response Range** tab of the Advanced Options dialog box, specify a range for a dynamic number of virtual testers.

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:

- In the **Graph** tab of the Advanced Options dialog box, specify the graph type after filtering the report on the command ID.



Display method: Table Graph [✓ Both]

Graph type: Line Area Stack [✓ Bar] Pie

Graph labels
X Axis:
Y Axis:
Header:
Footer:

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 must be set up to collect the information on computer resources before you can report on them. When you run a suite, the Run Suite dialog box appears, and you must select the **Monitor resources** check box.

To map computer resources onto response time:

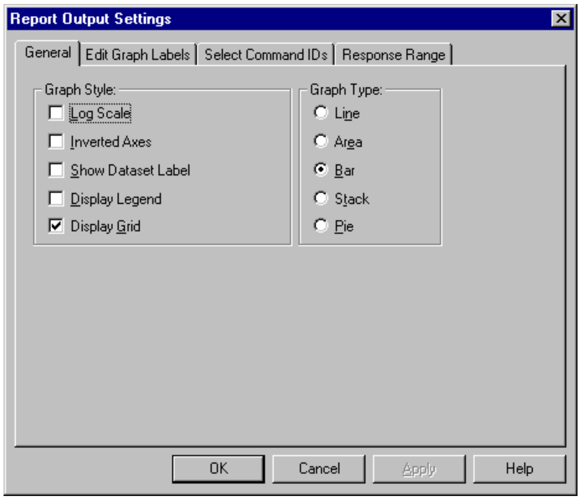
- Right-click on the graph of the Response vs. Time report, and then click **Show Resources**.

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**.


Report Output Settings

Graph Style:
- [ ] Log Scale
- [ ] Inverted Axes
- [ ] Show Dataset Label
- [ ] Display Legend
- [x] Display Grid

Graph Type:
- ○ Line
- ○ Area
- ● Bar
- ○ Stack
- ○ Pie

Note: The available options for changing the type or appearance of a graph vary according to the type of report you selected.

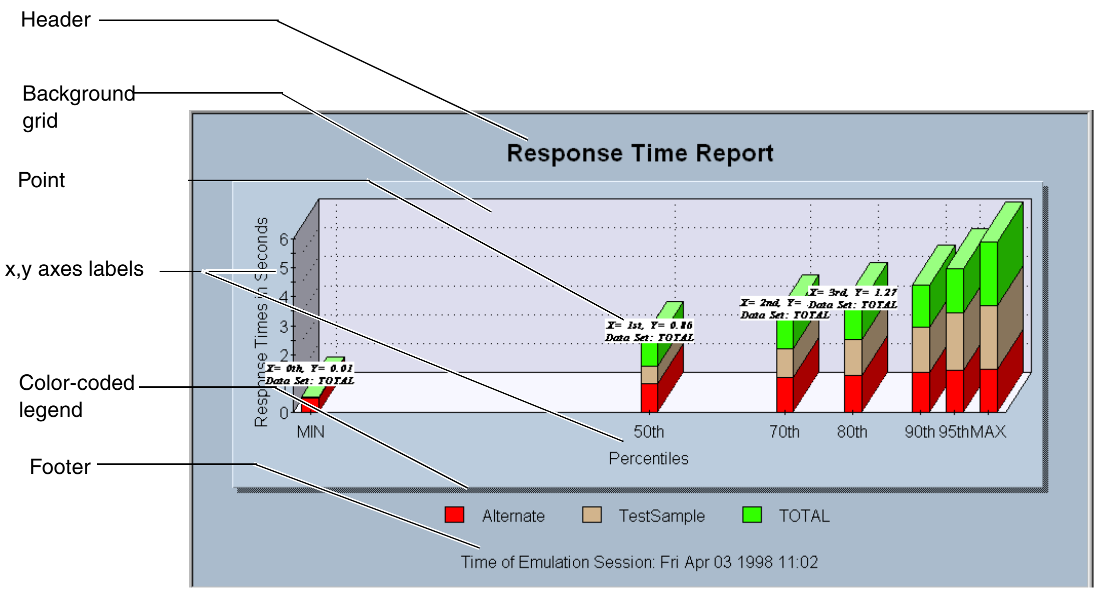
From this dialog box, you can:

- 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 select 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:

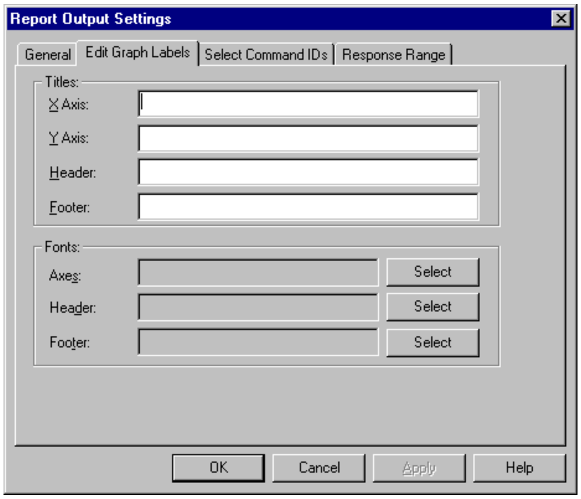
Action	Mouse/Key sequence	Other required action
Enlarge a graph's size.	CTRL-BUTTON1 BUTTON2	Drag the mouse toward the bottom of the graph.
Change a graph's position.	SHIFT-BUTTON1 BUTTON2	Move the mouse to reposition the graph.
Zoom in on a graph's axes.	SHIFT-BUTTON1	Draw a box around the area to zoom, 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**.

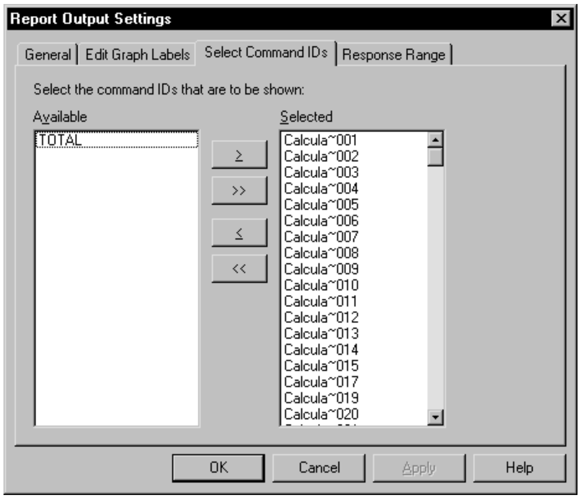


Filtering Command IDs that Appear in a Graph

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**.



Select the command IDs that are to be shown:

Available

TOTAL

Selected

Calcula~001
Calcula~002
Calcula~003
Calcula~004
Calcula~005
Calcula~006
Calcula~007
Calcula~008
Calcula~009
Calcula~010
Calcula~011
Calcula~012
Calcula~013
Calcula~014
Calcula~015
Calcula~017
Calcula~019
Calcula~020

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.

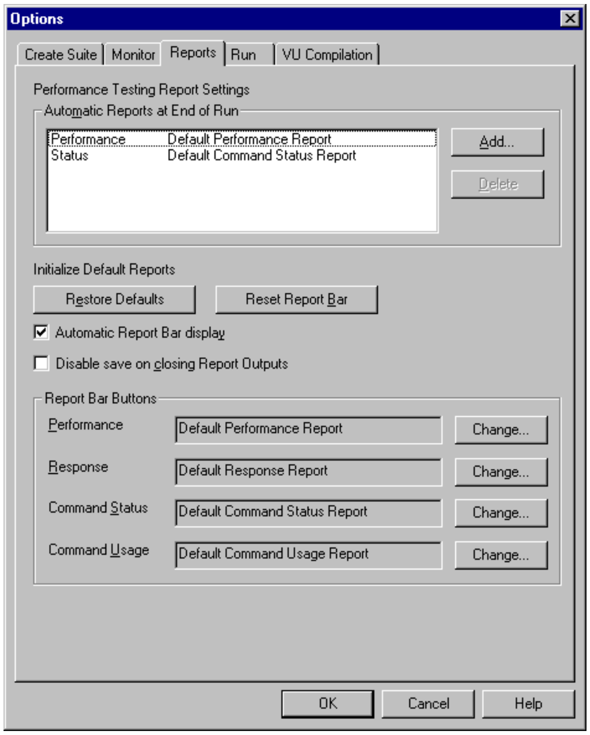
You also can change the reports that TestManager runs when you click a Report bar button. For example, instead of TestManager running the default Performance report, you can have it run a Performance report that you have defined.

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.


Options

Create Suite | Monitor | Reports | Run | VU Compilation

Performance Testing Report Settings

Automatic Reports at End of Run

Performance	Default Performance Report
Status	Default Command Status Report

Initialize Default Reports

☑ Automatic Report Bar display

☐ Disable save on closing Report Outputs

Report Bar Buttons

Performance	Default Performance Report
Response	Default Response Report
Command Status	Default Command Status Report
Command Usage	Default Command Usage Report

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

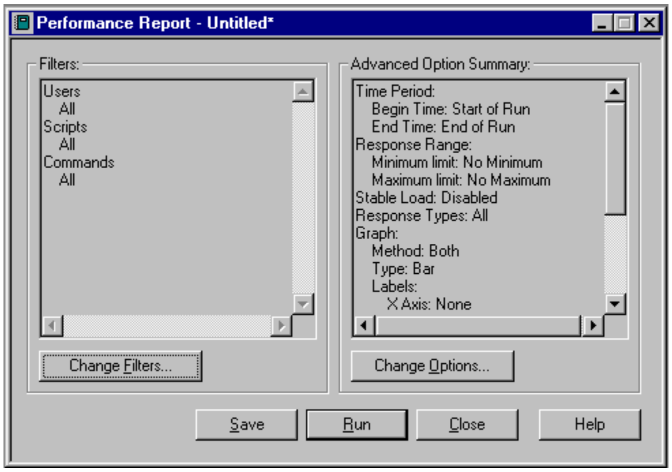
This section discusses the five kinds of performance reports available in TestManager.

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**.



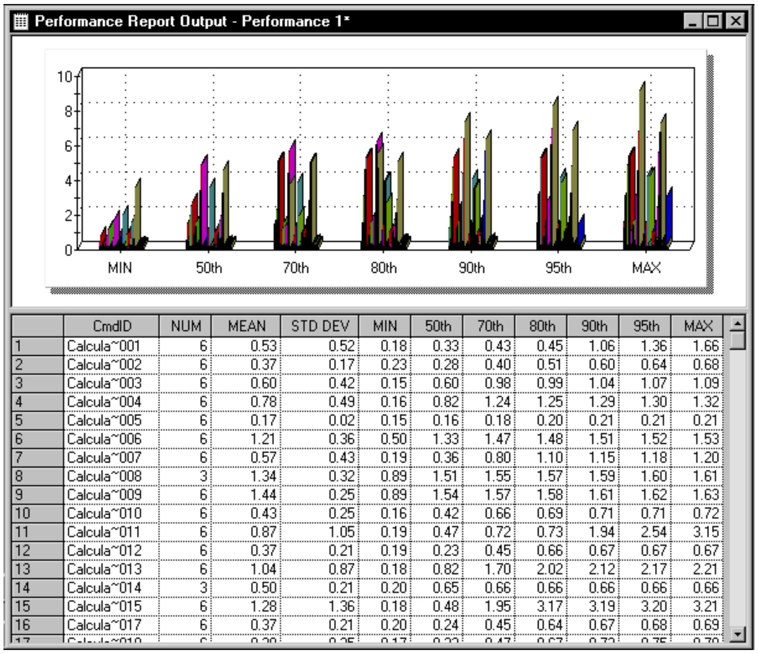
Users
All
Scripts
All
Commands
All

Time Period:
Begin Time: Start of Run
End Time: End of Run
Response Range:
Minimum limit: No Minimum
Maximum limit: No Maximum
Stable Load: Disabled
Response Types: All
Graph:
Method: Both
Type: Bar
Labels:
X Axis: None

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 could 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.



	CmdID	NUM	MEAN	STD DEV	MIN	50th	70th	80th	90th	95th	MAX
1	Calcula~001	6	0.53	0.52	0.18	0.33	0.43	0.45	1.06	1.36	1.66
2	Calcula~002	6	0.37	0.17	0.23	0.28	0.40	0.51	0.60	0.64	0.68
3	Calcula~003	6	0.60	0.42	0.15	0.60	0.98	0.99	1.04	1.07	1.09
4	Calcula~004	6	0.78	0.49	0.16	0.82	1.24	1.25	1.29	1.30	1.32
5	Calcula~005	6	0.17	0.02	0.15	0.16	0.18	0.20	0.21	0.21	0.21
6	Calcula~006	6	1.21	0.36	0.50	1.33	1.47	1.48	1.51	1.52	1.53
7	Calcula~007	6	0.57	0.43	0.19	0.36	0.80	1.10	1.15	1.18	1.20
8	Calcula~008	3	1.34	0.32	0.89	1.51	1.55	1.57	1.59	1.60	1.61
9	Calcula~009	6	1.44	0.25	0.89	1.54	1.57	1.58	1.61	1.62	1.63
10	Calcula~010	6	0.43	0.25	0.16	0.42	0.66	0.69	0.71	0.71	0.72
11	Calcula~011	6	0.87	1.05	0.19	0.47	0.72	0.73	1.94	2.54	3.15
12	Calcula~012	6	0.37	0.21	0.19	0.23	0.45	0.66	0.67	0.67	0.67
13	Calcula~013	6	1.04	0.87	0.18	0.82	1.70	2.02	2.12	2.17	2.21
14	Calcula~014	3	0.50	0.21	0.20	0.65	0.66	0.66	0.66	0.66	0.66
15	Calcula~015	6	1.28	1.36	0.18	0.48	1.95	3.17	3.19	3.20	3.21
16	Calcula~017	6	0.37	0.21	0.20	0.24	0.45	0.64	0.67	0.68	0.69
17	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

- 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:

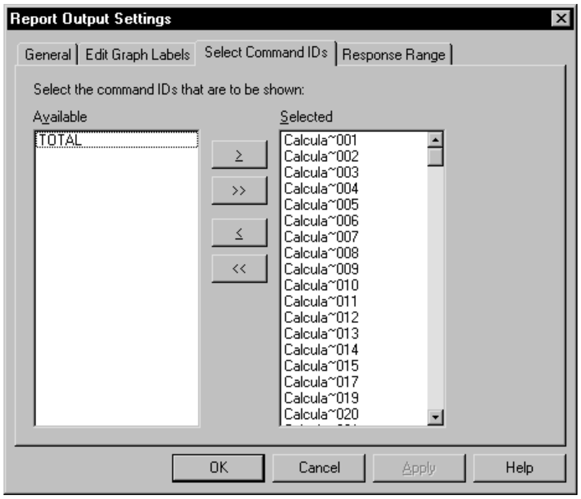
- **Cmd ID** – The command ID associated with the response.
- **NUM** – The number of responses for each command ID.
- **MEAN** – The arithmetic mean of the response times of all responses for each command ID.
- **STD DEV** – The standard deviation of the response times of all responses for each command 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 a Performance report, right-click on the graph and select **Settings**.



Select the command IDs that are to be shown:

Available
TOTAL

Selected
Calcula~001
Calcula~002
Calcula~003
Calcula~004
Calcula~005
Calcula~006
Calcula~007
Calcula~008
Calcula~009
Calcula~010
Calcula~011
Calcula~012
Calcula~013
Calcula~014
Calcula~015
Calcula~017
Calcula~019
Calcula~020

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 members of the technical support staff. You do the following:

- Decide that the time it takes to gain access to the Web site should generally not exceed 8 seconds.
- Create a suite with a test script that accesses 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.

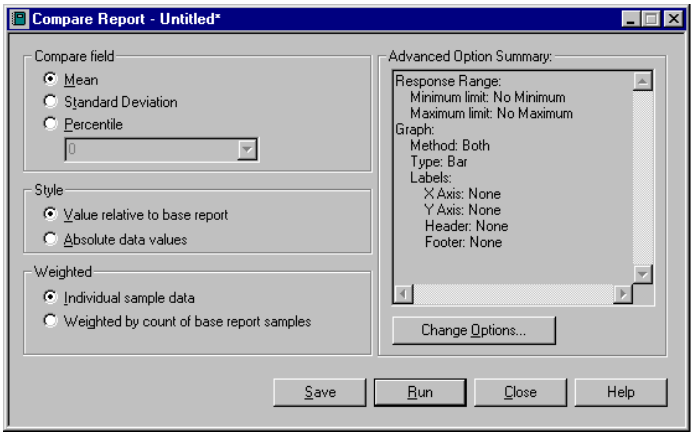
When you run a Compare Performance report, you specify the base Performance report and up to six other Performance reports.

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**.



Compare field
- [✓ Mean]
- Standard Deviation
- Percentile
- 0

Style
- [✓ Value relative to base report]
- Absolute data values

Weighted
- [✓ Individual sample data]
- Weighted by count of base report samples

Advanced Option Summary:

```
Response Range:
  Minimum limit: No Minimum
  Maximum limit: No Maximum
Graph:
  Method: Both
  Type: Bar
  Labels:
    X Axis: None
    Y Axis: None
    Header: None
    Footer: None
```

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

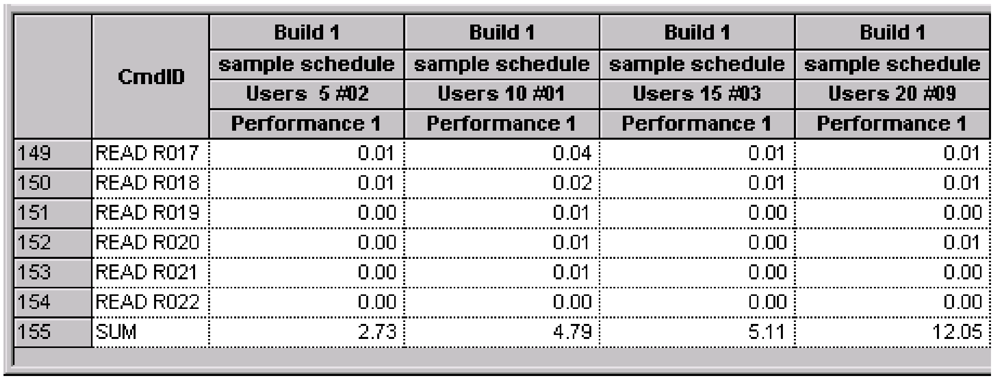
Absolute Compare Performance Reports

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 312, and select 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.

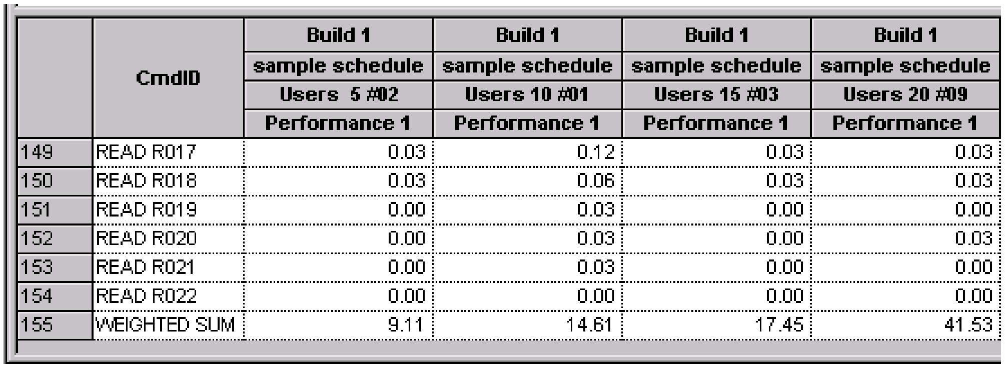
To define a weighted absolute report:

- Follow the steps in *Defining a Compare Performance Report* on page 312, and select 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 312, and select the **Value relative to base report** option.

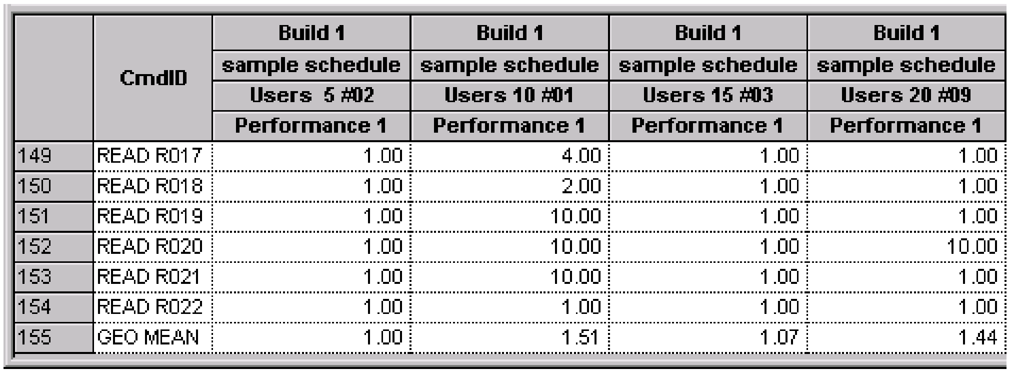
The final line of the report gives the geometric mean of the relative response times for each report. To determine the geometric mean, TestManager multiplies the response times, and then takes a root of the product that is equal to the number of response times.

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, ..., x_k is expressed as:

$$(x_1 x_2 \ldots x_k)^{1/k}$$

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 312, and select the **Value relative to base report** and the **Weighted by count of base report samples** options.

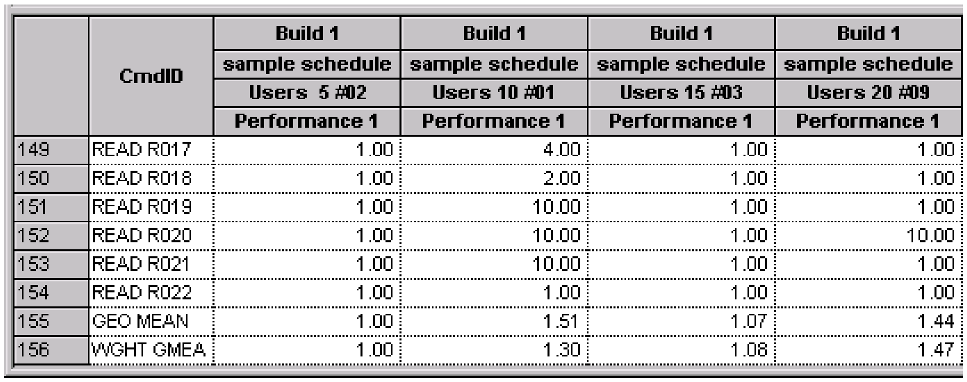
The weighted geometric mean differs from the geometric mean in that it takes into account the frequency with which the different command IDs occur. Frequently used command IDs have a greater influence on the weighted geometric mean than infrequently used ones—in contrast to the geometric mean, where 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, ..., xk with frequencies (weights) of f_1, f_2, ..., fk, where $f_1 + f_2 + ... + fk = N$, is expressed as:

$$(x_1^{f_1} x_2^{f_2} \ldots x_k^{f_k})^{1/N}$$

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:

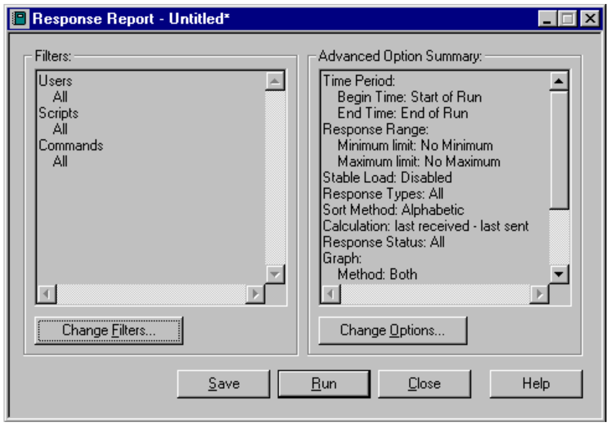
If	Then the Compare Performance report
A command ID is in the base report but does not exist in the other reports.	Lists **n/a** for that command ID in the table and does not include information for that command ID in the report graph.
A command ID is in the report but does not occur in the base report.	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 **0** in the base report, and lists the other results corresponding to that command ID as **Undefn**.
All the response times for a report are listed as **n/a** or **Undefn**.	Lists the geometric mean or sum as **n/a**.

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**.



Users
  All
Scripts
  All
Commands
  All

Time Period:
  Begin Time: Start of Run
  End Time: End of Run
Response Range:
  Minimum limit: No Minimum
  Maximum limit: No Maximum
Stable Load: Disabled
Response Types: All
Sort Method: Alphabetic
Calculation: last received - last sent
Response Status: All
Graph:
  Method: Both

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.

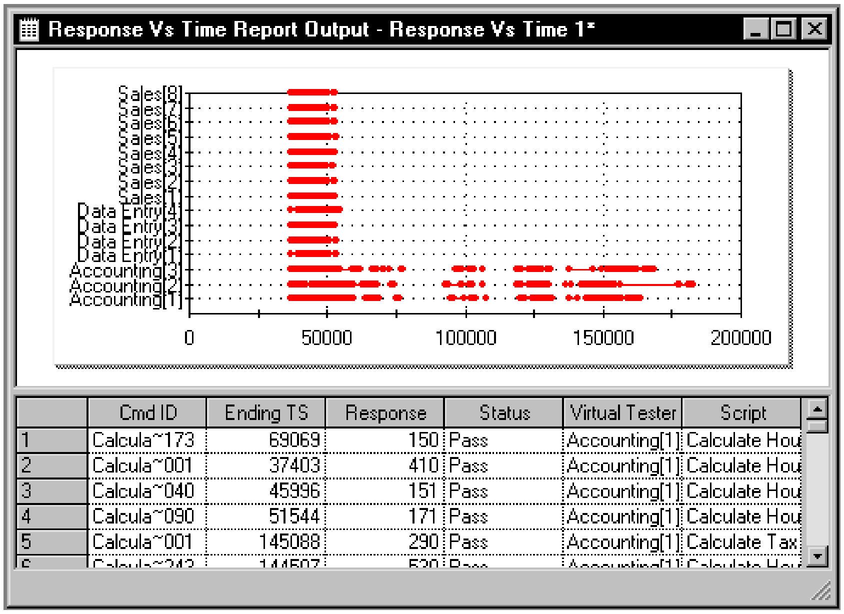
- Checking any spikes in the response time. If the response time is relatively flat except for one or two spikes, you might want to investigate the cause of the spikes.

- 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.



	Cmd ID	Ending TS	Response	Status	Virtual Tester	Script
1	Calcula~173	69069	150	Pass	Accounting[1]	Calculate Hou
2	Calcula~001	37403	410	Pass	Accounting[1]	Calculate Hou
3	Calcula~040	45996	151	Pass	Accounting[1]	Calculate Hou
4	Calcula~090	51544	171	Pass	Accounting[1]	Calculate Hou
5	Calcula~001	145088	290	Pass	Accounting[1]	Calculate Tax
6	Calcula~243	144507	520	Pass	Accounting[1]	Calculate Hou

The graph plots each virtual tester versus the response time in milliseconds. The graph contains many short lines that resemble dots. The lines 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.

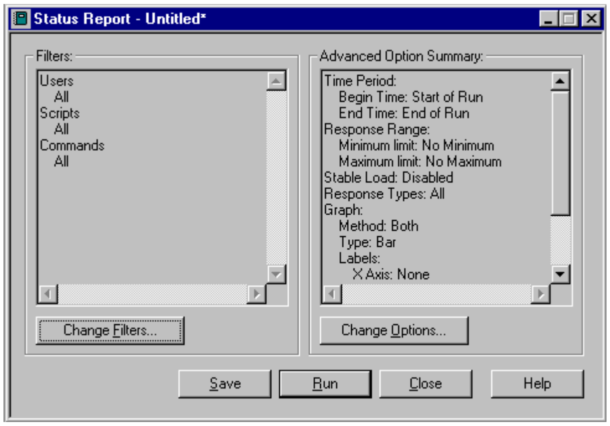
- **Virtual Tester** – The virtual tester corresponding to the response.
- **Script** – The name of the test script corresponding to the response.

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 that it has failed.

To define a new Command Status report:

- Click **Reports > New > Command Status**.

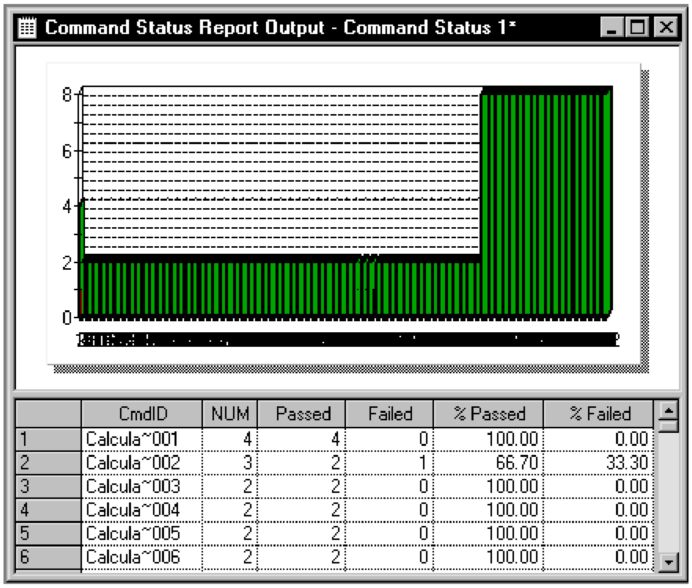


Users
All
Scripts
All
Commands
All

Time Period:
Begin Time: Start of Run
End Time: End of Run
Response Range:
Minimum limit: No Minimum
Maximum limit: No Maximum
Stable Load: Disabled
Response Types: All
Graph:
Method: Both
Type: Bar
Labels:
X Axis: None

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.



	CmdID	NUM	Passed	Failed	% Passed	% Failed
1	Calcula~001	4	4	0	100.00	0.00
2	Calcula~002	3	2	1	66.70	33.30
3	Calcula~003	2	2	0	100.00	0.00
4	Calcula~004	2	2	0	100.00	0.00
5	Calcula~005	2	2	0	100.00	0.00
6	Calcula~006	2	2	0	100.00	0.00

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.

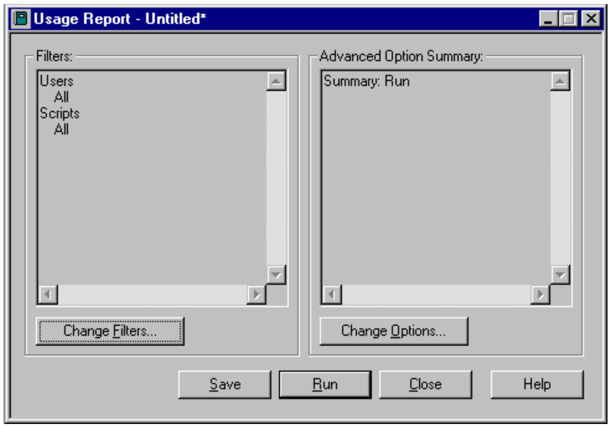
The last line of the report lists the totals for each column.

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.

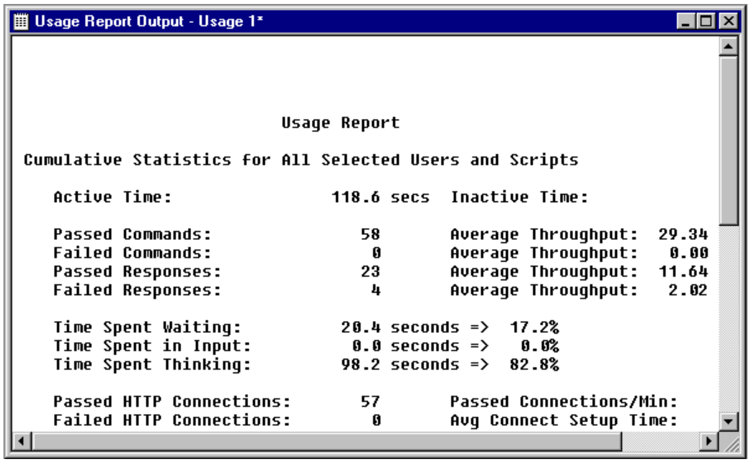
To define a new Command Usage report:

- 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:



```
                          Usage Report

Cumulative Statistics for All Selected Users and Scripts

   Active Time:                118.6 secs  Inactive Time:

   Passed Commands:               58       Average Throughput:  29.34
   Failed Commands:                0       Average Throughput:   0.00
   Passed Responses:              23       Average Throughput:  11.64
   Failed Responses:               4       Average Throughput:   2.02

   Time Spent Waiting:          20.4 seconds =>  17.2%
   Time Spent in Input:          0.0 seconds =>   0.0%
   Time Spent Thinking:         98.2 seconds =>  82.8%

   Passed HTTP Connections:       57       Passed Connections/Min:
   Failed HTTP Connections:        0       Avg Connect Setup Time:
```

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 the 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** – Number of rows received by all reported `sqlnrecv` commands.
- **Received Rows/Sec** – Average number of rows received per second. Derived by dividing the number of rows received by the active time
- **Average Rows/Response** – Average number of rows in the passed and failed responses. Derived by dividing the number of rows received by the number of passed and failed responses.
- **Average Think Time** – Average think time in seconds for `sqlexec` and `sqlprepare` statements only.
- **SQL Execution Commands** – Number of `sqlexec` commands reported.
- **Preparation Commands** – Number of `sqlprepare` commands reported.
- **Rows Processed** – Number of rows processed by all reported `sqlexec` commands.

- **Processed Rows/Sec** – Average number of rows processed per second. Derived by dividing the number of rows processed by the active time.
- **Avg Rows/Execute Cmd** – Average number of rows processed by each `sqlexec` command. Derived by dividing the number of rows processed by the number of `sqlexec` commands reported.
- **Avg Row Process Time** – Average time in milliseconds for processing a row by an `sqlexec` command. Derived by dividing the time spent on `sqlexec` commands by the number of rows processed.
- **Avg Execution Time** – Average time in milliseconds to execute an `sqlexec` 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** – Number of successful HTTP connections established by all reported `http_request` commands.
- **Failed HTTP Connections** – Number of HTTP connection attempts that failed to establish a connection for all reported `http_request` commands.
- **HTTP Sent Kbytes** – Kilobytes of data sent by reported `http_request` commands.
- **HTTP Received Kbytes** – Kilobytes of data received by reported `http_nrecv` and `http_recv` commands.
- **Sent Kbytes/Connection** – Kilobytes of data sent by reported `http_request` commands per connection. Derived by dividing the kilobytes of data sent by the number of successfully established HTTP connections.
- **Passed Connections/Min** – Number of successful HTTP connections established per minute. Derived by dividing the number of successful HTTP connections by the active time.
- **Avg Connect Setup Time** – Average time, in milliseconds, required to establish a successful HTTP connection. Derived by dividing the total connection time for all recorded `http_request` commands by the number of successful connections.
- **HTTP Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `http_request` commands by the active time.

- **HTTP Recv Kbytes/Sec** – Kilobytes of data received per second. Derived by dividing the kilobytes of data received by all recorded `http_nrecv` and `http_recv` commands by the active time.
- **Recv Kbytes/Connection** – Kilobytes of data received by reported `http_nrecv` and `http_recv` commands per connection. Derived by dividing the kilobytes of data received by the number of successfully established HTTP connections.

If any socket emulation commands were executed, the Command Usage report includes:

- **Passed Socket Connections** – Number of successful socket connections established by all reported `sock_connect` functions.
- **Socket Sent Kbytes** – Kilobytes of data sent by reported `sock_send` commands.
- **Socket Received Kbytes** – Kilobytes of data received by reported `sock_nrecv` and `sock_recv` commands.
- **Passed Connections/Min** – Number of successful socket connections established per minute. Derived by dividing the number of successful socket connections by the active time.
- **Socket Sent Kbytes/Sec** – Kilobytes of data sent per second. Derived by dividing the kilobytes of data sent by all recorded `sock_send` commands by the active time.
- **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** – 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** – 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** – Number of `emulate` commands that report a passed status.
- **Passed emulate Time Spent** – Total time spent, from when the passed `emulate` commands start to where they end.
- **Failed emulate Commands** – Number of `emulate` commands that report a failed status.
- **Failed emulate Time Spent** – Total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Command Usage report includes:

- **Passed testcase Commands** – Number of `testcase` commands that report a passed status.
- **Failed testcase Commands** – Number of `testcase` commands that report a failed status.

Summary Statistics

- **Duration of Run** – Elapsed time from the beginning to the end of the run. The beginning of the run is the time of the first emulation activity among all 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 323.

- **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** – Number of virtual testers in the suite run.
- **Number of stop_time Cmds** – 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** – 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** – 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** – 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** – Number of `emulate` commands that report a passed status.
- **Passed emulate Time Spent** – Total time spent, from when the passed `emulate` commands start to where they end.
- **Failed emulate Commands** – Number of `emulate` commands that report a failed status.
- **Failed emulate Time Spent** – Total time spent, from when the failed `emulate` commands start to where they end.

If any `testcase` emulation commands were executed, the Command Usage report includes:

- **Passed testcase Commands** – Number of `testcase` commands that report a passed status.
- **Failed testcase Commands** – Number `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. This appendix includes the following topics:

- Running more than 245 virtual testers
- Running more than 1000 virtual testers
- Running more than 1000 virtual testers on one NT computer
- Running more than 24 virtual testers on a UNIX Agent
- Controlling TCP port numbers
- Setting up IP aliasing
- Assigning values to system environment variables

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 that 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 that 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)* *N* is the number of virtual testers on the Agent. *open_files* is the number of files explicitly opened within test scripts. *connections* is the number of connections open concurrently.
Total system-wide shared memory (SHMALL/SHMMAX)	724 + 5609*N* + 16*S* + 13*G* + *group_names* bytes *N* is the number of virtual testers on the Agent. *S* is the total number of shared variables in all the test scripts in the suite. *G* is the total number of user groups in the suite. *group_names* 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_semmap=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 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 computer and each Agent.

Connections made from the Local to the Agent computer are always made to a single well-known port on which the 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.
2. If not defined, 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.
2. If not defined, a port dynamically chosen by the system.

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 the following 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 listen on port 8888, add the following 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, because 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 **Suite > Edit Runtime**, and select the **Enable IP Aliasing** check box.

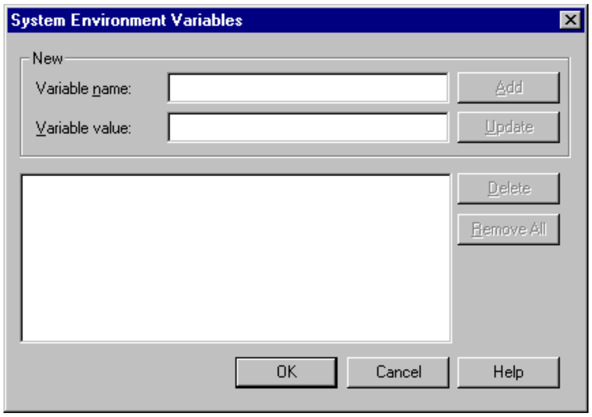
If IP Aliasing is selected in the suite, then at the beginning of a run, the TestManager software on each computer (Local or Agent) queries the system for all available IP addresses. Each suite scheduled to run on that computer is assigned an IP address from that list, in round-robin fashion. In other 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 the TestManager Help.

You can set system environment variables for listed testing platforms as follows:

Testing Platform	System Environment Variable Settings
Oracle on a UNIX Agent	Specify the directory that contains the client software in the variable `ORACLE_HOME`. Example: `ORACLE_HOME = /ora/app/oracle/product/8.0.5` If /var/opt/oracle does not contain tnsnames.ora, assign the pathname of the file to the variable `TNS_ADMIN`. Example: `TNS_ADMIN = /home/uname/oracletest`
Sybase on a UNIX Agent	Specify the directory that contains the client software in the variable `SYBASE`. Example: `SYBASE = /usr/local/sybasec` Specify the directory that contains the Sybase client libraries in the path of one of the following system environment variables: `PATH` (Windows) `LD_LIBRARY_PATH` (Solaris Agents) `SHLIB_PATH` (HP-UX Agents) `LIBPATH` (AIX Agents)

Testing Platform	System Environment Variable Settings

Java on a UNIX Agent	Specify the directory that contains the Java libraries in the variable `LD_LIBRARY_PATH`.
	Example:
	`LD_LIBRARY_PATH=/usr/jre118/lib/linux/native_threads`
	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 `LD_LIBRARY_PATH` in addition to the Java libraries.
	In addition, for Java Developers Kit version1.1, you must also set the following variable:
	`JAVA_COMPILER=NONE`

Local or Agents running TUXEDO test scripts	Specify the directory that contains the client software in the variable `TUXDIR`.
	Set the `NLSPATH` environment variable to the path of the directory that contains the `TUXEDO` message file.
	Set the value of `$TUXDIR/lib` to one of the following system environment variables:
	`LD_LIBRARY_PATH` (Solaris Agents)
	`SHLIB_PATH` (HP-UX Agents)
	`LIBPATH` (AIX Agents)
	For Windows NT Local computers, these must be defined only for TUXEDO client-only installations. The TUXEDO full runtime installation process sets them automatically. For more information, see the TUXEDO installation instructions.
	Set one of the following:
	The Workstation Listener's address to `WSNADDR`.
	Example:
	`WSNADDR=//sparky:36001`
	`WSNADDR=00028CA1C0A8F0D6`
	The Workstation Listener's host name and port to `WSLHOST` and `WSLPORT`. These variables override `WSNADDR`, if set.
	Example:
	`WSLHOST=sparky`
	`WSLPORT=36001`

Testing Platform	System Environment Variable Settings
Local or Agents running TUXEDO test scripts that use FML typed buffer field names	Set a list of FML field table file names to FIELDTBLS. 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 will fail, causing dependent commands to fail. Example: `FIELDTBLS=ct.fldtbl,inv.fldtbl` Set the absolute pathname of the directory containing the FML field table file to FLDTBLDIR. 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, `tux_setbuf_int()`) will fail, causing dependent commands to fail. Example: `FLDTBLDIR=/u1/tuxapp/dat`
Local or Agents running TUXEDO test scripts that use FML32 typed buffer field names	Set a list of FML32 field table file names to FIELDTBLS32. 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 will fail, causing dependent commands to fail. Example: `FIELDTBLS32=ct32.fldtbl,inv32.fldtbl` Set the absolute pathname of the directory containing the FML32 field table files to FLDTBLDIR32. 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 `tux_setbuf_int()`) will fail, causing dependent commands to fail. Example: `FLDTBLDIR32=/u1/tuxapp/dat`
Local or Agents running TUXEDO test scripts that use VIEW, X_COMMON, or X_C_TYPE typed buffers	Set a list of view description file names to VIEWFILES. This variable is used by Agents running test scripts that use VIEW, X_COMMON or X_C_TYPE 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 will fail, causing dependent commands to fail. Example: `VIEWFILES=ct.V,inv.V` Set the absolute pathname of the directory containing the view description files to VIEWDIR. If this variable is not set, `tux_tpalloc()` or `tux_alloc_buf()` calls that try to allocate a buffer of type VIEW, X_COMMON, or X_C_TYPE will fail, causing dependent commands or functions to fail. Example: `VIEWDIR=/u1/tuxapp/dat:/u1/tuxapp/dat2`

Testing Platform	System Environment Variable Settings
Local or Agents running TUXEDO test scripts that use `VIEW32` typed buffers	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. Example: `VIEWFILES32=ct32.V,inv32.V` 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` will fail, causing dependent commands or functions to fail. Example: `VIEWDIR32=/u1/tuxapp/dat`
Solaris Agents running TUXEDO test scripts	Set the TLI network service provider pathname to `WSDEVICE`. This value is typically /dev/tcp. If not set, playback terminates with an error message. Example: `WSDEVICE=/dev/tcp`
INFORMIX on a UNIX Agent computer	Assign a valid entry in the `$INFORMIXDIR/etc/sqlhosts` file to `INFORMIXSERVER`. Assign a value to `INFORMIXDIR`. The value depends on your version of `INFORMIX CLI` and `INFORMIX ESQL/C`.

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	Dates in the format shown. The day portion of the string is always two characters. Days 1 through 9 begin with a blank space. 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. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 01011900 and **Maximum** to 12312050.	Oct 8, 1997 Jun 17, 1964 Nov 10, 1978 If the comma is the delimiter, the values are stored in the datapool as follows: "Oct 8, 1997" "Jun 17, 1964" "Nov 10, 1978"
Date - August 10, 1994	Dates in the format shown. The day portion of the string is always two characters. Days 1 through 9 begin with a blank space. 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. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 01011900 and **Maximum** to 12312050.	October 8, 1997 June 17, 1964 November 10, 1978 If the comma is the delimiter, the values are stored in the datapool as follows: "October 8, 1997" "June 17, 1964" "November 10, 1978"

Standard data type name	**Description**	**Examples**
Date - MM/DD/YY	Dates in the format shown. You can only specify a range of dates in the same century (that is, the year in **Maximum** must be greater than the year in **Minimum**). 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, 1999, set **Minimum** to 010100 and **Maximum** to 123199.	10/08/97 06/17/64 11/10/78 If the slash is the delimiter, the values are stored in the datapool as follows: "10/08/97" "06/17/64" "11/10/78"
Date - MM/DD/YYYY	Dates in the format shown. To include the slashes (/) as ordinary characters rather than as the .csv file delimiter, the dates are enclosed in double quotes when stored in the datapool. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 01011900 and **Maximum** to 12312050.	10/08/1997 06/17/1964 11/10/1978 If the slash is the delimiter, the values are stored in the datapool as follows: "10/08/1997" "06/17/1964" "11/10/1978"
Date - MMDDYY	Dates in the format shown. You can only specify a range of dates in the same century (that is, the year in **Maximum** must be greater than the year in **Minimum**). To set a range of dates from January 1, 1900 through December 31, 1999, set **Minimum** to 010100 and **Maximum** to 123199.	100897 061764 111078
Date - MM-DD-YY	Dates in the format shown. You can only specify a range of dates in the same century (that is, the year in **Maximum** must be greater than the year in **Minimum**). To set a range of dates from January 1, 1900 through December 31, 1999, set **Minimum** to 010100 and **Maximum** to 123199.	10-08-97 06-17-64 11-10-78
Date - MMDDYYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 01011900 and **Maximum** to 12312050.	10081997 06171964 11101978
Date - MM-DD-YYYY	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 01011900 and **Maximum** to 12312050.	10-08-1997 06-17-1964 11-10-1978

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 **Minimum** to 19000101 and **Maximum** to 20501231.	1997/10/08 1964/06/17 1978/11/10 If the slash is the delimiter, the values are stored in the datapool as follows: "1997/10/08" "1964/06/17" "1978/11/10"
Date - YYYYMMDD	Dates in the format shown. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 19000101 and **Maximum** to 20501231.	19971008 19640617 19781110
Date, Julian - DDDYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 1999, set **Minimum** to 00100 and **Maximum** to 36599.	28197 16964 31478
Date, Julian - DDDYYYY	Dates in the format shown. DDD is the total number of days that have passed in a year. For example, January 1 is 001, and February 1 is 032. To set a range of dates from January 1, 1900 through December 31, 2050, set **Minimum** to 0011900 and **Maximum** to 3652050.	2811997 1691964 3141978
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 **Minimum** to 00001 and **Maximum** 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 **Minimum** to 1900001 and **Maximum** to 2050365.	1997281 1964169 1978314

Standard data type name	**Description**	**Examples**
Float - X.XXX	Positive and negative decimal numbers in the format shown. Set **Length** to the number of decimal places to allow (up to 6). Set **Minimum** and **Maximum** to the range of numbers to generate. To generate numbers with more than 9 digits (the maximum allowed with the Integers - Signed data type), use the Float - X.XXX data type and set **Decimals** to 0.	243.63918 -95.99 155075028157503
Float - X.XXXE+NN	Positive and negative decimal numbers in the exponential notation format shown. Set **Length** to the number of decimal places to allow (up to 6). Set **Minimum** and **Maximum** to the range of numbers to generate.	4.0285177E+068 -3.2381443E+024 8.8373255E+119
Gender	Either M or F, with no following period.	M F
Hexadecimal	Hexadecimal numbers.	1d6b77 ff 3824e7d
Integers - Signed	Positive and negative whole numbers. This is the default data type. To include negative numbers in the list of generated values, set **Minimum** to the lowest negative number you want to allow. Maximum range: ■ **Minimum** = -999999999 (-999,999,999) ■ **Maximum** = 999999999 (999,999,999) For larger numbers, use a float data type. If you do not specify a range, the default range is 0 through 999,999,999. 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.	1349 -392993 441393316

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 **Length** to 3.	781 412 505
Phone - Exchange	Telephone exchanges. To generate correct exchange lengths, set **Length** 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 **Length** to 4.	2400 8993 8498
Random Alphabetic String	Strings of random uppercase and lowercase letters. **Length** determines the number of characters generated.	AQSEFuOZUIUIpAGsEM DESieAiRFiEqiEIDiicEw edEIDiIcisewsDIEdgP
Random Alphanumeric String	Strings of random uppercase and lowercase letters and digits. **Length** 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 288.	Any values in an ASCII text file.
Space Character	An empty string.	""
State Abbrev. - U.S.	Two-character state abbreviations.	MA CA NC
String Constant	A constant with the value of **Seed**. The datapool column is filled with this one alphanumeric value.	1234 AAA 1b1b
Time - HH.MM.SS	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 **Minimum** to 0 and **Maximum** 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 **Minimum** to 0 and **Maximum** 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 **Minimum** to 0 and **Maximum** 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 **Length** to 5.	02173 95401 84104
Zip Code - 9 Digit	Nine-digit U. S. postal zip codes.	021733104 954012694 841040190
Zip Code - 9 Digit with Dash	Nine-digit U.S. postal zip codes with a dash between the fifth and sixth digits.	02173-3104 95401-2694 84104-0190
Zoned Decimal	Zoned decimal numbers.	3086036 450 499658196

Data Type Ranges

The following table 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
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

ManualTest Web Execution C

This appendix provides an overview of Rational ManualTest Web Execution, a feature of Rational TestManager, that lets you run a test case with a manual test script implementation from a Web browser. The appendix includes the following topics:

- About ManualTest Web execution
- Overview of tasks
- Software requirements
- About shared projects
- Troubleshooting
- How to run a test case from a Web browser
- Viewing the results

About ManualTest Web Execution

A *manual test script* is a set of testing instructions that are run by a human tester. A manual test script can consist of steps and verification points that you type into a manual test script using Rational ManualTest. After you create a manual test script and associate it with a test case, creating a test case implementation, you can run the test case from a Web browser. For information about creating a manual test script, see *Creating Manual Test Scripts* on page 61. For information about test case implementations, see *Associating an Implementation with a Test Case* on page 65.

With the **ManualTest Web Execution** component of Rational TestManager, you can:

- Run a test case with a manual test script implementation from a Web browser. The advantage to using the ManualTest Execution feature of TestManager is that you only need Web browser software to run a test case with a manual test script implementation.
- Indicate results and add comments as you perform each task in a manual test script. These results appear in a test log.
- Include your test case results in a TestManager report.

When you run a test case, you view and record the results of performing the manual steps and verification points in the manual test script.

Note: You can run a test case from a Web browser only if it has a manual test script implementation; you cannot run a test case with an automated test script implementation from a Web browser. If a test case has both a manual test script and an automated test script implementation, only the manual test script runs from a Web browser.

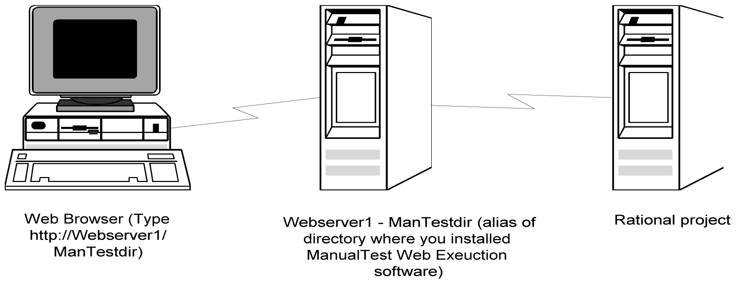
To run a test case with a manual test script implementation, you or an administrator install and configure a Web server with ManualTest Web Execution software and configure a Web browser on each client that will access a Rational project. To run a test case using a Web browser, you type the `machinename`, the network name of the Web server, and the `alias`, the name of an alias for the directory where you or your administrator installed the ManualTest Web Execution software on the Web server.

For example:

```
http://Webserver1/ManTestdir
```

For information about installing and configuring a Web server, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual.

The following figure shows a Web browser accessing the Web server to run a test case stored in a shared project:


Web Browser (Type http://Webserver1/ ManTestdir)
Webserver1 - ManTestdir (alias of directory where you installed ManualTest Web Exeuction software)
Rational project

Overview of Tasks

The following table lists the tasks that you perform to run a test case with a manual test script implementation from a Web browser and where you can find information about each task.

Task	For more information, see
1 Install and configure a Web server.	The *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual
2 Set up a Web browser.	*Setting Up a Web Browser* on page 355
3 Use Rational ManualTest to create a manual test script.	*Creating Manual Test Scripts* on page 61
4 Run a test case from a Web browser.	*How to Run a Test Case from a Web Browser* on page 356
5 View the results in the Test Log window of TestManager.	*Opening a Test Log in TestManager* on page 128

Software Requirements

Make sure that your Web server and Web browsers conform to the following minimum requirements:

Web Server Requirements

Make sure that your Web server conforms to the following minimum requirement:

- Windows ME, Windows 2000 Professional, Windows 2000 Server, Windows 2000 Advanced Server, Windows 98, Windows 95, Windows NT 4.0 Workstation, or Windows NT 4.0 Server.
- Microsoft Internet Explorer 5.0 or later.

The following table lists the operating systems that you can use for a Web server and the appropriate Web server software that you must install for each operating system.

For operating system	Install this software
All versions of Windows 2000 and Windows ME	Microsoft Internet Information Services 5.0 (IIS 5.0) from the Windows 2000 CD
Windows 98, Windows 95, or the Windows NT 4.0 Workstation	Microsoft Personal Web Server (PWS) from the Windows NT 4.0 Option Pack (available from Microsoft, www.microsoft.com)
Windows NT 4.0 Server	Microsoft Internet Information Server (IIS) from the Windows NT 4.0 Option Pack (available from Microsoft, www.microsoft.com)

Note: We recommend that you use Windows 2000 Server, Windows 2000 Advanced Server, or Windows NT 4.0 Server, as the Web server platform. You can not use a shared or networked project with the Windows 2000 Professional or Windows NT 4.0 Workstation.

Web Browser Requirements

To access the Web server as a client, use one of the following Web browsers:

- Netscape Navigator 4.0 or later
- Microsoft Internet Explorer 4.0 or later

About Shared Projects

We recommend that when you create a project, you make it a shared project so others can access your manual test scripts from a Web browser. To share a project, create the project in a shared directory and use the Uniform Naming Convention (UNC) for the directory name. (For more information about creating a shared directory, see the *Using the Rational Administrator* manual or the Rational Administrator Help.)

Setting Up a Web Browser

You can use Netscape Navigator 4.0 (or later) or Microsoft Internet Explorer 4.0 (or later) as your Web browser to run a test case. You can use your Web browser on a system running any operating system software.

Netscape Navigator

To set up a Netscape Navigator browser to run a test case:

1. Start Netscape Navigator.
2. Click **Edit > Preferences**. Under Category, click **Advanced**.
3. Double-click **Advanced**, and then click **Cache** to display the Cache panel.
4. In the Cache panel, click **Every time**.
5. Click **OK**.

Microsoft Internet Explorer

To set up a Microsoft Internet Explorer browser to run a test case:

1. Start Internet Explorer.
2. Do one of the following:
   - For Internet Explorer 5.0 or later, click **Tools > Internet Options**.
   - For Internet Explorer 4.0, click **View > Internet Options**.
3. Click the **General** tab.
4. Under **Temporary Internet files**, click **Settings**.
5. Under **Check for newer versions of stored pages**, click **Every visit to the page**.
6. Click **OK**. Click **OK** again.

How to Run a Test Case from a Web Browser

You run a test case with a manual test script implementation from a Web browser with the Rational ManualTest Web Execution component of TestManager. You can indicate results and add comments as you perform each task in a manual test script.

To run a test case with a manual test script implementation from a Web browser:

1 Start a Web browser, either Netscape Navigator 4.0 (or later) or Microsoft Internet Explorer 4.0 (or later).

2 Connect to the Web server by typing the following:
`http://machinename/alias`

where *`machinename`* is the network name of the Web server, and *`alias`* is the name of an alias for the directory where you or your administrator installed your Rational software on the Web server.

For example:
`http://Webserver/TM`

For information about setting up an alias for a Web server running any version of Windows, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual. For a Web server running Windows NT 4.0 Server, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual.

3 Log into a Rational project.

a Type your user name and password for the project that contains the test case that you want to run. If you do not know the user name and password, see your project administrator.

b Select a project from the list of projects. You need privileges to access a shared project. (For information about creating a shared project, see the *Using the Rational Administrator* manual or the Rational Administrator Help.)

Note: If your project does not appear in the list of projects, use the Rational Administrator software to register your project. For information about registering an existing project, see the Rational Administrator Help.

c Click **OK**. The hierarchy of test cases appears. Only those test cases with manual test script implementations appear in the hierarchy.

4 Navigate through the Test Case tree and click a test case with a manual test script implementation that you want to run.

For detailed information about using the ManualTest Web Execution component, see the ManualTest Web Execution Help.

Viewing the Results

To view the results in the Test Log window of TestManager, see *Opening a Test Log in TestManager* on page 128.

Troubleshooting

This section lists some problems that you may experience when running a test case from a Web browser and how to correct each problem.

Note: The error messages in this troubleshooting section are ManualTest Web Execution error messages, not Web browser error messages.

If you have problems with your Web server, check to make sure that your Web server meets the software requirements. For information, see *Software Requirements* on page 353.

Problem – Your Rational projects do not appear when you log into a Rational project. (You type http://*`machinename/alias`* and log into a Rational project.)
Error message – None.
Solution – All web clients use the same user account to access a Rational project either on a Web server (if the project is on the Web server), or on the domain (to access shared projects on other systems in the domain). There are two things to check:

- Check to make sure that the user account on the Web server or on the domain (for shared projects) has privileges to read and write into a Rational project. (Ask your administrator to check the privileges of the user account that the administrator sets up on the Web server.)
- To allow Web clients access, you must also log into this account when you create a new Rational project or register an existing Rational project using the Rational Administrator.

Problem – You cannot connect from a Web browser to a Web server running the Microsoft Personal Web Server (PWS).
Error message – None.
Solution – If you restart a Web server running PWS, PWS may not start automatically when the server restarts. This is an intermittent problem. To fix the problem, restart PWS.

To restart PWS:

1 Click **Start > Programs > Windows NT 4.0 Option Pack > Microsoft Personal Web Server > Personal Web Manager**.

2 Under **Publishing**, click **Start**.

3 Click **Properties > Exit**.

Problem – When you log into a project from a Web browser, you get the following error message.
Error message – Unable to connect to project.
Solution – Make sure that the Web server privileges are set correctly. For information, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual.

Problem – You get an error message when you select a manual test script.
Error message – Error message that includes `Server.ObjectCreate` in the message.
Solution – Make sure that you or the Web server administrator installs Microsoft Internet Explorer 5.0 or later on the Web server.

Problem – When you type text in a dialog box and submit it, you get erratic behavior. Alternatively, when you open a manual test script, results and comments are already filled in from the last session.
Error message – None.
Solution – Disable caching on your Web browser. For information about disabling caching, see *Setting Up a Web Browser* on page 355.

Problem – After you connect to the Web server, a Login dialog box appears. In the Login dialog box, the project select list is empty.
Error message – None.
Solution – Create a project and create manual test scripts, or register an existing project that contains manual test scripts.

To create a project or register an existing project:

1 Do one of the following:

   - For IIS, log into the user account of the virtual directory that you configured to run a test case. For information, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual.

   - For PWS, log into the user account that the Web server runs under. For information, see the *Installing Rational Testing Products* manual or the *Installing Rational Suite* manual.

2 Start the Rational Administrator and create a new project, or register an existing project. For information about creating or registering a project, see the *Using the Rational Administrator* manual or the Rational Administrator online Help.

3 If you create or register a shared project, make sure that the privileges for the project directory are set for the virtual directory user account for IIS, or for the user account that the Web server runs under for PWS.

4 Restart the Web server.

Index