Getting Started with Rational Robot

Version 2000.02.10

Rational®
the e-development company™

Getting Started with Rational Robot

Revised 04/2000

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

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

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

P/N 800-023364-000

Contents

Introduction

If you've installed Rational Robot, you're on your way to ensuring that your business-critical client/server and Web applications perform exactly as you want.

With Rational Robot, you can plan, develop, and execute functional tests for your Windows NT 4.0, Windows 2000, Windows 98, and Windows 95 applications. Thanks to Robot's cross-Windows technology, tests created for applications on one Windows platform can be used to test applications on the others.

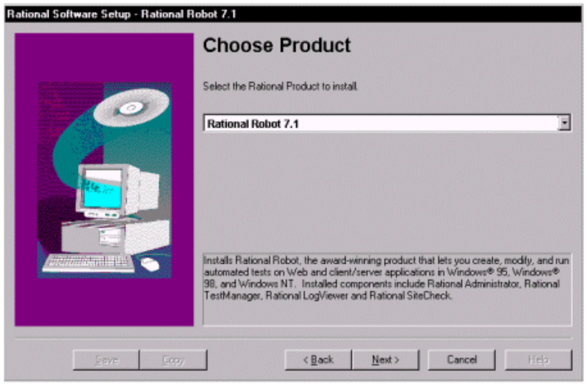
RATIONAL
SOFTWARE
Rational Robot
VERSION 7.1
Rational Software
18880 Homestead Road • Cupertino, CA 95014
(408) 863-9900 • (800) 728-1212
www.rational.com
© Copyright 1998 Rational Software Corporation. ALL RIGHTS RESERVED.
See "About Box" for legal notices.
A Rational Suite Product

Here are some of the things that Rational Robot's functional testing lets you do:

- Test the appearance and state of GUI objects and the interactions between client applications and server databases.
- Test Web and e-commerce applications developed using Java and Dynamic HTML (DHTML).
- Test client/server applications developed using Microsoft Visual Basic, Visual C++, Oracle Developer/2000, and Sybase PowerBuilder.
- Test ERP applications developed by PeopleSoft, Oracle, and SAP.

Take This Path to Successful Functional Testing

1. Install Rational Robot as part of one of these packages—Rational Suite TestStudio, Rational TeamTest, or Rational Robot.



Rational Software Setup - Rational Robot 7.1

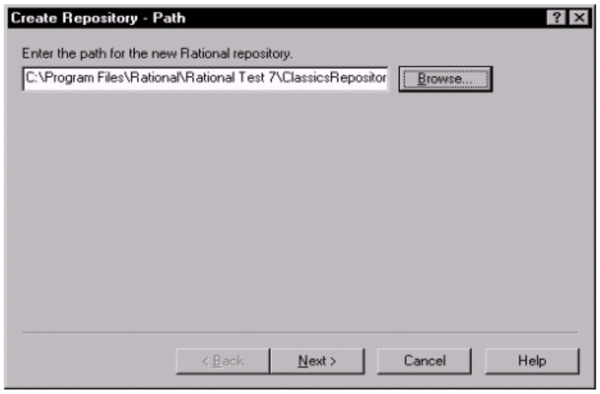
Choose Product

Select the Rational Product to install.

Rational Robot 7.1

Installs Rational Robot, the award-winning product that lets you create, modify, and run automated tests on Web and client/server applications in Windows® 95, Windows® 98, and Windows NT. Installed components include Rational Administrator, Rational TestManager, Rational LogViewer and Rational SiteCheck.

2. Create a Rational repository with the Rational Administrator to store your test assets.

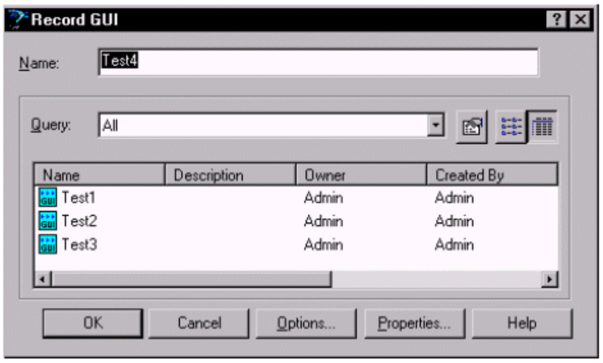


Create Repository - Path

Enter the path for the new Rational repository.

C:\Program Files\Rational\Rational Test 7\ClassicsRepositor

3. Record and play back scripts in Rational Robot.



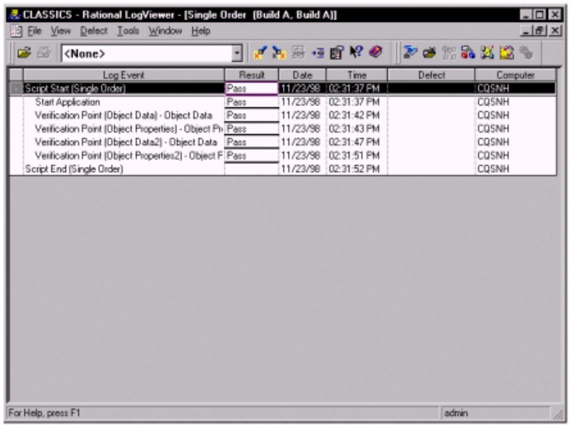
Record GUI

Name: Test4

Query: All

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

4. Review the results of playback in the Rational LogViewer and Comparators.

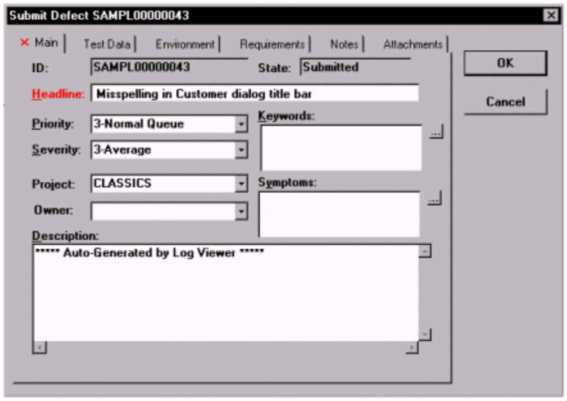


CLASSICS - Rational LogViewer - [Single Order (Build A, Build A)]

Log Event	Result	Date	Time	Defect	Computer
Script Start (Single Order)	Pass	11/23/98	02:31:37 PM		CQSNH
Start Application	Pass	11/23/98	02:31:37 PM		CQSNH
Verification Point (Object Data) - Object Data	Pass	11/23/98	02:31:42 PM		CQSNH
Verification Point (Object Properties) - Object Pr	Pass	11/23/98	02:31:43 PM		CQSNH
Verification Point (Object Data2) - Object Data	Pass	11/23/98	02:31:47 PM		CQSNH
Verification Point (Object Properties2) - Object F	Pass	11/23/98	02:31:51 PM		CQSNH
Script End (Single Order)		11/23/98	02:31:52 PM		CQSNH

For Help, press F1 admin

5. Create defects about any failures in your scripts with Rational ClearQuest.

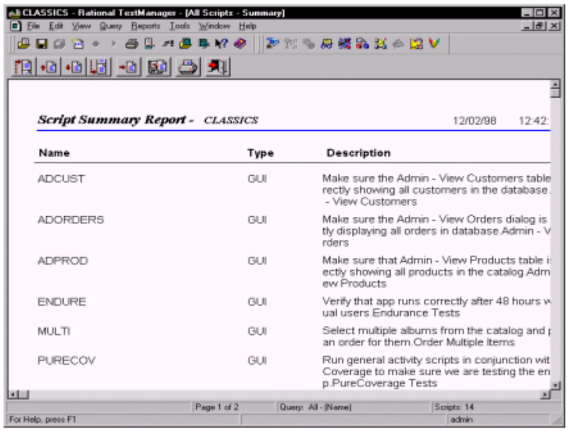


Submit Defect SAMPL00000043

Main | Test Data | Environment | Requirements | Notes | Attachments
OK
Cancel

ID: SAMPL00000043 State: Submitted
Headline: Misspelling in Customer dialog title bar
Priority: 3-Normal Queue
Severity: 3-Average
Keywords:
Project: CLASSICS
Symptoms:
Owner:
Description:
***** Auto-Generated by Log Viewer *****

6. Analyze the results of your tests by creating reports in TestManager, LogViewer, and ClearQuest.



CLASSICS - Rational TestManager - [All Scripts - Summary]

Script Summary Report - CLASSICS 12/02/98 12:42:

Name	Type	Description
ADCUST	GUI	Make sure the Admin - View Customers table rectly showing all customers in the database. - View Customers
ADORDERS	GUI	Make sure the Admin - View Orders dialog is tly displaying all orders in database.Admin - V rders
ADPROD	GUI	Make sure that Admin - View Products table i ectly showing all products in the catalog.Adm ew Products
ENDURE	GUI	Verify that app runs correctly after 48 hours w ual users.Endurance Tests
MULTI	GUI	Select multiple albums from the catalog and p an order for them.Order Multiple Items
PURECOV	GUI	Run general activity scripts in conjunction wit Coverage to make sure we are testing the en p.PureCoverage Tests

Page 1 of 2 Query: All - (Name) Scripts: 14
For Help, press F1 admin

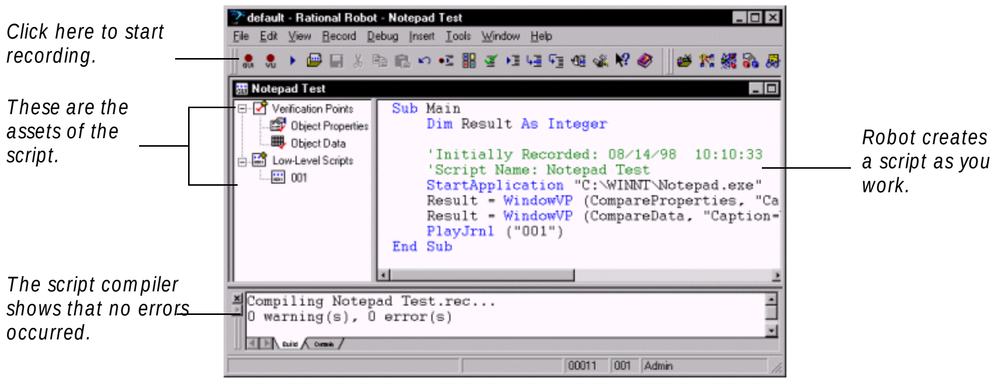
How Does Rational Robot Work?

Rational Robot lets all members of your development and testing teams implement a complete and effective testing methodology.

Robot replaces the repetitive, often error-prone process of manual testing with the software tools to automate your testing effort. With Robot's automated functional testing, you save time and ensure that your testing process produces predictable and accurate results.

With Robot, you can start recording tests in as few as two mouse clicks. After recording, Robot plays back the tests in a fraction of the time it would take to repeat the steps manually.

Robot's **Object-Oriented Recording** technology lets you generate scripts quickly—simply by running and using the application-under-test. Object-Oriented Recording identifies objects by their internal object names, not by screen coordinates. And, if objects in successive builds change locations or their text changes, Robot still finds and tests them during playback.


Click here to start recording.
These are the assets of the script.
The script compiler shows that no errors occurred.
Robot creates a script as you work.
default - Rational Robot - Notepad Test
File Edit View Record Debug Insert Tools Window Help
Notepad Test
Verification Points
Object Properties
Object Data
Low-Level Scripts
001
00011 001 Admin

```
Sub Main
    Dim Result As Integer

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

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

Robot's **Object Testing** technology lets you test standard Windows objects and development environment (IDE) objects, whether they are visible or hidden in the interface. With Object Testing, you can test hundreds, even thousands of properties for an object as well as an object's data.

Using Rational Tools for Successful Software Testing

Rational Robot is part of a suite of integrated tools that deliver a comprehensive solution for the entire functional testing process—from writing and maintaining requirements to creating effective test scripts to tracking defects and analyzing results.

This suite of Rational Test tools includes:

Rational Administrator – Use to administer Rational Test products and components.

Rational TestManager – Use to plan tests and manage test assets.

Rational LogViewer and Comparators – Use to review and analyze test results.

Rational TestFactory – Use to automatically generate scripts that provide extensive application coverage.

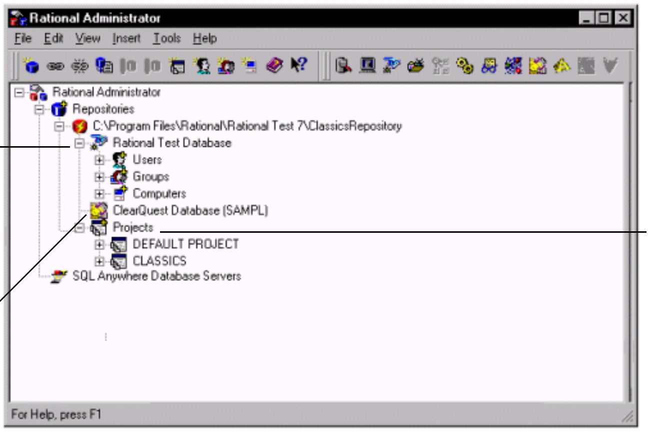
Rational SiteCheck – Use to manage Internet and intranet Web sites.

Rational ClearQuest – Use to track and manage change requests throughout the development and testing process.

Rational Purify, Visual Quantify, Visual Pure Coverage – Use to perform run-time error-checking, determine application performance, and analyze code coverage during playback.

Managing Repositories with the Rational Administrator

The Rational Administrator lets you create and manage **Rational repositories** used for storing application testing information. Each repository consists of one or more databases and several directories of files.


Rational Administrator
File Edit View Insert Tools Help
Rational Administrator
Repositories
C:\Program Files\Rational\Rational Test 7\ClassicsRepository
Rational Test Database
Users
Groups
Computers
ClearQuest Database (SAMPL)
Projects
DEFAULT PROJECT
CLASSICS
SQL Anywhere Database Servers
For Help, press F1

Each repository consists of a Rational Test database for storing test information.

A repository can also contain a ClearQuest database for storing defect information.

Projects in the repository help you organize your testing information and resources for easy tracking.

Planning and Managing Tests with Rational TestManager

Rational TestManager helps you plan testing strategies and manage your software testing effort. You can use TestManager to track information through all phases of the software development, test, and revision cycles.

If you want, you can skip using TestManager and simply start recording with Robot. But the preferred usage model for any testing activity is to plan first. This lets you know the size of the testing effort at the beginning of a project. You'll also be able to assign the development of each part of the testing project to the appropriate person. Using TestManager to plan ahead ensures that your testing effort will run as smoothly as possible.

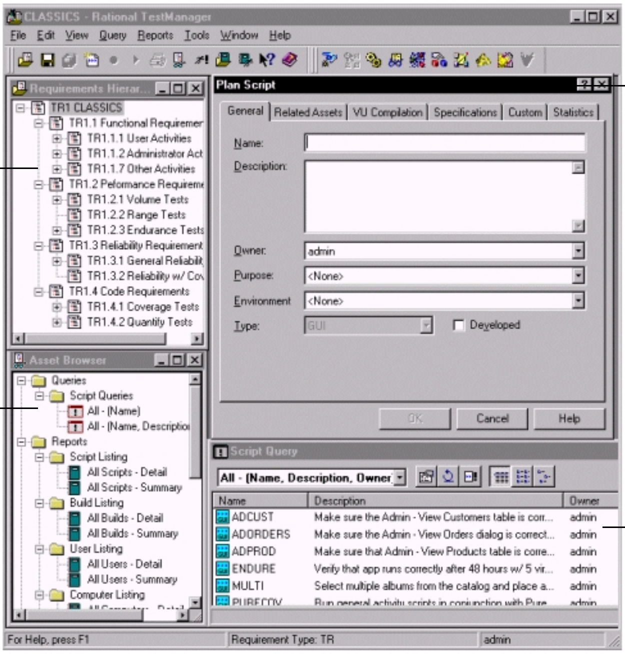
With Rational TestManager, you can do the following:

Create a test requirements hierarchy to represent the features and functionality that need to be tested in the application.

Create, manage, and display queries, reports, and builds.

Plan a new script or edit the properties of an existing script.

Display the results of a query.



Name:

Description:

Owner: admin

Purpose: <None>

Environment: <None>

Type: GUI

☐ Developed

All - (Name, Description, Owner

Name	Description	Owner
ADCUST	Make sure the Admin - View Customers table is corr...	admin
ADORDERS	Make sure the Admin - View Orders dialog is correct...	admin
ADPROD	Make sure that Admin - View Products table is corre...	admin
ENDURE	Verify that app runs correctly after 48 hours w/ 5 vir...	admin
MULTI	Select multiple albums from the catalog and place a...	admin
PURECOV	Run general activity scripts in conjunction with Pure...	admin

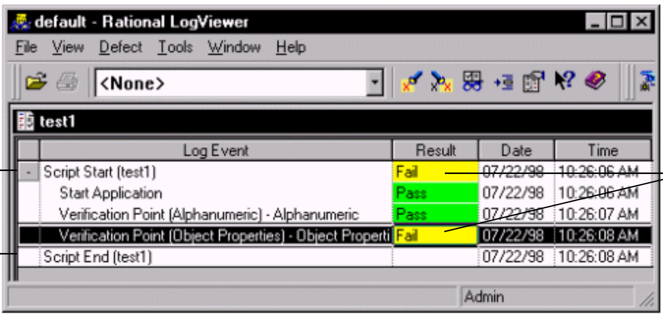
Requirement Type: TR | admin

Analyzing Results in the LogViewer and Comparators

The Rational LogViewer lets you view logs that are created when you play back scripts in Robot or run Pilots in TestFactory. (For information about TestFactory, see page 9.)

Reviewing the playback results in the LogViewer reveals whether each script and its components passed or failed.

Each script produces a log file that displays the results of each test within the script.

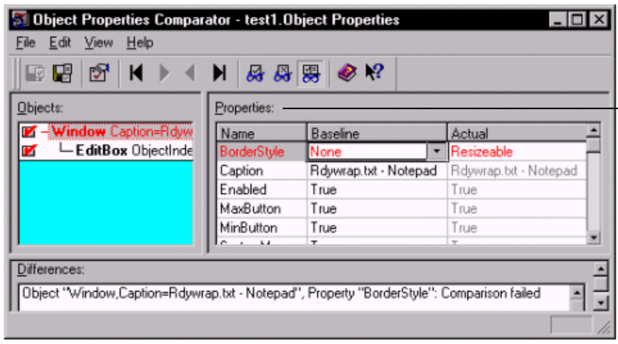

default - Rational LogViewer
File View Defect Tools Window Help
<None>
test1
Admin

Log Event	Result	Date	Time
Script Start (test1)	Fail	07/22/98	10:26:06 AM
Start Application	Pass	07/22/98	10:26:06 AM
Verification Point (Alphanumeric) - Alphanumeric	Pass	07/22/98	10:26:07 AM
Verification Point (Object Properties) - Object Properti	Fail	07/22/98	10:26:08 AM
Script End (test1)		07/22/98	10:26:08 AM

Since one of the verification points in the script failed, the entire script fails.

To analyze each failure and then remedy it, you can use one of the LogViewer's four Comparators—Grid, Object Properties, Text, and Image. Each Comparator graphically displays the 'before and after' results of playback.

If there is no failure on playback, only a **baseline file** displaying the recorded data or image is displayed. If a failure occurs on playback, an **actual file** is displayed next to the baseline file.



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

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

By comparing the baseline and actual files, you can determine if a failure is an intentional change or a defect.

Testing Applications with Rational TestFactory

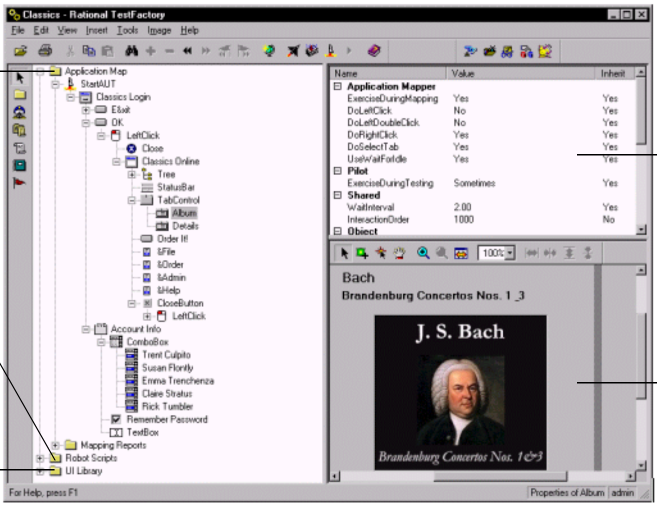
TestFactory provides a new level of automated testing as an integrated component of Rational Robot. You can use TestFactory to:

- Automatically create and maintain a detailed map of the application-under-test.
- Automatically generate scripts that provide extensive product coverage and scripts that encounter defects, without recording.
- Track executed and unexecuted source code, and report detailed findings.
- Shorten the product testing cycle by minimizing the time invested in writing navigation code.
- Automate distributed functional testing with TestAccelerator—an application that drives and manages the execution of scripts on remote machines.
- Play back Robot scripts in TestFactory to see code coverage and to create regression suites, and play back TestFactory scripts in Robot to debug them.

The application map lists each window and control in the application-under-test.

Scripts recorded in Robot can be used along with scripts created in TestFactory.

TestFactory can test the objects in the classes and subclasses known to occur in GUIs.



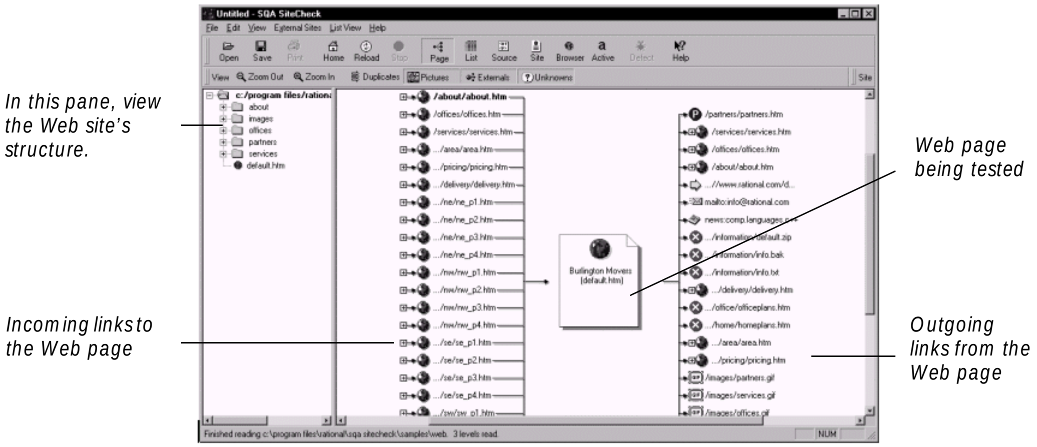
Name	Value	Inherit
Application Mapper		
ExerciseDuringMapping	Yes	Yes
DoLeftClick	No	Yes
DoLeftDoubleClick	No	Yes
DoRightClick	Yes	Yes
DoSelectTab	Yes	Yes
UseWaitForIdle	Yes	Yes
Pilot		
ExerciseDuringTesting	Sometimes	Yes
Shared		
WaitInterval	2.00	Yes
InteractionOrder	1000	No
Object		

In this pane, view information about whatever is selected in the left pane—usually properties for a selected UI object.

In this pane, view a bitmap of the selected object.

Managing Intranet and Web Sites with Rational SiteCheck

Rational SiteCheck lets you test the structural integrity of your intranet or World Wide Web site. It's designed to help you view, track, and maintain your rapidly changing site. You can test the content of your Web site after every revision to ensure that changes have not resulted in defects. And you can capture a baseline of your Web site and compare it to the Web site at another point in time.


In this pane, view the Web site's structure.
Incoming links to the Web page
Web page being tested
Outgoing links from the Web page
Untitled - SQA SiteCheck
File Edit View External Sites List View Help
Open Save Print Home Reload Stop Page List Source Site Browser Active Detect Help
View Zoom Out Zoom In Duplicates Pictures Externals Unknowns
c:/program files/rational
about
images
offices
partners
services
default.htm
/about/about.htm
Burlington Movers (default.htm)
NUM

Finished reading c:\program files\rational\sqa sitecheck\samples\web. 3 levels read.

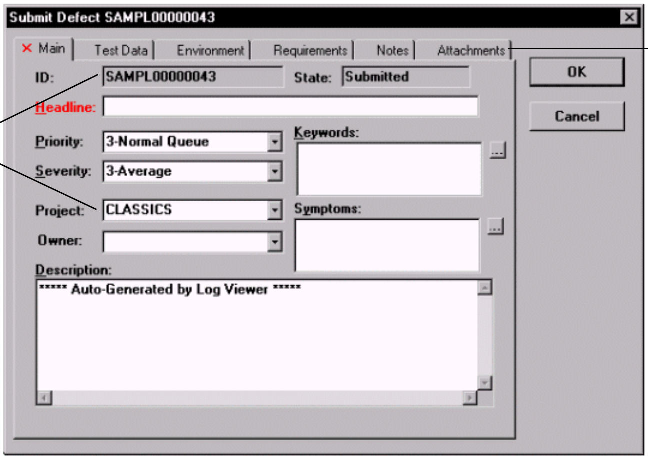
With SiteCheck you can:

- Visualize the structure of your Web site and display the relationship between each page and the rest of the site.
- Identify and analyze Web pages with active content, such as forms, Java, JavaScript, ActiveX, and Visual Basic Script (VBScript).
- Filter information so that you can inspect specific file types and defects, including broken links.
- Examine and edit the source code for any Web page, with color-coded text.
- Update and repair files using the integrated editor, or configure your favorite HTML editor to perform modifications to HTML files.
- Perform comprehensive testing of secure Web sites with Secure Socket Layer (SSL) support, proxy server configuration, and support for multiple password realms.

Managing Defects with Rational ClearQuest

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

Information automatically transferred to ClearQuest from the LogViewer.



ID: SAMPL00000043
State: Submitted
Headline:
Priority: 3-Normal Queue
Severity: 3-Average
Project: CLASSICS
Owner:
Description:
***** Auto-Generated by Log Viewer *****

Each tab provides important information about the defect.

With Robot and ClearQuest, you can:

- Submit defects directly from the LogViewer or SiteCheck.
- Generate defects directly from ClearQuest.
- Modify and track defects and change requests.
- Analyze project progress by running queries, charts, and reports.
- Automatically send email about a defect to its owner or to the project leader.

Collecting Diagnostic Information During Playback

As an application is developed and tested, you can use the diagnostic capabilities of Rational Purify, Rational Visual Quantify, and Rational Visual PureCoverage.

Rational Purify is a comprehensive C/C++ run-time error checking tool that automatically pinpoints run-time errors and memory leaks in all components of an application, including third-party libraries, ensuring that code is reliable.

Rational Visual Quantify is an advanced performance profiler that provides application performance analysis, enabling developers to quickly find, prioritize, and eliminate performance bottlenecks within an application.

Rational Visual PureCoverage is a customizable code coverage analysis tool that provides detailed application analysis and ensures that all code has been exercised, preventing untested code from reaching the end-user.

What’s Next

Now that you’ve been introduced to Rational Robot and its companion products and components, there’s still lots more to discover.

Here’s where you can find the information you need to get started with Rational Robot:

To find out about how to	Take a look at
Upgrade from SQA Suite 6.x to Rational Suite TestStudio, Rational TeamTest, or Rational Robot	The information in the next chapter, *Upgrade Information*.
Use each of the Rational Robot components	The tutorial beginning in the chapter *Learning Rational Robot – a Tutorial* on page 21.

Upgrade Information

This chapter explains how to upgrade from SQA Suite 6.x to the current Rational Test products. It includes the following topics:

- Summary of tasks
- New terminology
- Using defect reports
- Using SQA 6.x repository data in a Rational repository

If you're not upgrading, you should skip this chapter and take a look at the tutorial that begins on page 21.

Summary of Tasks

The following table lists tasks that you use to test an application, the component you used in SQA Suite 6.x for each task, and the component that you now use in Rational Test products to do the same task:

Task	**SQA Suite 6.x Component**	**Rational Test Component**
Create a repository.	SQA Administrator	Rational Administrator
Create test requirements.	SQA Manager	Rational TestManager
Create test scripts or procedures.	SQA Robot	Rational Robot
Check for pass/fail of test scripts or procedures.	SQA Test LogViewer	Rational LogViewer
Track defects.	SQA Manager	Rational ClearQuest
Run a report on a defect.	SQA Manager	Rational ClearQuest
Run a report on test coverage.	SQA Manager	Rational TestManager
Customize a report format.	SQA Manager	Rational ClearQuest

New Terminology

The following table lists the terminology found in SQA Suite 6.x and the new terminology in Rational Test products:

SQA Suite 6.x Term	Rational Test Term
distributed GUI test	distributed functional test
distributed regression test	distributed functional test
Insert toolbar	GUI Insert toolbar
log file	log
SQA Administrator	Rational Administrator
SQA Grid Comparator	Grid Comparator
SQA Image Comparator	Image Comparator
SQA Manager	Rational TestManager
SQA Object Properties Comparator	Object Properties Comparator
SQA OCX	Rational ActiveX Test Control
SQA Repository	Rational repository
SQA Robot	Rational Robot
SQA SiteCheck	Rational SiteCheck
SQA Test Log Viewer	Rational LogViewer
SQA Text Comparator	Text Comparator
test case	verification point
test log	log
test procedure	script
test schedule	schedule
test script	script

Scripting Languages

You no longer need to select a scripting language at installation. SQABasic is the scripting language used for recording GUI scripts, and it is installed automatically.

Using Defect Reports

Your Rational Suite TestStudio or Rational TeamTest software comes with special defect reports to help you manage your testing efforts. Use Rational ClearQuest to run these special reports to track your defects when testing applications. When you create a Rational repository that contains an associated ClearQuest database, you automatically get these defect reports and report formats to use with ClearQuest. For more information about creating a repository with a ClearQuest database, see the section about creating a repository in the chapter *Managing a Rational Repository* in the *Using the Rational Administrator* manual.

You can customize a defect report or create new reports by customizing a report format or by creating a new report format using Rational ClearQuest. To customize a report format, you must install Crystal Reports 6.0 Professional Edition, which comes with your Rational Suite TestStudio or Rational TeamTest software.

To install Crystal Reports:

▸ Use the Crystal Reports installation directions that come with the Crystal Reports CD-ROM.

Using SQA 6.x Repository Data in a Rational Repository

After installation, you can use data from an existing SQA Suite 6.x repository in a Rational repository. When you create a new Rational repository, the Create Repository wizard gives you the option of initializing a repository with data from an existing SQA Suite 6.x repository.

For information about creating a Rational repository with SQA Suite 6.x data, see the section about creating a repository in the chapter *Managing a Rational Repository* in the *Using the Rational Administrator* manual.

When to Convert SQA Suite 6.x Repository Data

We recommend that you convert SQA Suite 6.x repository data to Rational repository data after you finish a project.

For information about creating a Rational repository with SQA Suite 6.x data, see the section about creating a repository in the chapter *Managing a Rational Repository* in the *Using the Rational Administrator* manual.

The following table lists the type of data in an SQA Suite 6.x repository and which data converts to a Rational repository:

Type of Data	**Converts from SQA Suite 6.x data to Rational repository data**
Custom reports and graphs	
Defects (For more information about defects, see *Details About Converting Defects* on the next page.)	√
E-mail rules	
Filters for defects or test procedures	
Groups (Privileges are not converted.)	√
List reports	
LoadTest schedules	
Projects	√
Requirements	√
Test cases (called verification points in a Rational repository)	√
Test logs	
Test plans	√
Test procedures (called scripts in a Rational repository)	√
Users (Privileges are not converted.)	√
Virtual user test procedures (called virtual user scripts in a Rational repository)	

Conversion Details

When you create a Rational repository with SQA Suite 6.x data, the following conversion takes place:

- A script file name or verification point ID with a period (.) in the name is prefixed with _RENAMED_.
- A period (.) in a script file name or verification point ID converts to an underscore (_).

For example, a script named wn95.tst converts to _RENAMED_wn95_tst. A verification point ID named alpha.b2 converts to _RENAMED_alpha_b2.

Details About Converting Defects

The Rational TeamTest and Rational Suite TestStudio products include a special version of ClearQuest to track your defects. For your convenience, these products also include a specially designed defect form, the TeamTest defect form, similar to the SQA 6.x Manager defect form. For more information about using the TeamTest defect form, see the Using the Rational Robot manual.

The following table lists each field of an SQA 6.x defect and the comparable Rational TeamTest defect field. The table also describes how certain fields are converted.

The word “List” in the Description column of this table indicates that if you customize a list in SQA 6.x, it converts to a list in the TeamTest defect form.

SQA 6.x Defect Field	TeamTest Defect Field	Conversion Description
ID	New ID generated	The SQA 6.x ID does not appear on the defect form, but you can use the query feature of ClearQuest to find it. For more information, see the Rational ClearQuest Help.
Description	Headline/ Description	Converts the first 125 characters or up to the first carriage return and places this text in the Headline field. The entire SQA 6.x Description field also appears in the Description field of the TeamTest defect form.
Priority	Priority	List
Severity	Severity	List
Occurrences	(not converted)	
Keywords	Keywords	List
Symptoms	Symptoms	List

Build Found	Notes	In SQA 6.x, a Build Found field is a text field. The text data from the Build Found field is stored in Notes. In Rational 7.x, Builds are objects that contain log folders and logs. A Build object is not created for each SQA 6.x Build Found field.
Build Fixed	Notes	In SQA 6.x, a Build Fixed field is a text field. The text data in the Build Fixed field is stored in Notes. In Rational 7.x, Builds are objects that contain log folders and logs. A Build object is not created for each SQA 6.x Build Fixed field.
Proc	Script	
Case	Verification Point	
Cycle	(not converted)	
Reported By	Reported By Contact	
Reported By Company	Reported By Company	
Hardware	Hardware	List
Operating System	Operating System	List
Other	Other Environment	
Test Station	(not converted)	In SQA 6.x, a Test Station field is a text field. In Rational 7.x, the Computers field replaces the Test Station field and is an object. A Computer object is not created for each SQA 6.x Test Station field.
Log	(not converted)	
Custom 1	Custom 1	List If you customized the label for this field in SQA 6.x, the label is not converted. Use the ClearQuest Designer to customize this label.

Custom 2	Custom 2	List If you customized the label for this field in SQA 6.x, the label is not converted. Use the ClearQuest Designer to customize this label.
Custom 3	Custom 3	If you customized the label for this field in SQA 6.x, the label is not converted. Use the ClearQuest Designer to customize this label.
Attachment	Attachments	You can attach more than one file after you convert to a 7.x Rational repository.
Requirement	Requirement	
Status History	History	
Resolution	Resolution	List
Resolution Description	Resolution Note	
Modified Software	(not converted)	

Troubleshooting

The convert.txt file in the converted project directory contains the conversion status of each verification point and script. Use this file as a diagnostic tool if you have any problems with your data after conversion.

Learning Rational Robot - a Tutorial

This tutorial takes you through a typical software testing cycle in which you'll use the functional testing capabilities of Rational Robot to test a sample client/server application.

As you follow the examples in this tutorial, you'll quickly realize how easy it is to use Rational Robot to identify changes or unintentional errors in an application. You'll understand how Robot lets you cover all phases of functional testing and how to best use Robot for your own testing projects.

What Is Automated Functional Testing?

Automated testing uses software tools to replace repetitive and often error-prone manual testing. Automated testing saves time and enables a reliable, predictable, and accurate testing process.

With automated functional testing, you validate the behavior of an application. Functional testing lets you exercise the application's GUI to verify that the application responds appropriately to user and system input, and that it conforms to project requirements.

Functional testing focuses on:

- The ability of the application to address requirements—for example, verifying that the application logic functions as designed.
- The appearance and state of GUI objects for example, verifying that changes to the GUI are correct from one build to another.
- The operation of the application—for example, verifying that an application accurately processes server requests and responses.

About the Sample Application

The sample application that you'll use with this tutorial is called Classics Online, developed using Microsoft Visual Basic.

The intent of the Classics Online application is to allow customers to browse through an online catalog and place orders. When the Classics Online application is completed, tested, and successfully deployed, it will provide automated order- entry and fulfillment capabilities for customers.

About the Tutorial Examples

This tutorial contains seven examples. Each is based on a scenario in which you're an employee of Classics, Inc., working in the Quality Assurance (QA) department. Your department is responsible for testing the new application—making sure defects are discovered and fixed before the application is deployed.

As you go through the examples in this tutorial, you'll become familiar with the six general phases that occur in most testing projects and the Rational Test products used in each phase:

For this testing phase	Use this Rational Test product
Test Planning—Define requirements, and plan and manage your test assets.	Rational TestManager *See Example 1.*
Test Development—Record, verify, and enhance your scripts.	Rational Robot *See Examples 2, 3, and 7.*
Test Execution—Play back the scripts against different builds of your application.	Rational Robot *See Example 4.*
Test Results—Review and analyze the results of the tests.	Rational LogViewer and Comparator *See Examples 3 and 4.*
Defect Tracking—Specify defects and assign them to the appropriate person for resolution.	Rational ClearQuest *See Example 5.*
Summary Reporting and Analysis— Run reports to determine test coverage and the state of defects.	Rational TestManager and ClearQuest *See Example 6.*

Where to Find Other Useful Testing Tips

In addition to this tutorial, you can learn about Rational Robot by using the testing tips in the *Rational Robot Try it!* cards. Each card shows you—in a minimal amount of steps—how to use Robot to test the controls specific to one of the following development environments: Visual Basic, Oracle, PowerBuilder, Java, and HTML.

When you use the testing tips in a *Try it!* card on the corresponding sample applet, you'll discover, in just minutes, how to record tests that verify the controls specific to the environment you select.

Preparing to Start the Tutorial

Before You Begin

Before you can test the Classics Online sample application, you need to complete these tasks:

- Install Rational Robot.
- Install the Classics Online sample application.
- Connect to a sample Rational repository.

Each task is described in the sections that follow.

Installing Rational Robot

Rational Robot is available in three different Rational product packages. To install your version of Rational Robot, see one of the following manuals:

To install Rational Robot as part of	Take a look at
Rational Robot package	*Installing Rational TeamTest and Rational Robot*
Rational TeamTest package	*Installing Rational TeamTest and Rational Robot*
Rational Suite TestStudio package	*Installing Rational Suite*

Installing the Sample Application

After you've installed Rational Robot, you need to install the Classics Online sample application.

1. Click **Start** → **Programs** → *Rational program group* → **Rational Test** → **Setup Rational Test Samples**.
2. Select **Classics Online**.

NOTE: If you want to use the testing tips in the *Rational Robot Try it!* cards, select some or all of the other samples.

3. Click **Next**, and then click **Finish**.

The sample application appears on the Windows Start menu under **Programs** → **Rational Test Samples**.

Connecting to the Sample Repository

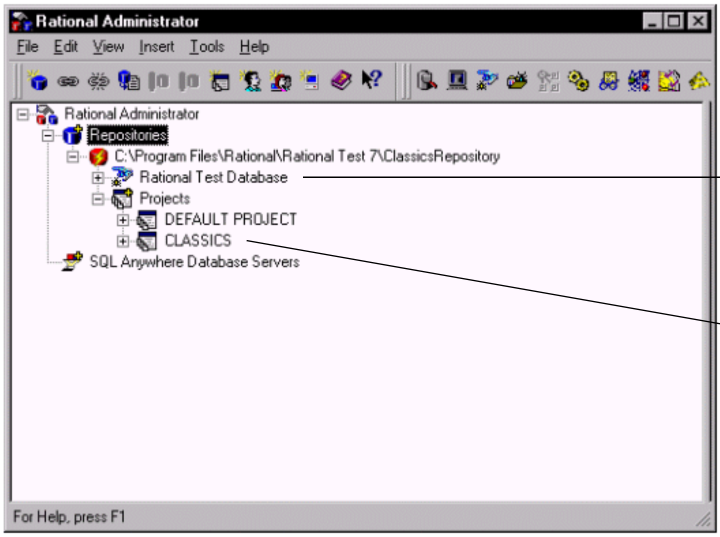
When you use a Rational Test application, you need to connect to a Rational repository. The **Rational repository** stores information about your software testing and development efforts. When you test an application, you can create as many repositories as needed. And you can use the repository as a single user, or you can share it as a member of a team.

For this tutorial, we've provided you with a sample Rational repository, installed when you installed the sample application.

To connect to the sample repository:

1. Click **Start** → **Programs** → **Rational program group** → **Rational Test** → **Rational Administrator**.
2. Click **File** → **Connect**.
3. Select the **ClassicsRepository** path and click **OK**.

4. Accept **admin** as the user ID, with no password, and click **OK**.



The database stores your test assets.

A project helps you organize your testing information and resources for easy tracking.

5. Minimize the Rational Administrator window.

What's Next

You're ready to begin the tutorial by following the examples in this guide.

Test Planning

Objectives

- Define two test requirements that identify the functionality to test.
- Add the test requirements to the requirements hierarchy.
- Define a script to verify the requirements.
- Associate the script with the test requirements.

Scenario

To help you with your test planning activities, you'll use Rational TestManager. Like other Rational Test products, TestManager accesses a Rational repository. The repository maintains **test assets**—all of the information about your testing project.

For this tutorial, we assume that you've already created a test plan. The **test plan** describes the features and functionality that you're going to test and how you're going to test them. Often, the test plan describes resource requirements and defines schedules.

As part of the test planning phase for this tutorial, you'll define **test requirements**—the features and functionality that you plan to test in the application.

You'll use TestManager to build a **test requirements hierarchy**. Each level in the hierarchy will represent a specific functional characteristic or feature that's found in the application or expected to be in the application. By defining these test requirements before you begin to test, you'll have a good idea about what you have to test.

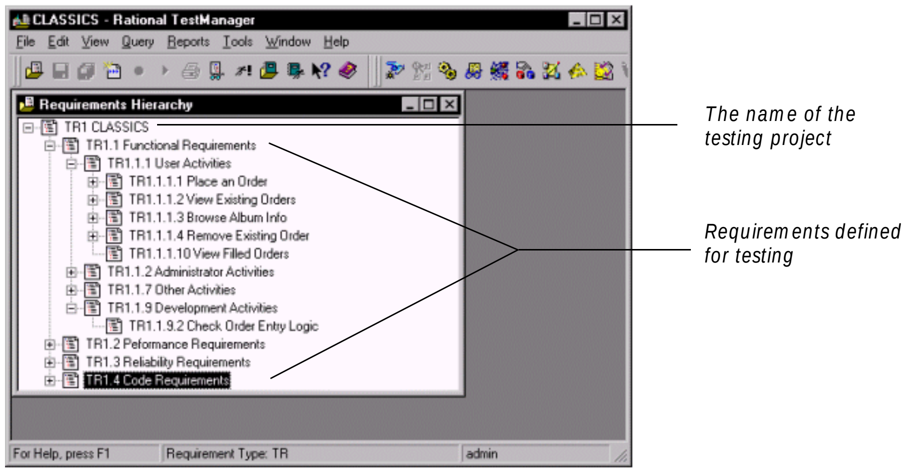
After you've defined the test requirements, you'll plan a script that you can use to verify each requirement that you've created.

Defining Test Requirements

As part of the test planning phase, you need to define the requirements for your testing project. The requirements will help you ensure that you provide full test coverage for your application.

Since testing is usually an iterative process that tests one level of the application, then the next, and then goes back to repeat the process, TestManager lets you create a test requirements hierarchy to easily define and display your requirements.

A test requirements hierarchy is a graphical tree structure. The root is the name of the testing project—for example, **Classics**. The branches define the test requirements for each phase of the project.


CLASSICS - Rational TestManager
File Edit View Query Reports Tools Window Help
Requirements Hierarchy
For Help, press F1
Requirement Type: TR
admin
The name of the testing project
Requirements defined for testing

```
TR1 CLASSICS
  TR1.1 Functional Requirements
    TR1.1.1 User Activities
      TR1.1.1.1 Place an Order
      TR1.1.1.2 View Existing Orders
      TR1.1.1.3 Browse Album Info
      TR1.1.1.4 Remove Existing Order
      TR1.1.1.10 View Filled Orders
    TR1.1.2 Administrator Activities
    TR1.1.7 Other Activities
    TR1.1.9 Development Activities
      TR1.1.9.2 Check Order Entry Logic
  TR1.2 Peformance Requirements
  TR1.3 Reliability Requirements
  TR1.4 Code Requirements
```

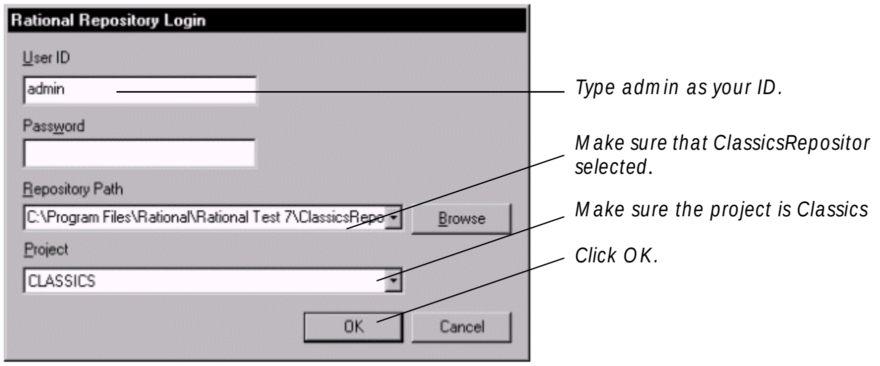
Requirements in TestManager are stored in a Rational RequisitePro database. RequisitePro is a requirements management tool that helps project teams control the development process by organizing, managing, and tracking the changing requirements of their testing project.

Defining and Inserting a Requirement in the Hierarchy

As you define and insert requirements, you can view the evolving hierarchy in TestManager. This ensures that the requirements that appear in the hierarchy reflect the functionality you want to test. You can add requirements to the hierarchy at any time in the testing cycle, and you can delete them at any time.

To define and insert a test requirement:

1. Click **Start** → **Programs** → **Rational product name** → **Rational TestManager**.



Rational Repository Login

User ID: admin

Password:

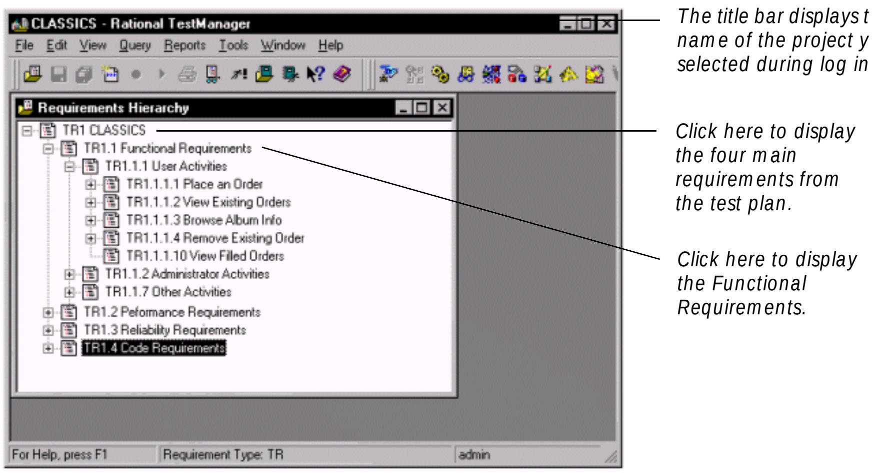
Repository Path: C:\Program Files\Rational\Rational Test 7\ClassicsRepo

Project: CLASSICS

Browse
OK
Cancel
Type admin as your ID.
Make sure that ClassicsRepositor selected.
Make sure the project is Classics
Click OK.

2. Click **File** → **Open Requirements**.

   Because you're connected to the sample repository, **ClassicsRepository** and to the project **Classics**, the requirements hierarchy for that project appears.



CLASSICS - Rational TestManager

Requirements Hierarchy

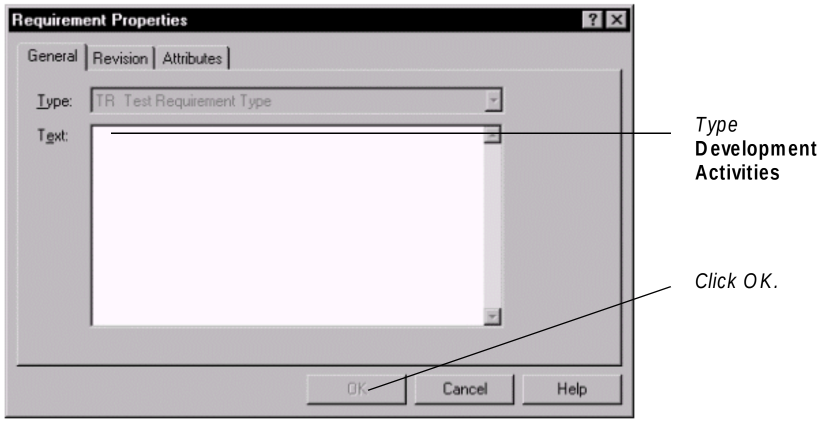
- TR1 CLASSICS
  - TR1.1 Functional Requirements
    - TR1.1.1 User Activities
      - TR1.1.1.1 Place an Order
      - TR1.1.1.2 View Existing Orders
      - TR1.1.1.3 Browse Album Info
      - TR1.1.1.4 Remove Existing Order
      - TR1.1.1.10 View Filled Orders
    - TR1.1.2 Administrator Activities
    - TR1.1.7 Other Activities
  - TR1.2 Peformance Requirements
  - TR1.3 Reliability Requirements
  - TR1.4 Code Requirements

For Help, press F1 | Requirement Type: TR | admin

File
Edit
View
Query
Reports
Tools
Window
Help
The title bar displays t name of the project y selected during log in
Click here to display the four main requirements from the test plan.
Click here to display the Functional Requirements.

3. To insert a child requirement at the same level as User Activities, select **User Activities**.

4. Click **Edit** → **Insert Requirement**.



Requirement Properties

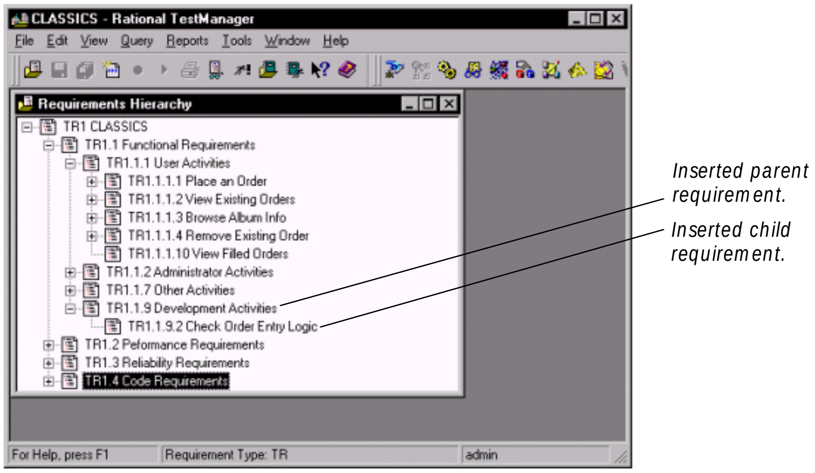
General | Revision | Attributes

Type: TR Test Requirement Type

Text:

OK | Cancel | Help

5. To insert a child requirement under **Development Activities**, select it.
6. Click **Edit** → **Insert Child Requirement**.
7. Type **Check Order Entry Logic** and click **OK**.



CLASSICS - Rational TestManager

File Edit View Query Reports Tools Window Help

Requirements Hierarchy

```
TR1 CLASSICS
  TR1.1 Functional Requirements
    TR1.1.1 User Activities
      TR1.1.1.1 Place an Order
      TR1.1.1.2 View Existing Orders
      TR1.1.1.3 Browse Album Info
      TR1.1.1.4 Remove Existing Order
      TR1.1.1.10 View Filled Orders
    TR1.1.2 Administrator Activities
    TR1.1.7 Other Activities
    TR1.1.9 Development Activities
      TR1.1.9.2 Check Order Entry Logic
  TR1.2 Peformance Requirements
  TR1.3 Reliability Requirements
  TR1.4 Code Requirements
```

For Help, press F1 | Requirement Type: TR | admin

You've just inserted a requirement. Later on, a member of the development team will check the order-entry logic as part of the testing process for the Classics Online application.

Defining a Script

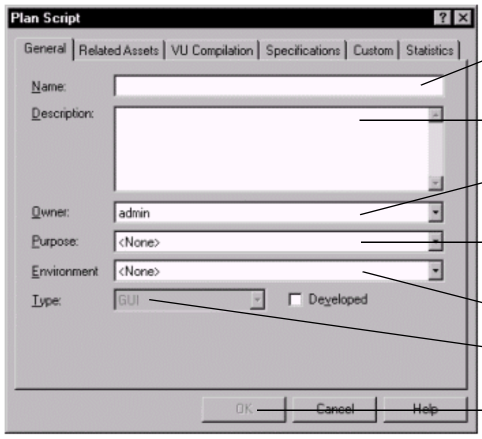
The next step in the planning process is to define scripts that you'll use to validate your test requirements. A **script** is a specific sequence of actions and tests recorded against the application-under-test. When you define a script, it becomes a place-holder for the script that you'll actually record later using Rational Robot.

If you want, you can skip this test planning phase and simply start recording with Robot. But the preferred usage model for any testing activity is to plan first. This lets you know at the beginning of a project the size of the testing effort. You'll also be able to assign the development of each script to the appropriate person. And, you'll be able to develop logical and consistent naming conventions for all of your scripts.

Defining a Script and Attaching It to a Requirement

To define a script and attach it to a requirements:

1. Make sure that the test requirements hierarchy for the Classics project is open and the **User Activities** branch expanded.
2. Click **Place an Order** and expand it to view the requirements under it.
3. Right-click the requirement **Single Item - Place an order for a single album**.
4. Click **Plan Script**.
5. Fill in the fields of the **General** tab.



Name the script SINGLE ORDER.

Type Make sure album information is retrieved correctly from database.

Select BRABKIN.

Select Functional as the test type.

Select your Windows environment.

Select Functional as the test type.

Click OK.

The script **SINGLE ORDER** appears under the requirement **Single Item** in the hierarchy. **SINGLE ORDER** is the placeholder for the script that you'll record later to test the **Single Item** requirement.

6. To confirm that the script is attached to the requirement, select **SINGLE ORDER** and right-click. Since one of the options is **Detach**, you can assume that the script is attached to the requirement.

Summary

You've just had a quick introduction to some of the test planning functionality in TestManager. And, you've started to plan your test suite by using TestManager to:

- Define test requirements and insert them in the Requirements hierarchy.
- Define a script for recording later in Rational Robot.
- Attach the script to a specific test requirement to track the progress of your testing efforts.

What's Next

You're now ready to begin recording with Rational Robot.

Recording a Script

Objectives

- Record a script that validates specific test requirements.
- Test the album data displayed in the tree control.
- Test the text property of a pushbutton.
- Test the data retrieved from the order-entry database.
- Test the properties of a hidden data control.

Scenario

In the last example, you used Rational TestManager to develop a test strategy. By planning ahead, you've already defined a script called SINGLE ORDER that you can use in this example.

When you record scripts for an early build of an application, you're investing in future testing efforts. As soon as the next build becomes available, you can play back the scripts you recorded and test the new build in a fraction of the time it would take to perform the tests manually.

Recording a Script

The script that you'll record is the SINGLE ORDER script that you defined in TestManager.

When you record a script, Robot uses its **Object-Oriented Recording** technology to record both the actions as you navigate through the application-under-test and the tests that you insert to verify controls and data in the application.

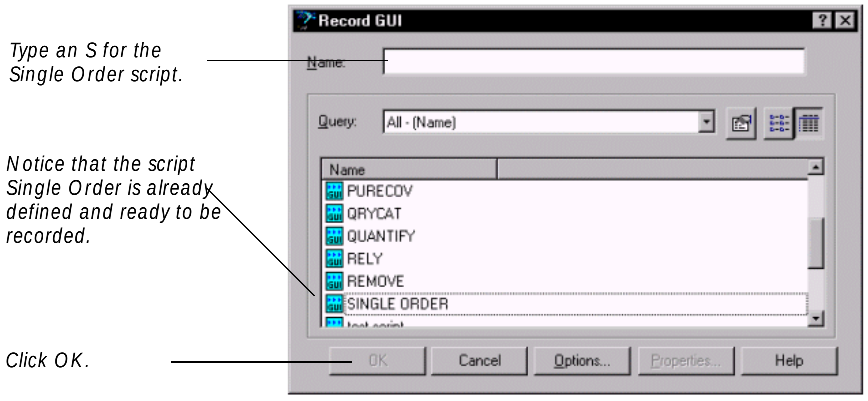
NOTE: If you were testing your own Visual Basic, Oracle Forms, Java, or any other application containing ActiveX controls, you would install one of the Rational Test Enablers before you began to test. Each enabler is non-intrusive and allows Rational Robot to thoroughly test the controls specific to your application's development environment. For more information about the Rational Test Enablers, see the *Using Rational Robot* manual.

Navigating Through the Application

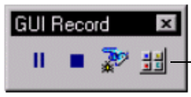
To start recording the script SINGLE ORDER:

1. Before you begin to record, minimize all open windows except TestManager.
2. Start Robot by clicking the **Rational Robot** button on the TestManager toolbar.

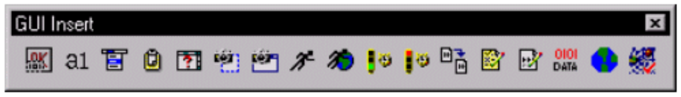
   Notice how easy it is to switch from one Rational Test product to another. Since you've already logged in to the Rational repository from TestManager, you can access Robot and other Rational Test products without logging in again.
3. Minimize TestManager.
4. From Robot, choose **File** → **Record GUI**.



The **GUI Record** toolbar appears in the upper-left corner after Robot is minimized. You're about to begin recording by inserting different testing capabilities into the script. To begin:



Click the Display GUI Insert Toolbar button to display the GUI Insert toolbar.



Point to (don't click) each button to view its ToolTip.

5. Click the **Start Application** button.

> **NOTE:** Starting the application-under-test while you're recording lets you start an application without using the Start menu or the Windows desktop. This gives you more precise control over your test environment and ensures that during playback, testing begins with the application in the same state that it was in during recording. It also lets you run tests unattended.

6. Click **Browse** and select the following default path for Build A of the Classics Online sample application:

   C:\Program Files\Rational\Rational Test 7\Sample Applications\ClassicsOnline\ClassicsA.exe

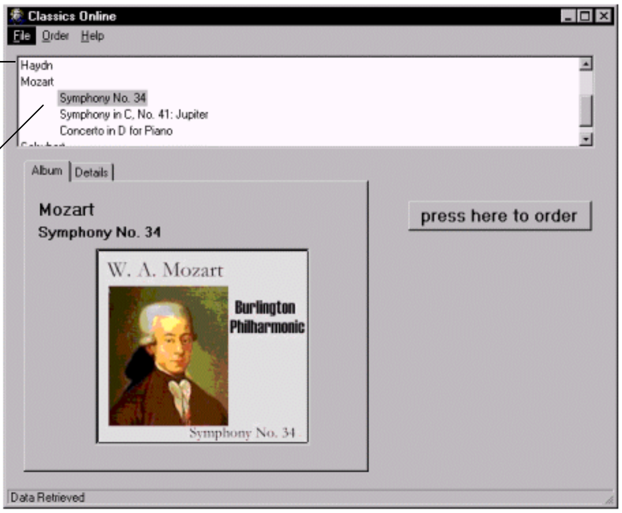
7. Click **Open**, and then click **OK**.

8. Click **OK** to log in as Trent Culpito and to open the Classics Online application.

9. Scroll down until you see **Mozart**.

ouble-click **Mozart** *to isplay the albums you an order.*

Select **Symphony No. 34.**



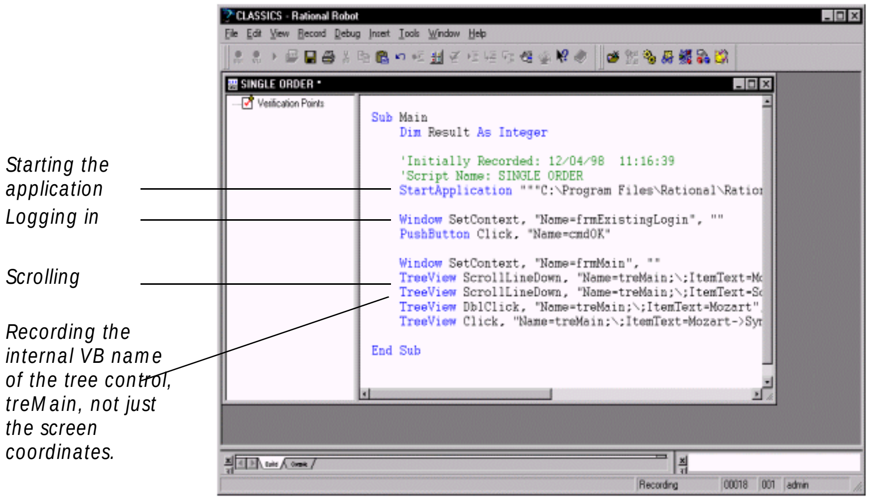
NOTE: It's important to know that as you follow this tutorial, you can make mistakes—add or repeat steps—without interfering with the script you're recording. As long as you don't close the Classics application before the script is complete or open different application windows, you should be all set.

Viewing the Beginnings of the Script

You've just navigated through the sample application and clicked a few objects. Robot captured each of your user actions in the SINGLE ORDER script. You'll continue to do some more recording, but now's a good time to take a look at the developing script.

To view the script:

1. On the **GUI Record** toolbar, click the **Open Robot Window** button. None of these actions are recorded in the script.
2. Maximize the script to view the actions that you've taken so far.


Starting the application
Logging in
Scrolling
Recording the internal VB name of the tree control, treMain, not just the screen coordinates.
CLASSICS - Rational Robot
File Edit View Record Debug Insert Tools Window Help
SINGLE ORDER *
Verification Points
Recording
00018
001
admin

```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 12/04/98  11:16:39
    'Script Name: SINGLE ORDER
    StartApplication """C:\Program Files\Rational\Ratio

    Window SetContext, "Name=frmExistingLogin", ""
    PushButton Click, "Name=cmdOK"

    Window SetContext, "Name=frmMain", ""
    TreeView ScrollLineDown, "Name=treMain;\;ItemText=Mo
    TreeView ScrollLineDown, "Name=treMain;\;ItemText=Sc
    TreeView DblClick, "Name=treMain;\;ItemText=Mozart"
    TreeView Click, "Name=treMain;\;ItemText=Mozart->Sy

End Sub
```

3. After you've looked at the recorded actions in the script, minimize the Robot window.

You're now ready to begin to insert verification points in the script to test specific parts of the Classics Online application.

Adding Verification Points

During recording, you can insert one or more **verification points** in a script to capture and store information about the objects that you're testing. The information becomes the baseline for your testing. You can then play back the script against future builds to check the progress of your application. If there are any changes, you'll be able to compare the baseline to the actual and decide which changes are enhancements and which are defects.

Robot provides various types of verification points. Two of the most powerful—**Object Properties** and **Object Data**—support Robot's in-depth Object Testing technology.

Object Testing inspects and verifies all the properties and data in the visible and non-visible objects in an application. Object Testing records objects by their internal object names, not just by the text that appears externally on the object nor by the object's location in a window or dialog box.

Capturing Data in a Tree Control – Verification Point 1

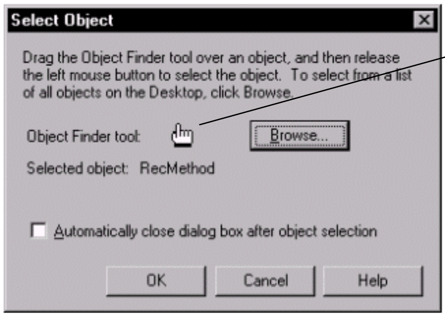
The first verification point that you'll insert into the single order script is an Object Data verification point. This type of test captures and verifies data inside standard Windows objects. It also provides specialized support for environment-specific objects such as Visual Basic data controls, ActiveX controls, Oracle Forms base-table blocks, and PowerBuilder DataWindows.

NOTE:

To create the first verification point:

1. If necessary, click the **Display GUI Insert Toolbar** button.
2. If necessary, minimize the TestManager window.
3. Click the **Object Data** button on the right side of the **GUI Insert** toolbar. ICON

   Robot automatically names the verification point with its autonaming feature. **Autonaming** automatically inserts a verification point name for you. You could use your own naming conventions, but for this tutorial you'll use the autonaming feature.
4. Click **OK** to accept **Object Data** as the name of the first verification point.


Select Object

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

Object Finder tool:

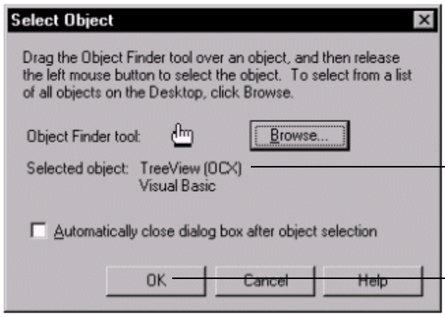
Selected object: RecMethod

Automatically close dialog box after object selection

Drag the Object Finder tool around the controls in the Classics Online window. Notice that it identifies each control.

Press the SHIFT key as you drag the tool to see the name Robot uses to recognize each control (Name=).

5. Drag the **Object Finder** tool to the list of composers and albums at the top of the application and release the mouse button.



Select Object

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

Object Finder tool:

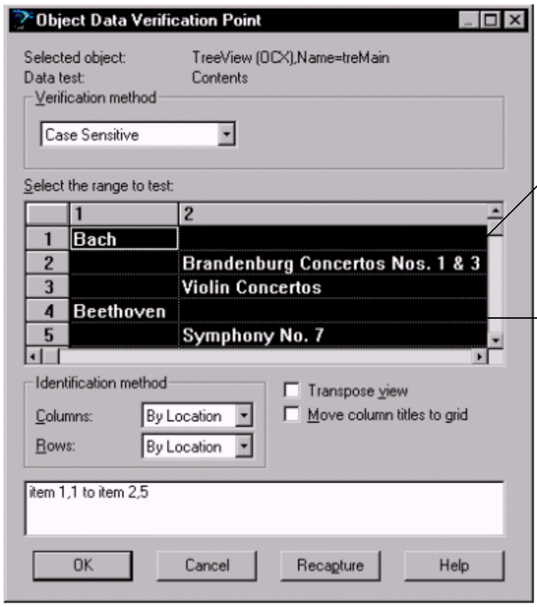
Selected object: TreeView (OCX) Visual Basic

☐ Automatically close dialog box after object selection

The selected object should be TreeView (OCX) Visual Basic. If it isn't, repeat step 5.

Click OK.

6. Scroll through the captured data in the Object Data Verification Point dialog box to see what Robot captured.



Object Data Verification Point

Selected object: TreeView (OCX),Name=treMain

Data test: Contents

Verification method: Case Sensitive

Select the range to test:

	1	2
1	Bach	
2		Brandenburg Concertos Nos. 1 & 3
3		Violin Concertos
4	Beethoven	
5		Symphony No. 7

Identification method

Columns: By Location

Rows: By Location

☐ Transpose view

☐ Move column titles to grid

item 1,1 to item 2,5

Robot captured all of the data in the tree control—even in branches that are not expanded.

You can select and deselect the cells. Robot tests only the cells that are selected before you continue.

Make sure all the cells are selected.

7. Click **OK** to complete the verification point.

In the next sections, you'll continue recording the single order script by adding two Object Properties verification points and another Object Data verification point.

Capturing the Properties of a Pushbutton – Verification Point 2

The next verification point that you'll insert into the single order script is an Object Properties verification point. Capturing and testing an object's properties is an extremely important capability in real-world testing situations.

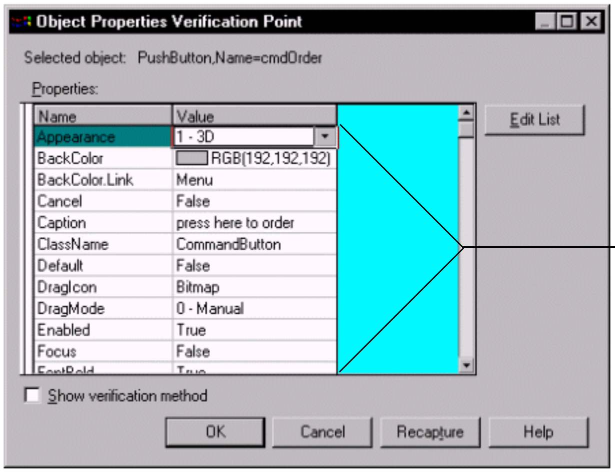
An **object's properties** describe its appearance (width, height, color, etc.), state (enabled, disabled, etc.), behavior, and non-visible properties (SQL statements, computed fields, etc.).

In the sample application, you'll record a verification point to verify the text on the pushbutton **press here to order**. In the next build of Classics (Build B), the text on this button will change, and Robot will catch this change. At that point, you'll decide whether it's a defect or an intentional change.

To create the second verification point:

1. Click the **Display GUI Insert Toolbar** button on the **GUI Record** toolbar.
2. Click the **Object Properties** button.
3. Accept the autonamed verification point and click **OK**.
4. Drag the Object Finder tool and release over the **press here to order** button.
5. Make sure that the selected object is **PushButton Visual Basic** and click **OK**.

   Robot captures all of the button's properties, but you'll edit the captured properties so that Robot tests only the text on the button.


Object Properties Verification Point

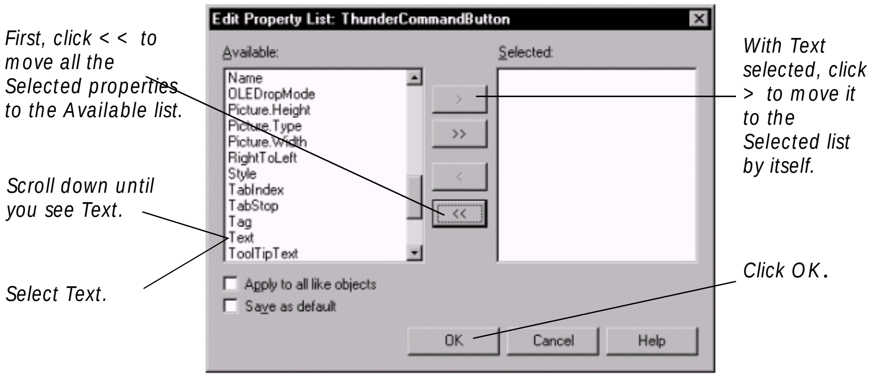
Selected object: PushButton,Name=cmdOrder

Properties:

Name	Value
Appearance	1 - 3D
BackColor	RGB(192,192,192)
BackColor.Link	Menu
Cancel	False
Caption	press here to order
ClassName	CommandButton
Default	False
DragIcon	Bitmap
DragMode	0 - Manual
Enabled	True
Focus	False

Robot captures all of the button's properties.

6. Click **Edit List**.


First, click << to move all the Selected properties to the Available list.
Scroll down until you see Text.
Select Text.
With Text selected, click > to move it to the Selected list by itself.
Click OK.
Edit Property List: ThunderCommandButton
Available:
Selected:
Name
OLEDropMode
Picture.Height
Picture.Type
Picture.Width
RightToLeft
Style
TabIndex
TabStop
Tag
Text
ToolTipText
Apply to all like objects
Save as default
OK
Cancel
Help

7. Click **OK** again to complete the verification point.

Capturing Data in a Databound Control - Verification Point 3

Now you're ready to make sure that Classics Online is adding new orders correctly to the order-entry database and retrieving the correct information from the database.

The object that you're going to test is a databound control.

NOTE: Throughout this tutorial, ActiveX and OCX are used interchangeably.

A **databound ActiveX** consists of two objects:

- The visible ActiveX control that displays the retrieved data on screen.
- A non-visible **data control** bound to the ActiveX. The data control retrieves the data from the database when an SQL (software query language) call is made from the application.

To capture the visible data displayed on screen in the ActiveX:

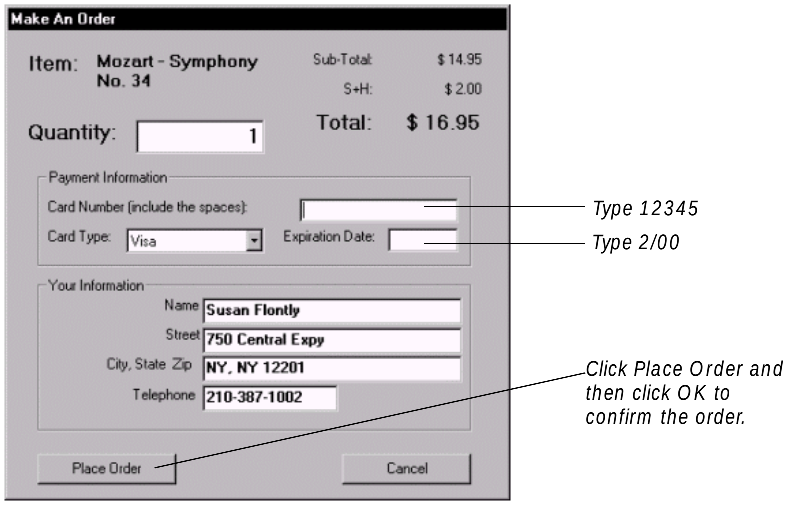
1. Log in as a new customer by selecting **File** → **Login As New User**.

2. Select **Susan Flontly** and click **OK**.

3. To view Susan's existing orders, select **Order** → **View Existing Order Status**. Notice that Susan has placed one order.

4. Click **Close**.

5. With **Mozart's Symphony No. 34** selected, click **press here to order**.



Make An Order

Item: Mozart - Symphony No. 34

Sub-Total: $ 14.95

S+H: $ 2.00

Total: $ 16.95

Quantity: 1

Payment Information

Card Number (include the spaces):

Card Type: Visa

Expiration Date:

Your Information

Name: Susan Flontly

Street: 750 Central Expy

City, State Zip: NY, NY 12201

Telephone: 210-387-1002

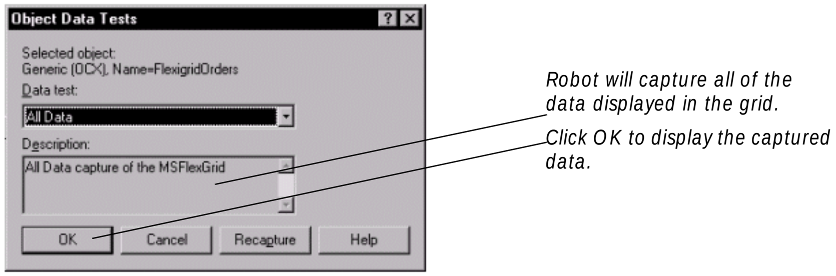
6. Select **Order** → **View Existing Order Status** and notice that a new order has been entered for Susan.

7. Keep the dialog box open and begin an Object Data verification point.

8. Accept the autonamed verification point (**ObjectData2**) and click **OK**.

9. Drag the Object Finder to the grid and release the mouse button.

10. Make sure that the selected object is **Generic (OCX) Visual Basic** and click **OK**.



Object Data Tests

Selected object:
Generic (OCX), Name=FlexigridOrders

Data test: All Data

Description:
All Data capture of the MSFlexGrid

OK
Cancel
Recapture
Help
Robot will capture all of the data displayed in the grid.
Click OK to display the captured data.

11. Click **OK** again to complete the verification point.

Capturing Properties of a Non-Visible Data Control - Verification Point 4

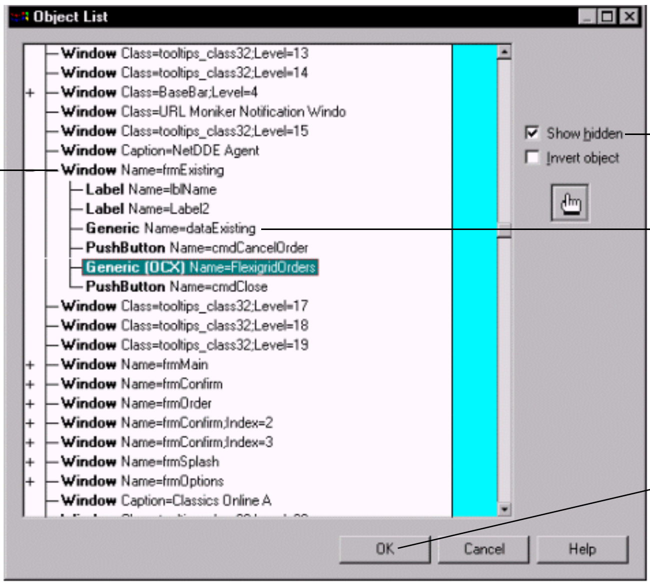
With this verification point, you'll test the application even more deeply than you did when you verified the visible data retrieved from the database. You'll capture the properties of the non-visible data control that makes SQL calls to the database to retrieve and update information.

To capture the properties of a non-visible object:

Move the pointing-hand over the objects in the window. Notice how SQA Robot displays the object type in the TestTip that appears under the pointing hand.

1. Make sure that the **View Existing Orders** window is open.
2. Begin an Object Properties verification point.
3. Accept the autonamed verification point and click **OK**. Instead of dragging the Object Finder to the grid, click the **Browse** button. This tells Robot to find and list all of the visible and non-visible objects on the Windows desktop.

NOTE: If necessary, you can resize the Object List dialog box to make it larger.



Object List

```
 ├─Window Class=tooltips_class32;Level=13
 ├─Window Class=tooltips_class32;Level=14
+├─Window Class=BaseBar;Level=4
 ├─Window Class=URL Moniker Notification Windo
 ├─Window Class=tooltips_class32;Level=15
 ├─Window Caption=NetDDE Agent
 ├─Window Name=frmExisting
 │  ├─Label Name=lblName
 │  ├─Label Name=Label2
 │  ├─Generic Name=dataExisting
 │  ├─PushButton Name=cmdCancelOrder
 │  ├─Generic (OCX) Name=FlexigridOrders
 │  └─PushButton Name=cmdClose
 ├─Window Class=tooltips_class32;Level=17
 ├─Window Class=tooltips_class32;Level=18
 ├─Window Class=tooltips_class32;Level=19
+├─Window Name=frmMain
+├─Window Name=frmConfirm
+├─Window Name=frmOrder
+├─Window Name=frmConfirm;Index=2
+├─Window Name=frmConfirm;Index=3
+├─Window Name=frmSplash
+├─Window Name=frmOptions
 ├─Window Caption=Classics Online A
```

f necessary, double-lick Window ame= frmExisting to xpand it. Each ranch represents an bject in the View xisting Orders dialog ox.

Clear, and then select this to see all of the hidden controls that Robot captured.

With Show hidden checked, select Generic Name= dataExisting—a non-visible data control that interacts with the order-entry database.

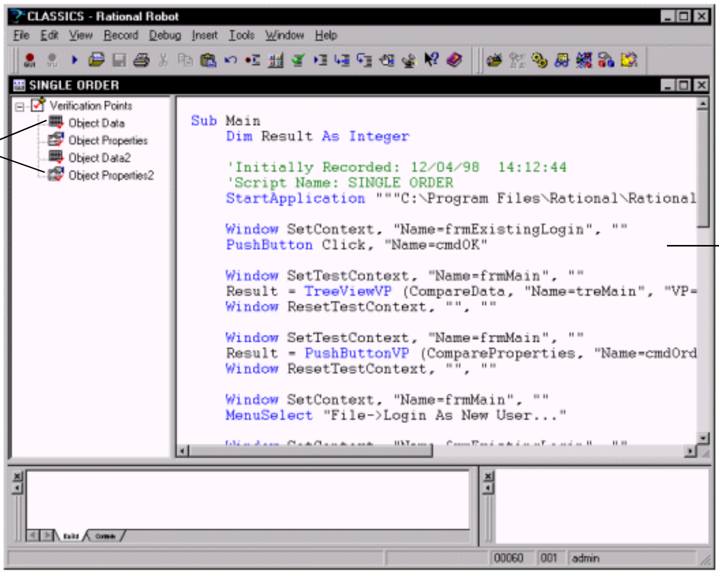
Click OK to capture all of the properties of the non-visible data control.

4. Click **OK** again to capture the properties of the control.

5. In the Object Properties Verification Point dialog box, scroll to the **Visible** property and notice that its value is **False**, indicating that the data control is non-visible but has been captured by Robot's Object Testing technology.
6. Click **OK** to complete the test.
7. Click **Close**.
8. Close Classics Online.
9. Press the **Stop Recording** button.

Reviewing the Verification Points You Recorded

After you stop recording, the Robot window appears.


Double-click each verification point to view what you just recorded in one of the Comparators.
The script
CLASSICS - Rational Robot
File Edit View Record Debug Insert Tools Window Help
SINGLE ORDER
Verification Points
Object Data
Object Properties
Object Data2
Object Properties2
00060
001
admin

```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 12/04/98  14:12:44
    'Script Name: SINGLE ORDER
    StartApplication """C:\Program Files\Rational\Rational

    Window SetContext, "Name=frmExistingLogin", ""
    PushButton Click, "Name=cmdOK"

    Window SetTestContext, "Name=frmMain", ""
    Result = TreeViewVP (CompareData, "Name=treMain", "VP=
    Window ResetTestContext, "", ""

    Window SetTestContext, "Name=frmMain", ""
    Result = PushButtonVP (CompareProperties, "Name=cmdOrd
    Window ResetTestContext, "", ""

    Window SetContext, "Name=frmMain", ""
    MenuSelect "File->Login As New User..."
```

About the Script You Recorded

The GUI script you recorded consists of commands written in is written in a high-level language. SQABasic, Robot's scripting language. SQABasic provides most of the syntax rules and core commands that are contained in the Microsoft Basic Language.

NOTE: To print the script, from Robot click **File** → **Print**.

What the Script Commands Mean

Every action begins as a **Window SetContext** command followed by the action. Comments begin with a single quote (').

Each verification point appears as a single line in the recorded script that begins with **Result =**

The following commands appear in the script:

- `Sub Main` - Indicates the beginning of the script.
- `Dim Result as Integer` - Declares the variable Result for verification point return values. The Result variable is local to the Main sub procedure.
- `'Initially Recorded:` - A comment that automatically shows the date and time when the script was recorded.
- `'Script Name:` - A comment that identifies the script.
- `Window SetContext` - Specifies the window where user actions occurred.
- `Window SetTestContext` - Specifies the window where a verification point was inserted.
- `Result` = Specifies that a verification point was recorded as part of the script. The verification point identifies the control by its internal object name and the script command recognition method.
- `Name` = Identifies all of the objects that were tested by their internal object names.
- `Pushbutton Click` - Indicates a user action command.

Summary

You've just recorded a script and inserted four verification points that check:

- The album data displayed in the tree control
- The text property of a pushbutton
- The data retrieved from the order-entry database
- The properties of a hidden data control

By developing scripts for an early build of an application, you'll be able to verify that future builds of the application perform as intended.

What's Next

Now that you've recorded a script, you're ready to play it back against the same build to verify that the script contains no errors and to establish a baseline for future testing.

Playing Back a Script

Objectives

- Play back the script on the same build to verify that all verification points pass.
- View the playback results in the LogViewer.

Scenario

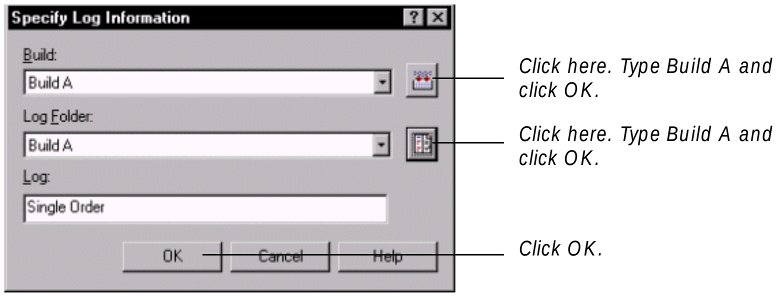
In the previous example, you recorded a script using Build A of the Classics Online application. In this example, you'll play back the script using the same build of the application to ensure that the script works as intended.

During playback, Robot compares the recorded data to the application-under-test. Since no changes have been made to the build, all verification points should pass. A verification point fails only if there are differences between the recorded baseline data and the current build of the application.

Playing Back a Script

To play back the SINGLE ORDER script against Build A:

1. Make sure that Robot is open.
2. From Robot, click **File** → **Playback**.
3. Select SINGLE ORDER and click **OK**.


Specify Log Information
Build:
Log Folder:
Log:
OK
Cancel
Help
Click here. Type Build A and click OK.
Click here. Type Build A and click OK.
Click OK.

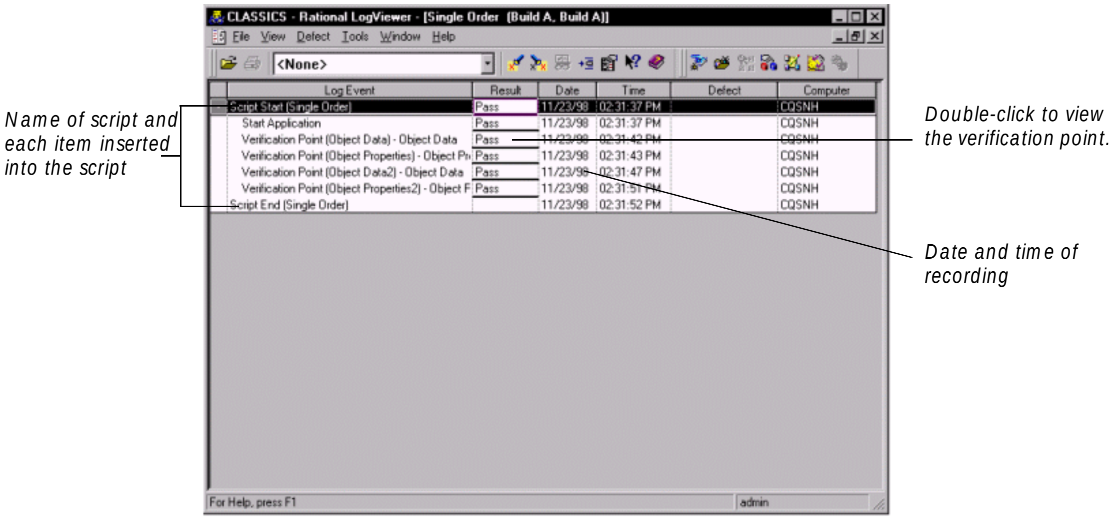
Build: Build A
Log Folder: Build A
Log: Single Order

The rest is automatic. Sit back and relax!

Robot plays back the sequence of recorded actions and verification points. The windows of the sample application are displayed on the screen as Robot plays back what you've recorded.

At the completion of playback, Robot updates the Rational repository and starts the LogViewer, another integrated Rational Test component.

The LogViewer shows the test results.


CLASSICS - Rational LogViewer - [Single Order (Build A, Build A)]
File
View
Defect
Tools
Window
Help
<None>
For Help, press F1
admin
Name of script and each item inserted into the script
Double-click to view the verification point.
Date and time of recording

Log Event	Result	Date	Time	Defect	Computer
Script Start (Single Order)	Pass	11/23/98	02:31:37 PM		CQSNH
Start Application	Pass	11/23/98	02:31:37 PM		CQSNH
Verification Point (Object Data) - Object Data	Pass	11/23/98	02:31:42 PM		CQSNH
Verification Point (Object Properties) - Object Pr	Pass	11/23/98	02:31:43 PM		CQSNH
Verification Point (Object Data2) - Object Data	Pass	11/23/98	02:31:47 PM		CQSNH
Verification Point (Object Properties2) - Object F	Pass	11/23/98	02:31:51 PM		CQSNH
Script End (Single Order)		11/23/98	02:31:52 PM		CQSNH

4. Click **File** → **Exit** to close the LogViewer and return to Robot.

Summary

You've just played back your recorded script against the same build of the application to verify that the script works as intended.

During playback, Robot compared the recorded data to the application-under-test. All verification points passed.

What's Next

Since the script passed, you'll use it as the baseline for future builds. When the script is played back and differences are found, you'll be able to use the LogViewer and its Comparators to decide whether the differences are intentional changes or defects in the build. And you'll be able to use Rational ClearQuest to track the defects until they are resolved.

Testing a New Build and Evaluating Results

Objectives

- Play back the SINGLE ORDER script to test a new build of the application.
- Use the LogViewer to review the playback results.
- Use the Comparators to analyze enhancements or defects in the application.

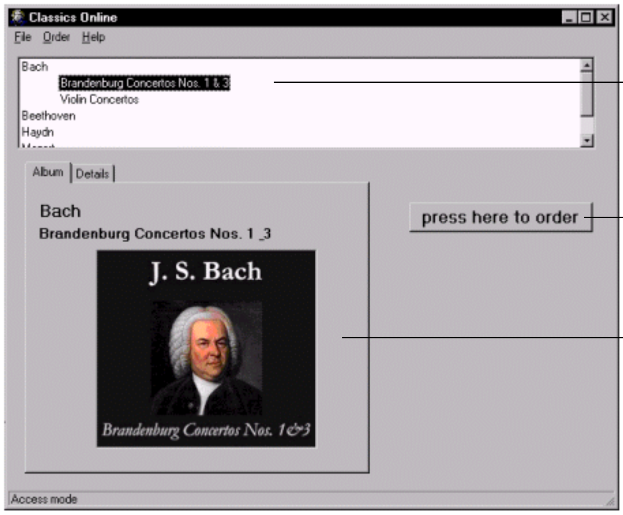
Scenario

In the previous example, you played back a script using Build A of the Classics Online application. In this example, there's a new build of the application that needs testing. You'll play back the recorded SINGLE ORDER script on the new build. Playback will take a fraction of the time that it would take to repeat the procedure manually.

Viewing the New Build

Before you play back the script against the new build, take a look at how much the Classics main window has changed from Build A to Build B.

Build A

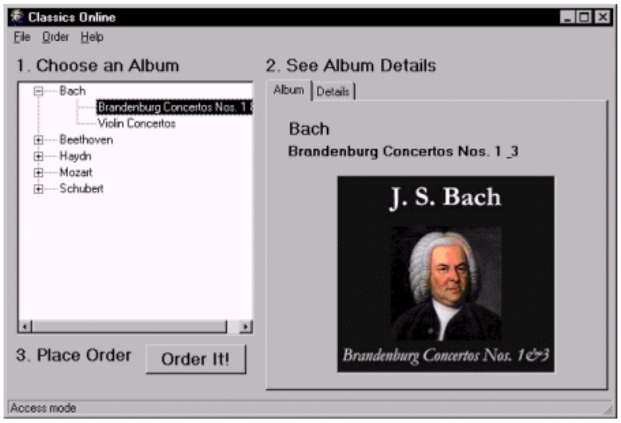


The look of the tree control changes in Build B. The data remains the same.

The text and location of the pushbutton change.

The layout changes a lot from Build A to Build B.

Build B



Even though objects in the new build are moved, rearranged, and renamed, you will be able to reuse the recorded script to test the objects. Robot's **Object-Oriented Recording** insulates scripts from changes and guarantees that you can play back scripts across builds.

With Object-Oriented Recording, Robot identifies objects by their internal object names. This ensures that even if the layout of the GUI changes and the objects in the GUI change locations, Robot will find the objects and test them against the baseline. If there are changes, Robot will flag them, and you'll decide whether they are enhancements or defects in the application.

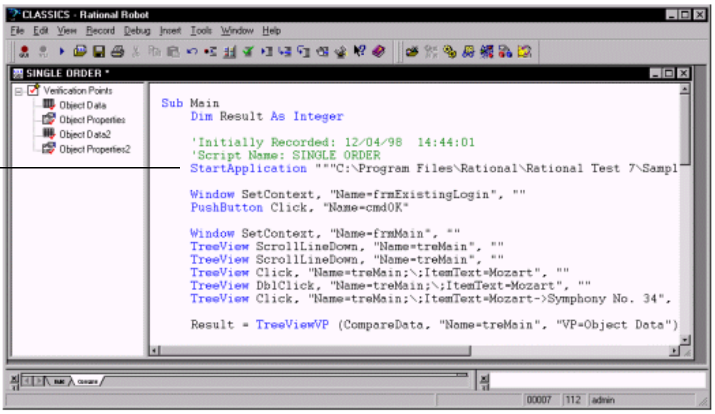
Playing Back a Script Against a New Build

Before you can play back the SINGLE ORDER script against a new build, you need to edit the Start Application path in the script so that Build B of the Classics application begins on playback.

To play back a script against a new build:

1. In Robot, click **File** → **Open Script** if the script is not open.
2. Select SINGLE ORDER and click **OK**.
3. You'll edit the script at the end of the **Start Application** line to change the build letter.

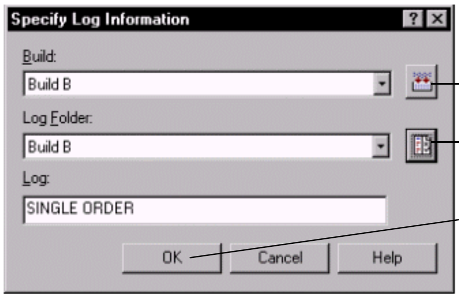
hange ClassicsA.exe
o ClassicsB.exe.



```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 12/04/98  14:44:01
    'Script Name: SINGLE ORDER
    StartApplication """C:\Program Files\Rational\Rational Test 7\Sampl

    Window SetContext, "Name=frmExistingLogin", ""
    PushButton Click, "Name=cmdOK"

    Window SetContext, "Name=frmMain", ""
    TreeView ScrollLineDown, "Name=treMain", ""
    TreeView ScrollLineDown, "Name=treMain", ""
    TreeView Click, "Name=treMain;\;ItemText=Mozart", ""
    TreeView DblClick, "Name=treMain;\;ItemText=Mozart", ""
    TreeView Click, "Name=treMain;\;ItemText=Mozart->Symphony No. 34",

    Result = TreeViewVP (CompareData, "Name=treMain", "VP=Object Data")
```

4. Click **File** → **Save**.
5. To begin playback, select the **Playback** button.

6. Select **SINGLE ORDER** and click **OK**.



Build: Build B

Log Folder: Build B

Log: SINGLE ORDER

Click here. Type Build B and click OK.

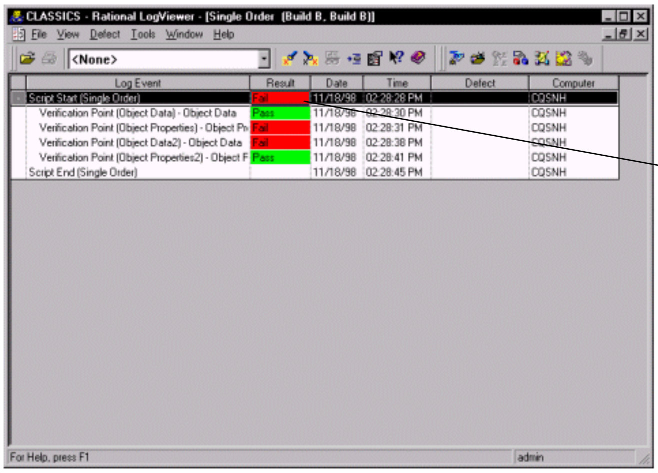
Click here. Type Build B and click OK.

Click OK.

Robot compiles the script and plays back the sequence of recorded actions and verification points. The windows of the sample application are displayed on the screen as Robot plays back what you've recorded.

Viewing the Test Results in the LogViewer

When playback is completed, Robot automatically opens the LogViewer so you can see the results of playing back the script against the new build. And, for some reason, two of the SINGLE SCRIPT verification points failed when played back against the new build. Read on to find out why.


CLASSICS - Rational LogViewer - [Single Order (Build B, Build B)]
File View Defect Tools Window Help
<None>
For Help, press F1
admin

Log Event	Result	Date	Time	Defect	Computer
Script Start (Single Order)	Fail	11/18/98	02:28:28 PM		CQSNH
Verification Point (Object Data) - Object Data	Pass	11/18/98	02:28:30 PM		CQSNH
Verification Point (Object Properties) - Object Pr	Fail	11/18/98	02:28:31 PM		CQSNH
Verification Point (Object Data2) - Object Data	Fail	11/18/98	02:28:38 PM		CQSNH
Verification Point (Object Properties2) - Object F	Pass	11/18/98	02:28:41 PM		CQSNH
Script End (Single Order)		11/18/98	02:28:45 PM		CQSNH

If any verification point in a script fails, the status for the entire script is Fail.

The test of the press here to order pushbutton failed, as did the test of the retrieval of data from the order-entry database.

Analyzing the Results in the Comparators

To let you analyze each failure and then remedy it, the LogViewer provides four Comparators:

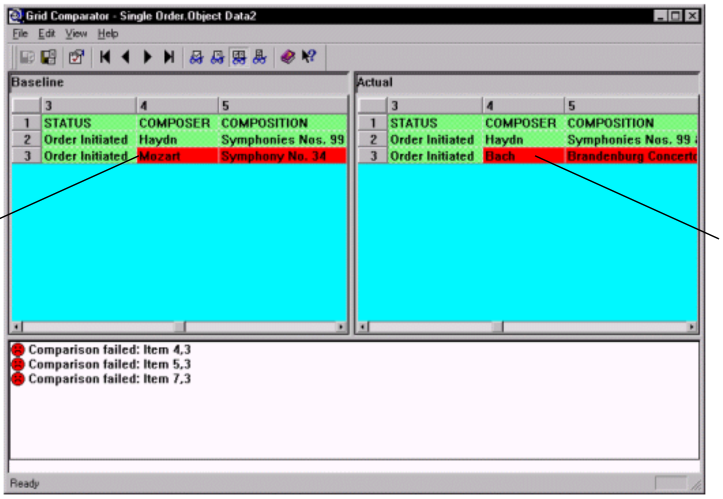
- **Grid -** Displays data files for text and numeric verification points that Robot displays in a grid.
- **Object Properties -** Displays the properties of objects captured using the Object Properties verification point.
- **Text -** Displays data files for text and numeric verification points in any format except grids.
- **Image -** Displays image files for the Region Image and Window Image verification points.

Each Comparator graphically displays the 'before and after' results of playback.

If there is no failure on playback, only a **baseline file** displaying the recorded data or image is displayed. If a failure occurs on playback, an **actual file** is displayed next to the baseline file. By comparing the files, you can determine whether a failure occurred because of an intentional change in the build or because of a defect.

Baseline file shows the album that was ordered.

Red indicates an error.



Baseline

	3	4	5
1	STATUS	COMPOSER	COMPOSITION
2	Order Initiated	Haydn	Symphonies Nos. 99
3	Order Initiated	Mozart	Symphony No. 34

Actual

	3	4	5
1	STATUS	COMPOSER	COMPOSITION
2	Order Initiated	Haydn	Symphonies Nos. 99
3	Order Initiated	Bach	Brandenburg Concerto

```
Comparison failed: Item 4,3
Comparison failed: Item 5,3
Comparison failed: Item 7,3
```

Actual file shows that different data was retrieved from the database. This indicates an error in the order-entry logic.

Viewing the Test of the Tree Control

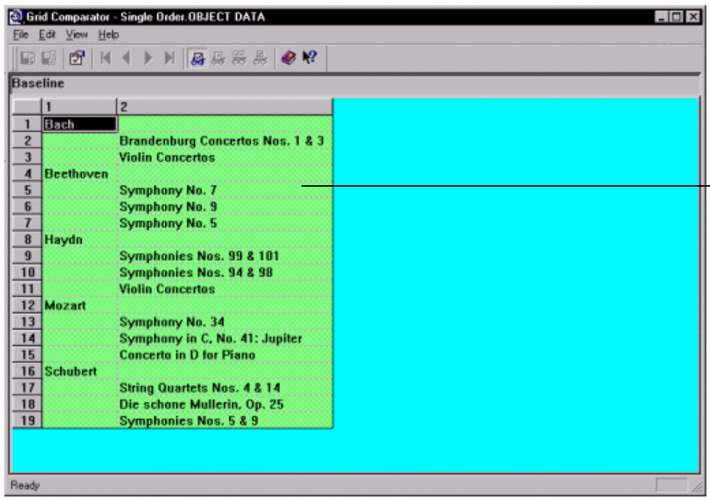
The first Object Data verification point passed even though the outward appearance of the tree control changed. The verification point checked the data in each branch of the tree control and found that the correct data for each branch was retrieved from the database.

If you had recorded an Object Properties verification point on the tree control, it would have failed because of the changes to some of the control's properties —lines, buttons, height, and width—in Build B.

To view the results of the first Object Data verification point:

1. Double-click **Verification Point (Object Data) – Object Data** in the **Log Event** column.

Because the verification point passed, the only file displayed in the Grid Comparator is the baseline file that contains the data tested.



	1	2
1	Bach	
2		Brandenburg Concertos Nos. 1 & 3
3		Violin Concertos
4	Beethoven	
5		Symphony No. 7
6		Symphony No. 9
7		Symphony No. 5
8	Haydn	
9		Symphonies Nos. 99 & 101
10		Symphonies Nos. 94 & 98
11		Violin Concertos
12	Mozart	
13		Symphony No. 34
14		Symphony in C, No. 41: Jupiter
15		Concerto in D for Piano
16	Schubert	
17		String Quartets Nos. 4 & 14
18		Die schone Mullerin, Op. 25
19		Symphonies Nos. 5 & 9

Baseline file show the results of the verification point.

2. Choose **File** → **Exit** to close the Comparator.

Viewing Two Verification Points That Failed

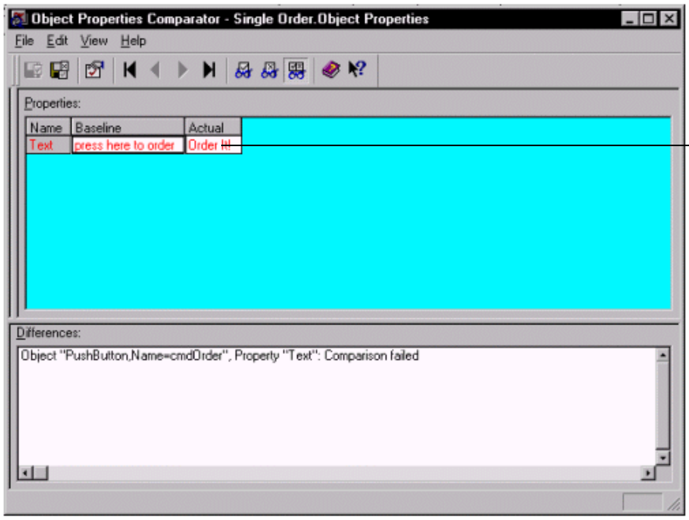
Two verification points failed. After you review and analyze the failures in the Comparators, you'll discover that one of the failures is an intentional code change in Build B. The other failure is a defect.

Viewing the Test of the Pushbutton

The first Object Properties verification point failed. It checked the text on the pushbutton and found that it had changed in Build B. Since the development team planned this change, it is an enhancement, not a defect. To ensure that the verification point passes the next time the script is played back, you need to incorporate the enhancement into the baseline script.

To view the results of the first Object Properties verification point:

1. Double-click **Verification Point (Object Properties) - Object Properties** in the **Log Event** column.


Object Properties Comparator - Single Order.Object Properties
File
Edit
View
Help
Properties:
Differences:

Name	Baseline	Actual
Text	press here to order	Order !!!

Object "PushButton,Name=cmdOrder", Property "Text": Comparison failed

The failure appears in red typeface.

The change is an intentional, minor enhancement.

2. Click **File** → **Replace Baseline with Actual.**
3. Click **Yes.**

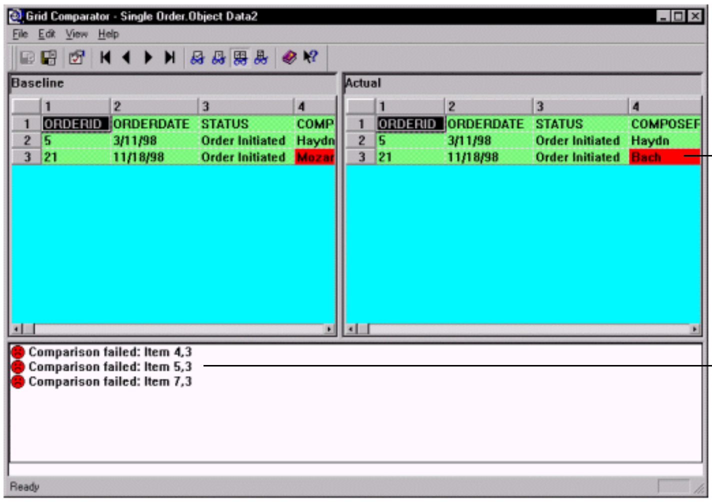
   The next time you play back SINGLE ORDER, the verification point will pass.
4. Choose **File** → **Exit** to return to the LogViewer.

Viewing the Test of the Data from the Order-Entry Database

The second Object Data verification point failed. It checked the data retrieved from the order-entry database and found that the incorrect data was retrieved.

To view the results of the second Object Data verification point:

1. Double-click **Verification Point (Object Data2) – Object Data** in the **Log Event** column.



Baseline

	1	2	3	4
1	ORDERID	ORDERDATE	STATUS	COMP
2	5	3/11/98	Order Initiated	Haydn
3	21	11/18/98	Order Initiated	Mozar

Actual

	1	2	3	4
1	ORDERID	ORDERDATE	STATUS	COMPOSEF
2	5	3/11/98	Order Initiated	Haydn
3	21	11/18/98	Order Initiated	Bach

```
Comparison failed: Item 4,3
Comparison failed: Item 5,3
Comparison failed: Item 7,3
```

The application retrieved the wrong record from the database.

An order was placed for a Mozart album, but a Bach album was ordered instead.

To quickly view the failure, click each of these, or click View → Next Difference.

Choose **File** → **Exit** to return to the LogViewer so you can report this error as a defect in the application.

Summary

You've just found how to use the LogViewer and Comparators to analyze changes in your application from build to build. And, you discovered how easy it is to update your test when an intentional change is made in the application.

In the example, you used the LogViewer and the Comparators to:

- View the results of the SINGLE ORDER script.
- Analyze each failure to determine if it was a defect or an intentional change.
- Update the baseline file with the actual file when the change was intentional.

What's Next

You're now ready to report the defect.

Specifying and Managing Defects

Objectives

- Use Rational ClearQuest to generate defect information.
- Use Rational ClearQuest to create an email rule to send email about the defect discovered in the sample application.

NOTE: You can follow the steps in this example if you have Rational ClearQuest installed. See the next page for instructions about verifying that ClearQuest is installed.

Scenario

When you played back the SINGLE ORDER script on Build B of the Classics application, two of the verification points failed. One of the failures was an intentional change to Build A—the name change on the order button. The other failure was a defect in the order-entry logic of the application. Even though an order was placed for a Mozart album, a Bach album was ordered instead.

In this example, you'll use Rational ClearQuest to enter the defect in the repository. ClearQuest lets you track defects through each phase of the development and testing process.

Making Sure That Rational ClearQuest Is Available

To see if Rational ClearQuest is available for defect tracking:

1. Click **Start** → **Programs** → *Rational product name*.
2. Check to see if ClearQuest is listed as a program on the submenu.
3. Do not start ClearQuest yet.

Creating a ClearQuest Master Database

Before you can start using ClearQuest to generate defects, you need to create a sample ClearQuest master database. The **master database** stores the schema used in the defect tracking form.

To create a ClearQuest master database:

1. Start the Rational Administrator.
2. If necessary, click **File** → **Connect** → **ClassicsRepository** and log in as **admin**.
3. Click **Tools** → **Rational ClearQuest Maintenance Tool**.
4. Click **Create a new schema repository** and click **Next**.
5. Keep MS Access as the database type.
6. Type **C:\Master.mdb** as the path for the database where the schema is stored.
7. Click **Next**.
8. Click **Yes** because other users will not use this sample master database.
9. Select **Create sample database** and click **Next**. This creates a sample user database that you can connect to the sample master database so you can store your defects for this tutorial.
10. Type **C:\User.mdb** as the path for the database where you'll store you defect data.
11. Click **Next**.
12. Click **Finish**.
13. Click **Done**.

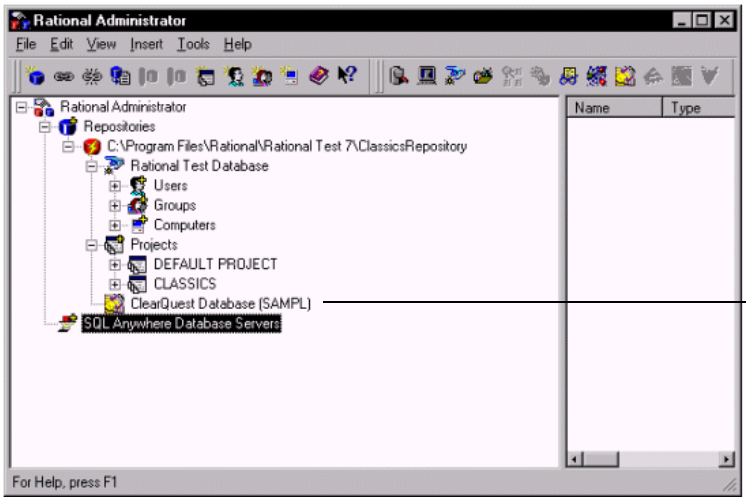
Attaching a ClearQuest Database to a Repository

In this tutorial, after you create the ClearQuest master database, you have to attach a ClearQuest user database called **SAMPL** to the sample repository. This **user database** contains all the defect data that you'll enter and the data that the LogViewer automatically generates.

By attaching the user database to the sample repository, you can be sure that you have access to the correct schema. The schema determines how the defect tracking form looks.

To attach the ClearQuest user database:

1. From the Rational Administrator, click **File** → **Attach ClearQuest Database**.
2. Accept **admin** as your temporary user ID and click **Next**.
3. Click **SAMPL** to select it as the database and click **Next**.
4. Click **Finish**.


Rational Administrator
File Edit View Insert Tools Help
Rational Administrator
Repositories
C:\Program Files\Rational\Rational Test 7\ClassicsRepository
Rational Test Database
Users
Groups
Computers
Projects
DEFAULT PROJECT
CLASSICS
ClearQuest Database (SAMPL)
SQL Anywhere Database Servers
Name
Type
For Help, press F1
The new defect database

Notice that ClearQuest database SAMPL is now in the tree under ClassicsRepository.

Generating a Defect from the LogViewer

When Robot discovers a change in your application, it logs the change in the LogViewer. From the LogViewer, you can then generate a defect so that the information is automatically transferred to the ClearQuest defect database.

When the defect is generated, the information from the LogViewer appears in a defect form. To enter a defect, you add information to the defect form and assign an owner and status to it.

To generate a defect:

1. Make sure that the LogViewer is open.

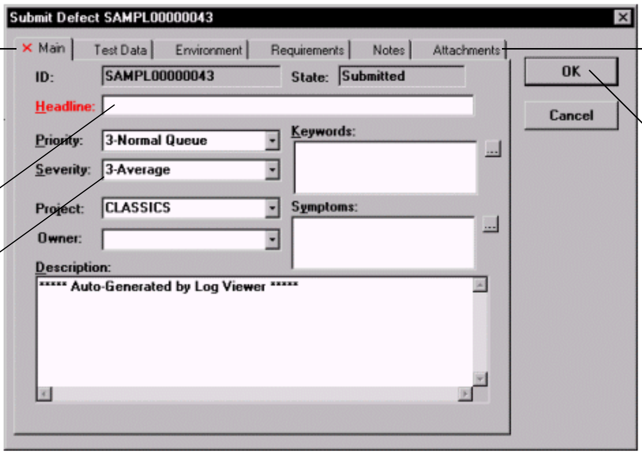
> **NOTE:** If you closed the LogViewer, open the Log Viewer and then do the following to open the correct log: Click **File** → **Open**. Double-click **Build B**. Double-click the **Build B** folder. Double-click **Single Order**.

2. In the LogViewer, select **Verification Point (Object Data2)**.
3. Click **Defect** → **Generate** to open the defect form.

You must type information in every tab that has a red X and in every field that has a red label.

Type First Failure as the headline for the defect.

Select Average for the Severity.



ID: SAMPL00000043
State: Submitted
Headline:
Priority: 3-Normal Queue
Severity: 3-Average
Project: CLASSICS
Owner:
Keywords:
Symptoms:
Description:
***** Auto-Generated by Log Viewer *****

Click the other tabs to see more auto-generated information.

Click OK to submit the defect to the ClearQuest database.

Accessing the Defect from the ClearQuest Database

After you generate a defect from the LogViewer, you can access the defect directly from ClearQuest.

To access the defect from ClearQuest:

1. From the LogViewer, click the ClearQuest icon.
2. In the left pane, double-click **All Defects** to view the defects in the **SAMPL** database.
3. To view the defect you just entered, double-click **Keyword Search** in the left pane.
4. Type **First Failure** and click **OK**.
5. Click **File** → **Close** to close the defect.

Sending Email About a Defect

After a defect is generated, either you can run ClearQuest to search for the defects assigned to you, or your project leader can configure ClearQuest to automatically send email about the defect to the person responsible for it.

Setting Up an Email Rule

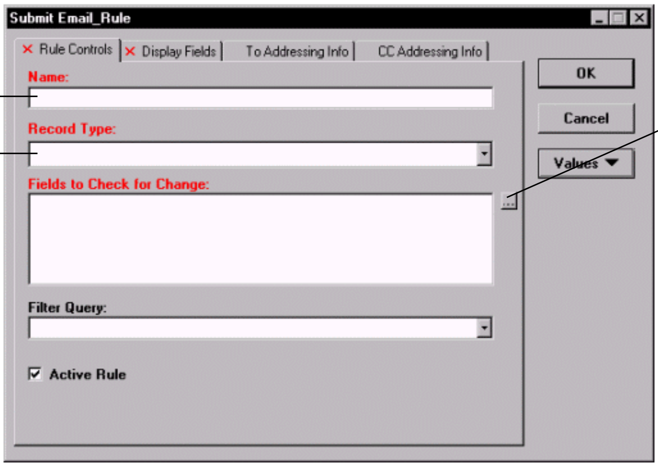
Before ClearQuest can automatically generate email about a defect, you must create one or more email rules that define when and to whom the defect will be emailed. You can create an email rule if you have Super User or Schema Designer privileges. And, for this tutorial, you do!

To set up an email rule:

1. From ClearQuest, click **Actions** → **New**.
2. Select **Email_rule** and click **OK**.

Type Send to Owner as the name of the rule.

Select Defect.



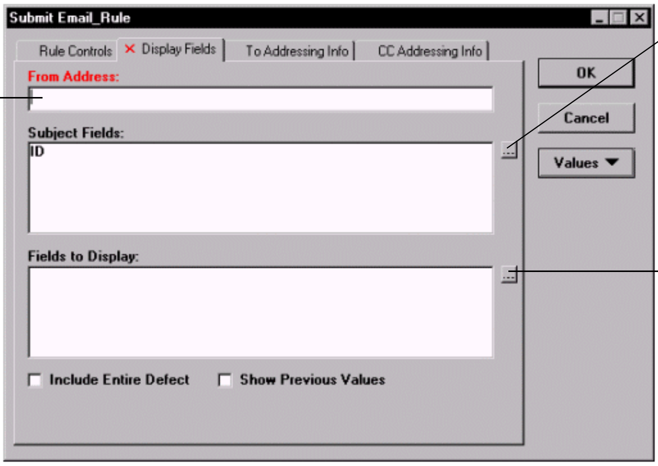
Click to display the list of available fields

Select Owner

Click Add To and click OK.

If the owner of th defect changes, t email rule is triggered and the defect is emailed

3. Select the **Display Fields** tab.

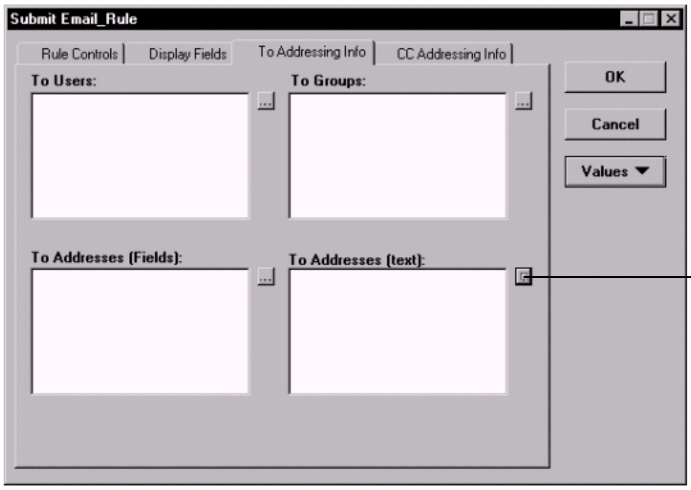


Type your email address.

Click here, and the add Headline. Click OK. The Information w appear as the subject of the ema

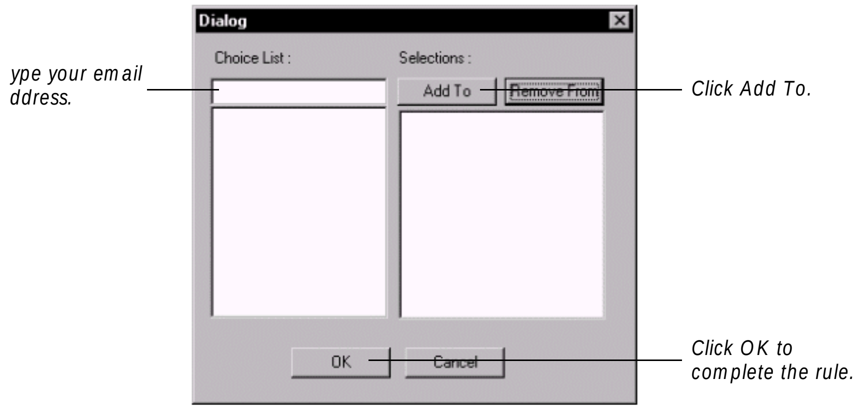
Click here, and the add Owner, Priority and Requirement. Click OK. The Information and the defect ID will appear in the email.

4. Select the **To Addressing Info** tab.



Click here to add your email address so the email can be sent to you.

5. To add your email address, do this:



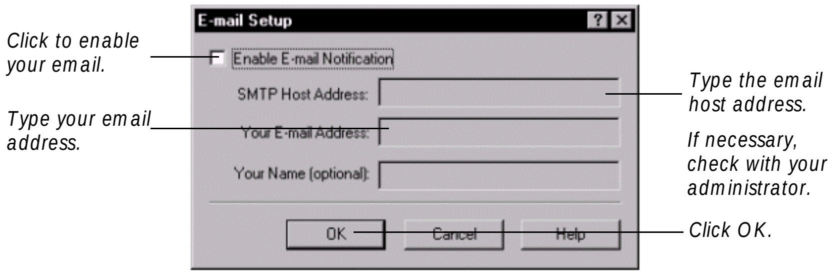
6. Click **OK** to close the Submit Email Rule dialog box.

Enabling Email Notification

After you've created the email rule, you enable email notification so you can receive email about defects.

To enable email notification:

▶ From ClearQuest, click **View** → **Email Options**.



Sending a Trial Email

Now that you've set up email notification, you can have a defect mailed to you. You'll trigger email by changing the owner of the defect.

To send a trial email:

1. Make sure that your email software is running.

2. From ClearQuest, double-click **All Defects** to display the defect list.

3. Under **All Defects**, double-click **Keyword Search**.

4. Type **First Failure** and click **OK**.
5. Click **Actions** and select **Modify**.
6. In the **Main** tab, select **QE** as the new owner.
7. Click **Apply** and see what happens!

You'll receive an automatic defect notification.

Summary

You've just found out how easy it is to generate a defect about a failed verification point displayed in the LogViewer. You were also able to send an email message about the defect so it could be fixed.

What's Next

You've successfully used the SINGLE ORDER script to test a portion of the Classics Online application. Now it's time to run a few reports about the testing effort to inform management and the rest of your team about where things stand.

Using Reports to Manage Test Progress

Objectives

- Run a Script Summary report from TestManager. The report lets you view all of the scripts that are planned for the Classics Online application.
- Run a Defect Summary report from ClearQuest. The report lets you view the open defects for the Classics Online application.

Scenario

As your testing progressed from defining test requirements to creating scripts, and from playing back the scripts to tracking defects, all of the test results were stored in the Rational repository. You were able to review and analyze the test results in the LogViewer. And now you're ready to create reports about the testing effort.

TestManager and ClearQuest provide integrated report writers. You can use them to create and customize dozens of graphs and reports that help you manage the progress of your testing effort.

In this example, you'll generate two reports. The first summarizes all of the scripts in the Classics project. The second lists all of the defects that are "open."

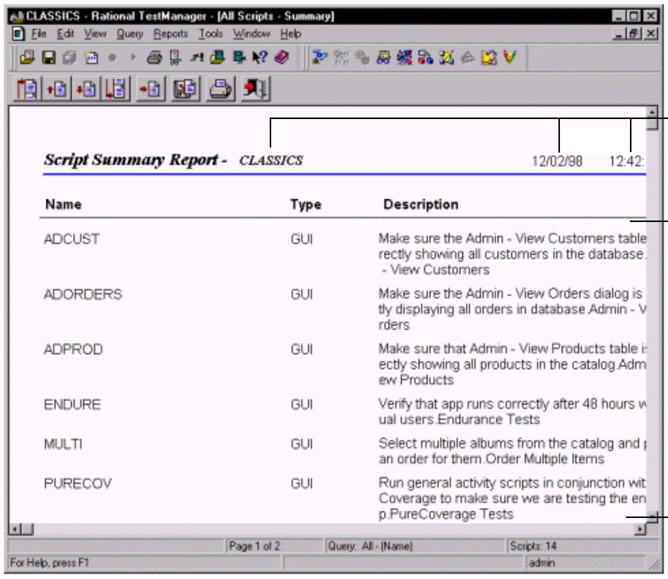
Running a Script Summary Report

The **Script Summary** report provides you with overview information for all of the scripts in the Classics project.

To run the Script Summary report:

1. From TestManager, choose **Reports** → **Run**.
2. Select **Script Listing** → **All Scripts** → **Summary** and click **OK**.

   The report is generated automatically.


CLASSICS - Rational TestManager - [All Scripts - Summary]
File Edit View Query Reports Tools Window Help

Script Summary Report - *CLASSICS* 12/02/98 12:42:

Name	Type	Description
ADCUST	GUI	Make sure the Admin - View Customers table rectly showing all customers in the database. - View Customers
ADORDERS	GUI	Make sure the Admin - View Orders dialog is tly displaying all orders in database.Admin - V rders
ADPROD	GUI	Make sure that Admin - View Products table i: ectly showing all products in the catalog.Adm ew Products
ENDURE	GUI	Verify that app runs correctly after 48 hours w ual users.Endurance Tests
MULTI	GUI	Select multiple albums from the catalog and an order for them.Order Multiple Items
PURECOV	GUI	Run general activity scripts in conjunction wit Coverage to make sure we are testing the en p.PureCoverage Tests

Page 1 of 2 Query: All - (Name) Scripts: 14
For Help, press F1 admin

Displays the project name as well as the date and time the report was run.

Lists the name of each script, its type, and the description entered when the script was defined in TestManager.

3. Scroll down to view the SINGLE ORDER script, and then close the report window.

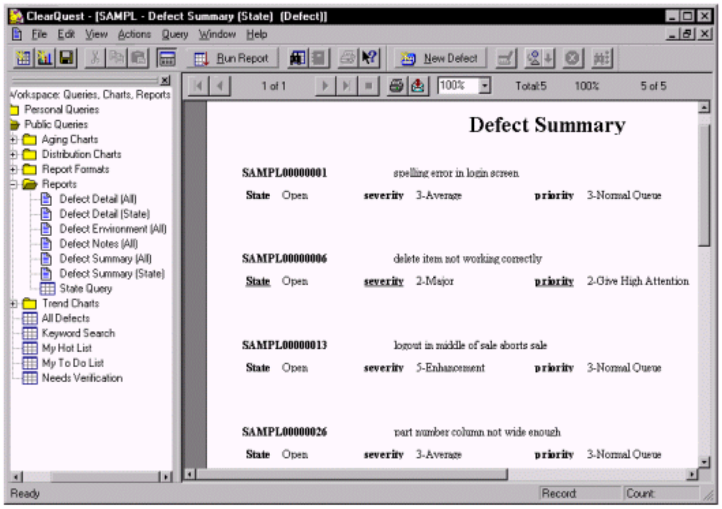
Running a Defect Summary Report

ClearQuest provides two default Defect Summary reports. The first lists all of the defects in the database. The second filters the defects according to their states—Submitted, Open, Resolved, Closed, or Duplicate.

In this example, you'll generate a report that lists all of the Open defects for the Classics project.

To run a defect report:

1. From ClearQuest, double-click **Public Queries** in the left pane to expand its branches.
2. Double-click **Reports**.
3. Select **Defect Summary (State)**.
4. Right-click, and then click **Run**.
5. Select **Open** and click **OK**.
6. To view the report, click **100%** and maximize the window.



Defect Summary

SAMPL00000001		spelling error in login screen				
State	Open	severity	3-Average	priority	3-Normal Queue	
SAMPL00000006		delete item not working correctly				
State	Open	severity	2-Major	priority	2-Give High Attention	
SAMPL00000013		logout in middle of sale aborts sale				
State	Open	severity	5-Enhancement	priority	3-Normal Queue	
SAMPL00000026		part number column not wide enough				
State	Open	severity	3-Average	priority	3-Normal Queue	

Lists each defect and its state, severity, and priority.

7. To print the report, click the printer icon.

Summary

TestManager and ClearQuest provide integrated report writers that you can use to create and customize dozens of graphs and reports to help you manage the progress of your testing effort.

In this example, you ran two reports. The first listed all of the scripts that are planned for the sample application. The second listed all of the "open" defects in the application.

What's Next

The first six examples have shown you how the integration between Rational Test products ensures a successful testing effort.

You did the following:

- Used Rational TestManager to define test requirements and a script.
- Used Rational Robot to record a script and play back the script against two different builds of the sample application.
- Used the Rational LogViewer and comparators to analyze test results.
- Used Rational ClearQuest to generate a defect.
- Used Rational TestManager and Rational ClearQuest to run reports about your testing effort.

This tight integration between products as well as their ease-of-use lets your development and testing teams work together to ensure that the software testing effort goes forward as planned.

If you still have time, take a look at the last section in this tutorial. It explains advanced scripting techniques that you'll find useful as you learn more about Robot and automated functional testing.

Enhancing Your Scripts

Objective

- Customize the SINGLE ORDER script by adding a header file and a library file that checks to make sure the order-entry database is reset before each playback of the script.

Scenario

When you recorded the SINGLE ORDER script, you inserted four verification points. One of them placed a new order in the order-entry database.

If the database is not reset to its initial state with the new order deleted before each playback of the script, the same order will be re-entered in the database, and the script will fail on playback. To make sure that the database is reset, you have to test to make sure that the newly entered orders are actually deleted and the database reset to its initial state.

To accomplish this, you must do the following :

- Delete the new record from the database.
- Enhance the SINGLE ORDER script by creating a procedure that checks the state of the order-entry database. You'll put the procedure in a library file that the SINGLE ORDER script calls on playback.

Deleting the New Record from the Database

To reset the database to its initial state, you have to delete the new order from the database.

To delete the new order from the database:

1. Make sure that the sample application is open.
2. Log in as Susan Flontly and click **OK.**
3. To view Susan's orders, **Order** → **View Existing Order Status.**
4. Select **Symphony No. 34** and click **Cancel Selected Order.**
5. Make sure that the only order in the database is **Haydn's Symphonies Nos. 98 & 101.**
6. Click **Close.**

Recording and Manually Customizing a Script

To make sure that the new order has been deleted, you'll manually customize the script instead of using Robot to automatically create an entire script. You'll customize the script by adding two SQABasic commands. And, you'll only use Robot to record navigational actions.

Finally, you'll copy the contents of the new script into a library file that will be called by the SINGLE ORDER script. Adding the library file to the SINGLE ORDER script tests to make sure that the order is deleted and the database reset before each playback of SINGLE ORDER.

Using Robot to Record Navigational Actions

You'll follow these steps to record navigational actions:

- Start the Classics Online sample application.
- Log in as Susan Flontly.
- Open the View Existing Order dialog box. (After all the navigation is recorded, it's here in the script that you'll manually add a command to test the order-entry grid in the dialog box.)
- Close the dialog box.
- Close the sample application. (After all the navigation is recorded, it's here in the script that you'll manually add the command to send a message to the LogViewer.)

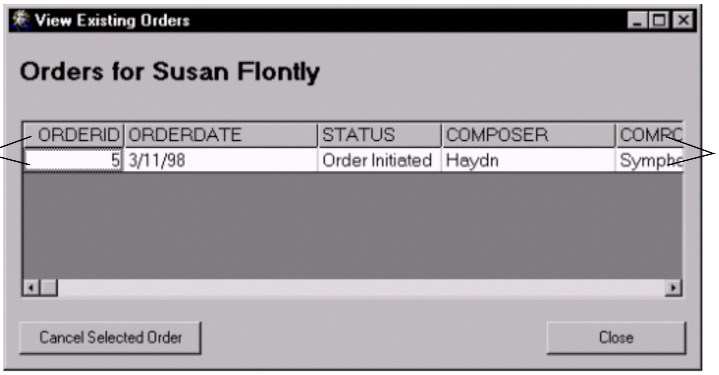
By using Robot to record the actions, you'll save yourself the time and energy of manually writing the actions in the script.

To record the necessary navigational actions:

1. From Robot, click **File** → **Record GUI**.
2. Type Temporary Script and click OK.
3. Click the **Display GUI Insert Toolbar** button.
4. Click the **Start Application** button.
5. Type or browse to the default location for Build A of the sample application:

   ```
   C:\Program Files\Rational\Rational Test 7\Sample
   Applications\Classics Online\ClassicsA.exe
   ```
6. Click **Open** and click **OK**.
7. Log in as **Susan Flontly** and click **OK**.
8. Click **Order** → **View Existing Order Status**.

Order-entry grid


View Existing Orders

Orders for Susan Flontly

ORDERID	ORDERDATE	STATUS	COMPOSER	COMPC
5	3/11/98	Order Initiated	Haydn	Symphe

Before the new order is placed for Susan Flontly, there are two rows in Susan's record—one is the header row and the other is an existing order.

9. Close the dialog box and the sample application.
10. Press the **Stop Recording** button.

Manually Customizing the Script

Now that you've recorded the actions, you'll customize the temporary script by adding two SQABasic commands—**SQAGetProperty** and **SQALogMessage**.

The first command will capture the row property of the existing order grid to make sure that the new order no longer appears.

The second command will write a message to the LogViewer. The message will indicate whether the database has been returned to its initial state.

NOTE: To become familiar with the syntax of the commands and to view comments about them, see the *SQABasic Language Reference* or the SQABasic online Help.

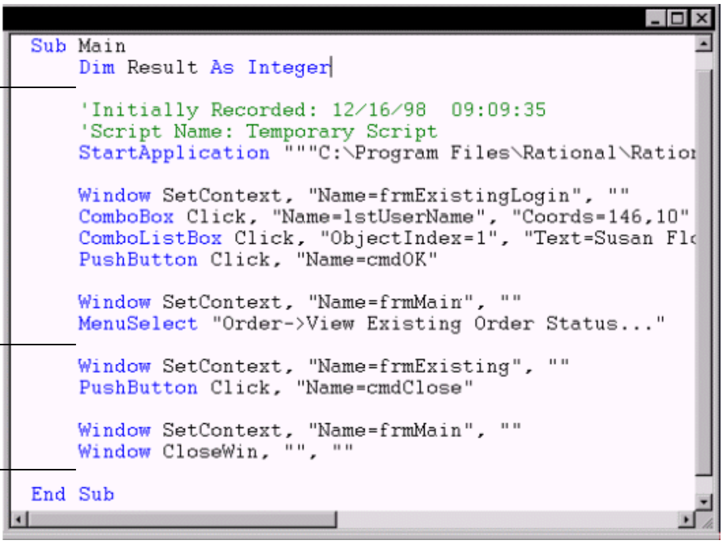
To customize the script:

1. Make sure the **Temporary Script** is open.

You'll declare a variable here that will contain the number of rows in the grid.

You'll add a command here to check for the number of rows in Susan Flontly's record.

You'll ad an If-Then statement here that sends a message to the LogViewer about the database reset



```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 12/16/98  09:09:35
    'Script Name: Temporary Script
    StartApplication """C:\Program Files\Rational\Ratio

    Window SetContext, "Name=frmExistingLogin", ""
    ComboBox Click, "Name=lstUserName", "Coords=146,10"
    ComboListBox Click, "ObjectIndex=1", "Text=Susan Fl

    PushButton Click, "Name=cmdOK"

    Window SetContext, "Name=frmMain", ""
    MenuSelect "Order->View Existing Order Status..."

    Window SetContext, "Name=frmExisting", ""
    PushButton Click, "Name=cmdClose"

    Window SetContext, "Name=frmMain", ""
    Window CloseWin, "", ""

End Sub
```

2. At the top of the script after the first **Dim** statement, add:

```
Dim RowCount As Variant
```

3. After the `MenuSelect` command, add an SQABasic command. The command retrieves the **Rows** property from the FlexigridOrders control. The property contains the number of rows in the grid—the number includes the header row and the record row—and puts the number into the RowCount variable:

   ```
   Result = SQAGetProperty ("\;Name=frmExisting;\;
   Name=FlexigridOrders", "Rows", RowCount)
   ```

   NOTE: Make sure that the command appears as one line in the script.

4. At the end of the script before `End Sub`, add the following "If-Then-Else" statement. If the cleanup is successful, this will send a message to the LogViewer stating that the database is reset correctly:

   ```
   If RowCount=2 Then
   'Display message in LogViewer
   SQALogMessage sqaPass, "Database is reset, ready for playback", ""
   Else
   SQALogMessage sqaFail, "Database is not reset", ""
   End If
   ```

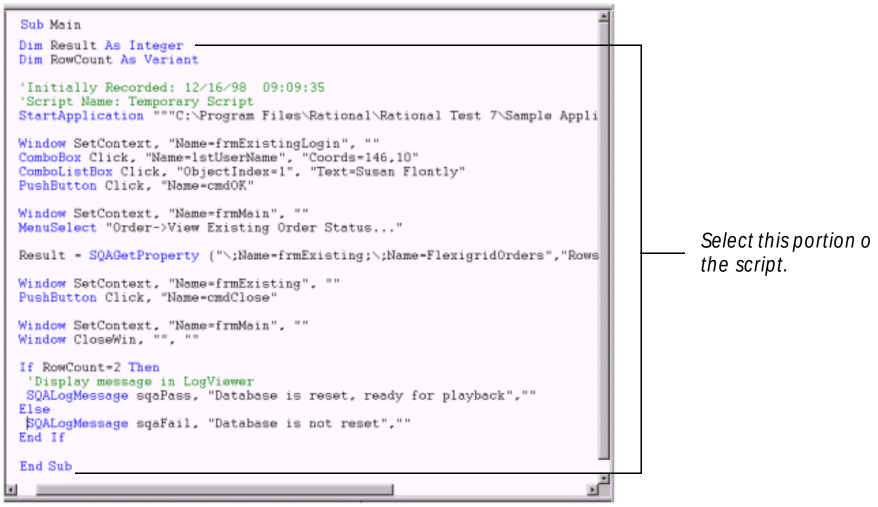
5. Click **File** → **Save** to save the additions to **Temporary Script**.

Creating a Library File

After you record the temporary script and add the SQABasic commands, you need to create a library file to copy contents of the script into.

To create a library file:

1. Make sure that **Temporary Script** is open.
2. Select a portion of the script beginning with `Dim Result As Integer` and ending with `End Sub`.



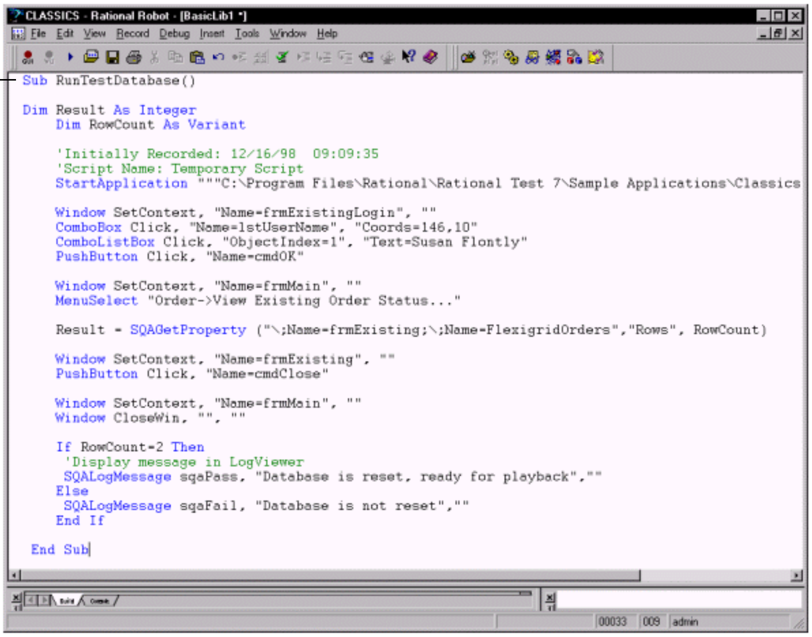
```
Sub Main
Dim Result As Integer
Dim RowCount As Variant

'Initially Recorded: 12/16/98  09:09:35
'Script Name: Temporary Script
StartApplication """C:\Program Files\Rational\Rational Test 7\Sample Appli

Window SetContext, "Name=frmExistingLogin", ""
ComboBox Click, "Name=lstUserName", "Coords=146,10"
ComboListBox Click, "ObjectIndex=1", "Text=Susan Flontly"
PushButton Click, "Name=cmdOK"

Window SetContext, "Name=frmMain", ""
MenuSelect "Order->View Existing Order Status..."

Result = SQAGetProperty ("\;Name=frmExisting;\;Name=FlexigridOrders","Rows

Window SetContext, "Name=frmExisting", ""
PushButton Click, "Name=cmdClose"

Window SetContext, "Name=frmMain", ""
Window CloseWin, "", ""

If RowCount=2 Then
 'Display message in LogViewer
 SQALogMessage sqaPass, "Database is reset, ready for playback",""
Else
 SQALogMessage sqaFail, "Database is not reset",""
End If

End Sub
```

3. Click **Edit** → **Copy**.
4. Begin to create the library file by clicking **File** → **New** → **SQABasic File**.
5. Click **Library Source File** and click **OK**.
6. Click **Edit** → **Paste** to copy the script into the library file.

7. At the top of the library file, type this line to name the procedure that you pasted into the file:

```
Sub RunTestDatabase ()
```

Type the name of the procedure here.


CLASSICS - Rational Robot - [BasicLib1 *]
File Edit View Record Debug Insert Tools Window Help
00033 009 admin

```
Sub RunTestDatabase()

Dim Result As Integer
    Dim RowCount As Variant

    'Initially Recorded: 12/16/98  09:09:35
    'Script Name: Temporary Script
    StartApplication """C:\Program Files\Rational\Rational Test 7\Sample Applications\Classics

    Window SetContext, "Name=frmExistingLogin", ""
    ComboBox Click, "Name=lstUserName", "Coords=146,10"
    ComboListBox Click, "ObjectIndex=1", "Text=Susan Flontly"
    PushButton Click, "Name=cmdOK"

    Window SetContext, "Name=frmMain", ""
    MenuSelect "Order->View Existing Order Status..."

    Result = SQAGetProperty ("\;Name=frmExisting;\;Name=FlexigridOrders","Rows", RowCount)

    Window SetContext, "Name=frmExisting", ""
    PushButton Click, "Name=cmdClose"

    Window SetContext, "Name=frmMain", ""
    Window CloseWin, "", ""

    If RowCount=2 Then
     'Display message in LogViewer
     SQALogMessage sqaPass, "Database is reset, ready for playback",""
    Else
     SQALogMessage sqaFail, "Database is not reset",""
    End If

 End Sub
```

8. Click **File** → **Save**.

9. Type **CheckDBLib** and click **Save**.

10. Click **File** → **Compile** to compile the library file.

Creating a Header File

After you create the library file, you'll create a header file that declares the custom procedure you just created. And you'll reference the header file from the SINGLE ORDER script.

To create the header file:

1. From Robot, click **File** → **New** → **SQABasic File**.

2. Click **Header File** and click **OK**.

3. Type the following line in the header file to declare the library file that contains the subprocedure RunTestDatabase:

```
Declare Sub RunTestDatabase BasicLib "CheckDBLib" ()
```

4. Click **File** → **Save**.
5. Type **CheckDBHeader** as the .sbh header file.
6. Click **Save**.
7. Click **File** → **Close**.

Adding the Header and Library Files to the Script

Finally, you'll reference the header file in the SINGLE ORDER script. And, you'll call the procedure (RunTestDatabase) from the script. Calling the procedure will check whether the database is reset and will send a message to the LogViewer.

To reference the header file and call the procedure:

1. Click **File** → **Open Script**.
2. Select **Single Order** and click **OK**.
3. At the top of the script above `Sub Main`, type:

```
'$Include "CheckDBHeader.sbh".
```

4. At the end of the script above the line `End Sub`, type:

```
Call RunTestDatabase()
```

5. Play back the script and see what happens!

The script will play back, check to make sure that the database is reset, and then write a Log Message to the LogViewer.

Summary

You've just discovered how easy it is to edit and customize a Robot script to make it do exactly what you want it to do.

You found out how to:

- Add SQABasic commands to a script.
- Create a header file.
- Create a library file.

What's Next

Congratulations on completing the tutorial!

You're now ready to use Rational Robot to test your own application. But before you begin, you need to create a new Rational repository to store your testing data.

Here's a quick procedure for creating a repository. For more detailed instructions and information about repositories, see the *Using the Rational Administrator* manual.

To create a Rational repository:

1. Close all Rational Test products.
2. Click **Start** → **Programs** → *Rational program name* → **Rational Administrator**.
3. Click **File** → **Create Repository**.
4. Type the drive, directory, and name for the new repository—for example, **C:\repo1**.
5. Click **Next**.
6. Accept **MS Access** as the database type and click **Next**.
7. Accept **Do not initialize the new repository with data from an existing repository** and click **Next**.
8. If you created a ClearQuest master database as part of this tutorial, accept **Create or Attach ClearQuest Database** and click **Next**. Otherwise, clear it, click **Next**, and skip to step 10.
9. Enter a database name—a limit of five alphanumeric characters beginning with a letter. Click **Next**.
10. Accept the defaults and click **Finish**.

Remember, if you want to find out more about Rational Robot, you can follow the testing tips in the *Rational Robot Try it!* cards. Also, for more complete information and instructions about using Rational Robot and its companion products, take a look at the user's guide and online Help for each product.

▸ ▸ ▸ Index