Getting Started with Rational TestFactory

Version 2000.02.10

Rational®
the e-development company™

Getting Started with Rational TestFactory

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

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

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

Revised 04/2000

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

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

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

P/N 800-023363-000

Contents

Introducing Rational TestFactory

Welcome to Rational TestFactory. You’re about to discover how TestFactory brings a new level of advanced automation to software quality testing.

Rational TestFactory is the next-generation automated software quality tool that automatically generates scripts that thoroughly test applications written in Microsoft® Visual Basic®, Microsoft Visual C++®, and Java, as well as Java applets. TestFactory amplifies your productivity by dramatically reducing the manual effort required to test software. Because it automatically models an application, builds regression test suites, and finds defects, you can start using TestFactory at any phase in the development cycle. You can generate scripts that flush out defects and provide extensive product coverage as soon as your application has a user interface to test.

TestFactory builds on Rational Robot’s capabilities to develop and run regression tests that validate specific paths through an application. It takes advantage of the advanced object recognition and playback features of Robot, and measures the product coverage its scripts provide. TestFactory also provides detailed coverage data on scripts created in Robot.

This release of TestFactory is designed for testing 32-bit applications written in Java, C++, and Microsoft Visual Basic (versions 4.0, 5.0, and 6.0). TestFactory and TestAccelerator run on the following Microsoft Windows platforms:

- Microsoft Windows NT® 4.0
- Microsoft Windows® 95
- Microsoft Windows® 98
- Microsoft Windows® 2000

TestFactory Features

TestFactory offers the following features:

- Automatically creates and maintains a detailed map of controls and actions in the user interface of your application.
- Lets you map and test every available path in a functional area of the application-under-test.
- Automatically generates scripts that provide extensive coverage of your application’s source code.
- Tracks executed and unexecuted source code and reports its findings.
- Automatically generates regression Test Suites containing scripts that uncover serious defects in your application.
- Simplifies maintenance of test assets, which means that testing new builds requires minimal effort.
- Lets you organize scripts into Test Suites and run them as batch jobs.
- Lets you quickly compose scripts that simulate user action sequences to increase the validity of your test assets.
- Provides code coverage data for Robot-recorded scripts, which you can include in your TestFactory Test Suites.
- Together with TestAccelerator, lets you run multiple Test Suites, Pilots, and scripts simultaneously on the Test Lab machines that you set up.
- Together with the Test Code Changes add-in for Visual Studio, lets you run scripts that test changed source code files from within the Visual Basic or Visual C++ development environment.

‣ ‣ ‣ About This Tutorial and the Sample Application

This tutorial is designed to get you up and running on TestFactory within a few hours. The examples take you through a sequence of tasks that you can perform later on your own application using TestFactory.

This tutorial contains information about how to use TestFactory to do the following:

- Instrument the source code of a sample application written in Visual Basic.
- Map the user interface of the sample application.
- Run Pilots that test selected regions of the application.
- Create Test Suites to run multiple scripts as batch jobs.
- Use the AutoPilot to run multiple test objects.
- Use TestFactory together with its ancillary program Rational TestAccelerator to run tests on a Test Lab machine.

NOTE: Please be sure that Microsoft Visual Studio 6.0 (Service Pack 2) or higher version is installed on your system. If it is not, you will not be able to run the tutorial.

About the Tutorial Examples

This tutorial contains a section that describes how to install the sample application you'll use throughout the tutorial, as well as seven example sections. Each example gives you instructions for completing one or more tasks that you would perform during the course of a TestFactory project. The following table lists the examples in the tutorial and the tasks that you'll complete as you go through each one.

Section	Tasks
Installing the Sample Application	Install the Classics sample application.
Example 1: *Starting TestFactory and Instrumenting the Application-Under-Test*	Start TestFactory, enter project information in the New Project Wizard, and instrument the Classics source files.
Example 2: *Mapping the Application-Under-Test*	Incrementally map the Classics application. Set up interaction objects to map beyond a login dialog box, and to map a specific path in the user interface. Map an unmapped control. Exclude a control from mapping. Use the Find Objects window to locate objects in the application map. Create TestFactory reports.
Example 3: *Automatically Generating Scripts*	Edit data entry styles for testing input controls. Set up and run Pilots to test functional areas of Classics, and then examine the Pilot run results.
Example 4: *Mapping a Secondary Application*	Map a secondary application that the main application-under-test loads and executes.
Example 5: *Creating and Running a Test Suite*	Create a Test Suite, run it on your local machine, and examine the run results.
Example 6: *Using the AutoPilot to Run Multiple Pilots, Scripts, and Test Suites*	Add Pilots, scripts, and Test Suites to the AutoPilot window, and then run the queued test objects on your local machine.
Example 7: *Running Tests on a Test Lab Machine*	Start and set up TestAccelerator on a Test Lab machine. Set up TestFactory to use Test Lab machines, and then run a Pilot, a Test Suite, and test objects listed in the AutoPilot on a Test Lab machine.

Because software quality testing is an iterative, cyclical process, the tasks described in these examples represent ongoing efforts that you will perform again and again as your application develops and as new builds become available. As you work with your own application, you'll find that you return to earlier tasks to remap a portion of the application, fine-tune selected regions of the application map, reconfigure and rerun TestFactory reports, and run Pilots to generate new scripts.

In addition to this tutorial, we encourage you to take advantage of the TestFactory **Project Assistant**. The Project Assistant opens after you start TestFactory and enter project information in the New Project Wizard. You can use it at any time to quickly access useful information about basic TestFactory tasks.

About the Sample Application

The sample application that you'll use with this tutorial is called *Classics Online*. It was developed using Microsoft Visual Basic 6.0. Classics Online is designed to let customers of a music retail chain store browse through an online catalog and place orders, and to let store managers track inventory. When the Classics Online application is completed, tested, and successfully deployed, it will provide automated order entry and fulfillment capabilities for the retail chain.

Classics Online is a work in progress. It consists of the main Classics application, and a secondary application called *Inventory*, which the main application loads and executes. Three builds of Classics (A, B, and C) and two builds of Inventory (B and C) are included in the Classics Online folder, which is copied to your system when you install Rational Suite TestStudio. The examples in this tutorial use the ClassicsC build of the main application and the InventoryC build of the secondary application. You'll learn how to access the application files in the next section of this tutorial.

‣ ‣ ‣ Preparing to Use the Tutorial

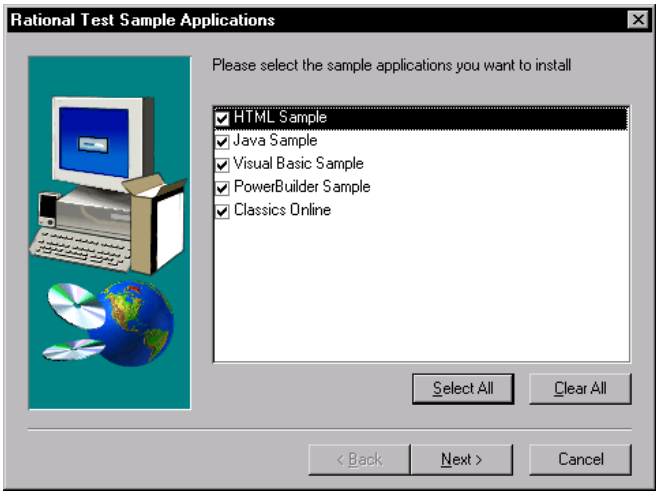
Objectives

- Install the Classics sample application.
- Run two utilities so that you can work with the sample application.

Installing the Classics Sample Application

To install Classics and its secondary application, Inventory:

1. Click **Start** → **Programs** → **Rational Suite TestStudio** → **Rational Test** → **Set Up Rational Test Samples**.



2. Clear the check boxes for sample applications that you do not want to install, and leave the **Classics Online** check box selected.
3. Click **Next**.
4. To complete the installation, click **Finish**.

After you install Classics, you can find the application files in the following directory:

\\Program Files\Rational\Rational Test 7\Sample Applications\
Classics Online\classics source (or \inventory source)

Making Post-Installation Adjustments

After you install Classics, and before you try to use the application, you must run the *Vbcmpfix.exe* file to correct a minor Microsoft Visual Basic problem, and run the *Vbctrls.reg* file to update the system registry.

The following steps are necessary only for using the Classics sample application. You do not need to perform these steps before you work with your own applications.

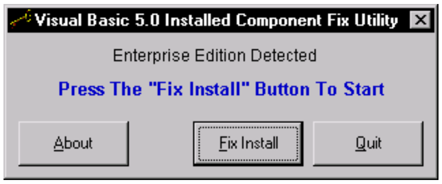
Running the Vbcmpfix.exe File

To run the *Vbcmpfix.exe* file:

1. Open the following directory:

   \\Program Files\Rational\Rational Test 7\Sample Applications\Classics Online

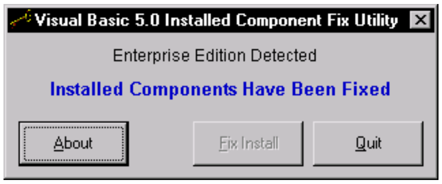
2. Double-click the *Vbcmpfix.exe* file.


Visual Basic 5.0 Installed Component Fix Utility

Enterprise Edition Detected

Press The "Fix Install" Button To Start

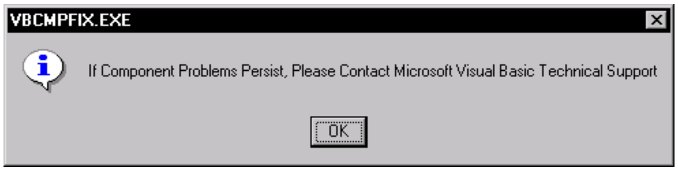
3. Click **Fix Install**.


Visual Basic 5.0 Installed Component Fix Utility

Enterprise Edition Detected

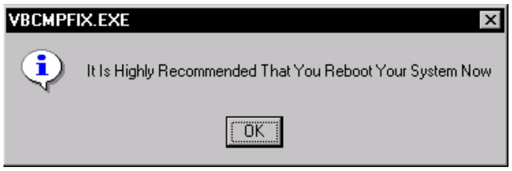
Installed Components Have Been Fixed

4. Click **Quit**.


VBCMPFIX.EXE

If Component Problems Persist, Please Contact Microsoft Visual Basic Technical Support

5. Click **OK**.


VBCMPFIX.EXE

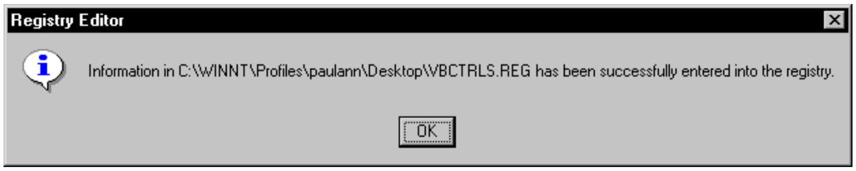
It Is Highly Recommended That You Reboot Your System Now

6. Click **OK**. Although a message advises you to reboot your system now, don’t do it just yet.

Locating and Running the Vbctrls.reg File

To locate and run the *vbctrls.reg* file:

1. Locate the *vbctrls.reg* file on the Visual Basic 6.0 CD-ROM or the Visual Studio 6.0 CD-ROM. For Visual Basic 6.0, you can find the *vbctrls.reg* file in the \Common\Tools\Vb\Controls folder on Disk 1. For Visual Studio 6.0, you can find the *vbctrls.reg* file in the same folder on Disk 3.
2. Copy the *vbctrls.reg* file to your machine.
3. Double-click the file name.


Registry Editor

Information in C:\WINNT\Profiles\paulann\Desktop\VBCTRLS.REG has been successfully entered into the registry.

4. On the Registry Editor message box, click **OK**.

Restarting the System

After you run the *Vbcmpfix.exe* and *vbctrls.reg* files, be sure to restart your system.

What’s Next

You're now ready to begin working with the Classics sample application in TestFactory. The remaining examples in this tutorial take you through the basic steps needed to complete the tasks described. For more complete information about the processes described, see the *Using Rational TestFactory* manual and TestFactory Help.

The next section shows you how to start TestFactory, specify project information in the New Project Wizard, and then instrument the Classics source code files.

▸▸▸ EXAMPLE 1

Starting TestFactory and Instrumenting the Application Source Code Files

Objectives

- Start TestFactory and log on to the Classics project in the Classics Repository.
- Use the New Project Wizard to supply TestFactory with basic information about the Classics application and project.
- Instrument the Classics source files.

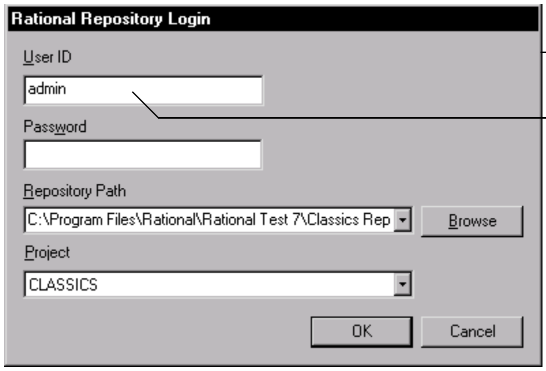
Scenario

This example shows you how to instrument an application so that TestFactory can calculate code coverage values for your scripts when you start testing. The following exercises show you how to start TestFactory, use the New Project Wizard to supply basic project information, and then instrument the source files for the Visual Basic sample application Classics.

Starting TestFactory and Opening the Classics Project

Before you start TestFactory, quit all applications open on the desktop.

1. To open the Rational Repository Login dialog box, click **Start** → **Programs** → **Rational Suite TestStudio** → **Rational TestFactory**.



Rational Repository Login

User ID: admin

Password:

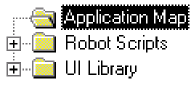
Repository Path: C:\Program Files\Rational\Rational Test 7\Classics Rep

Project: CLASSICS

2. In the **User ID** box, type **admin**.
3. Leave the **Password** box empty.
4. In the **Repository Path** list, click **C:\Program Files\Rational\Rational Test 7\ClassicsRepository.**

   Alternatively, browse to the path where you installed Rational Suite TestStudio.
5. In the **Project** list, click **CLASSICS**.
6. Click **OK**.

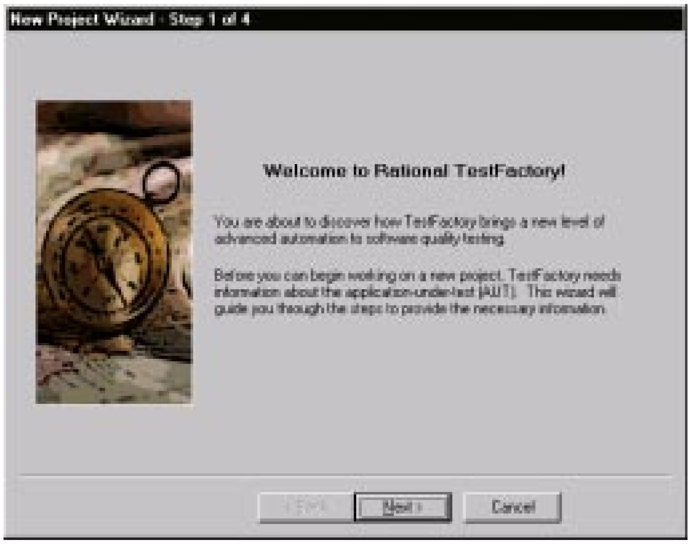
TestFactory initializes the CLASSICS project by building a UI library of object classes, and placing the Application Map folder, the Robot Scripts folder, and the UI Library folder in the left pane of the TestFactory window.



Entering Information in the New Project Wizard

After you open a project in TestFactory for the first time, the **New Project Wizard** prompts you to provide information about the application and project.

1. Read the information provided on step 1 of the New Project Wizard, and then click **Next**.



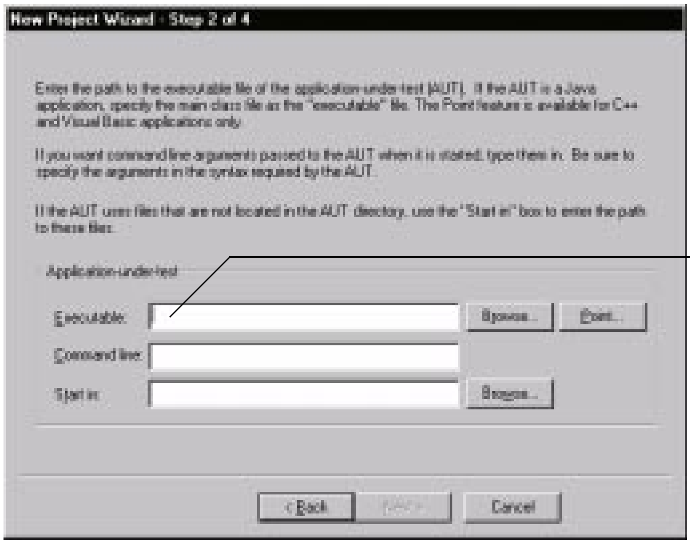
Welcome to Rational TestFactory!

You are about to discover how TestFactory brings a new level of advanced automation to software quality testing.

Before you can begin working on a new project, TestFactory needs information about the application-under-test (AUT). This wizard will guide you through the steps to provide the necessary information.

2. In the **Executable** box on step 2, type or browse to the following path:

   C:\Program Files\Rational\Rational Test 7\Sample Applications \Classics Online\ClassicsC.exe



New Project Wizard - Step 2 of 4

Enter the path to the executable file of the application-under-test (AUT). If the AUT is a Java application, specify the main class file as the "executable" file. The Point feature is available for C++ and Visual Basic applications only.

If you want command line arguments passed to the AUT when it is started, type them in. Be sure to specify the arguments in the syntax required by the AUT.

If the AUT uses files that are not located in the AUT directory, use the "Start in" box to enter the path to these files.

Application-under-test

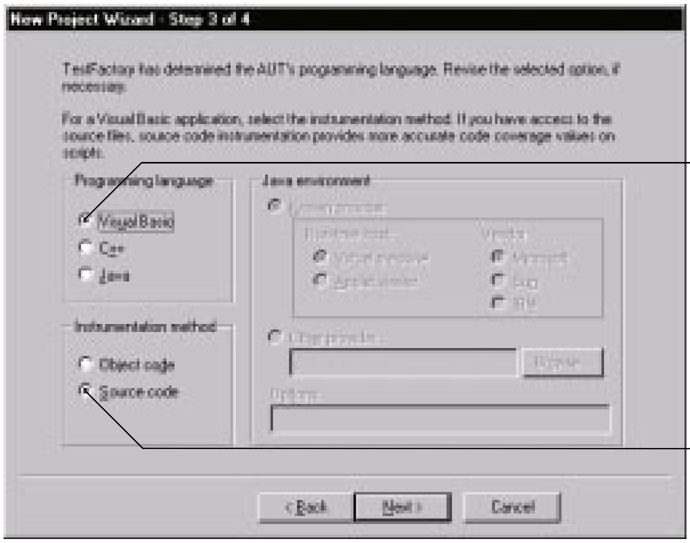
Executable:

Command line:

Start in:

Enter the path to the ClassicsC.exe file here.

3. Click **Next.**



New Project Wizard - Step 3 of 4

TestFactory has determined the AUT's programming language. Revise the selected option, if necessary.

For a Visual Basic application, select the instrumentation method. If you have access to the source files, source code instrumentation provides more accurate code coverage values on scripts.

Programming language: [✓ Visual Basic] C++ Java

Instrumentation method: Object code [✓ Source code]

Leave ***Visual Basic*** *selected.*

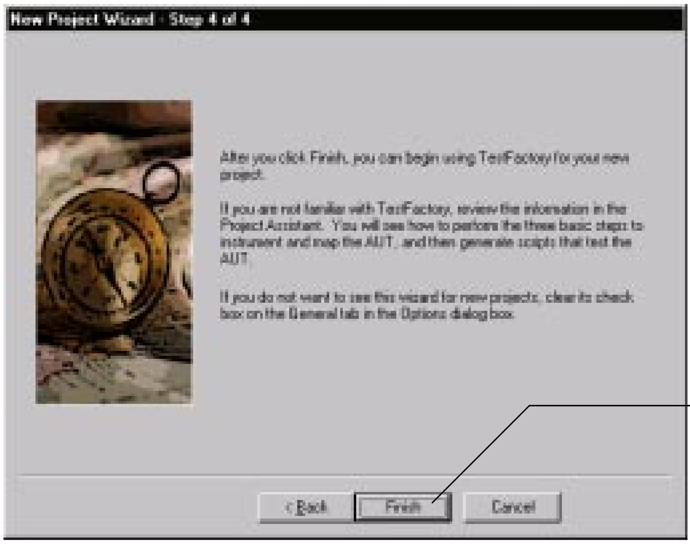
Leave ***Source code*** *selected.*

TestFactory uses the executable file name you provided to determine that ClassicsC is written in the Visual Basic programming language.

4. Under **Programming language**, leave **Visual Basic** selected.

   Later in this example, you'll instrument the Classics application. Instrumentation gives TestFactory the information it needs to determine how well your scripts exercise the application source code during testing. You can instrument a Visual Basic application by using the **object code** method or the **source code** method. For this exercise, you'll use the source code method.

5. Under **Instrumentation**, leave **Source code** selected.
6. Click **Next**.


New Project Wizard - Step 4 of 4
< Back
Finish
Cancel
Click here to close the wizard.

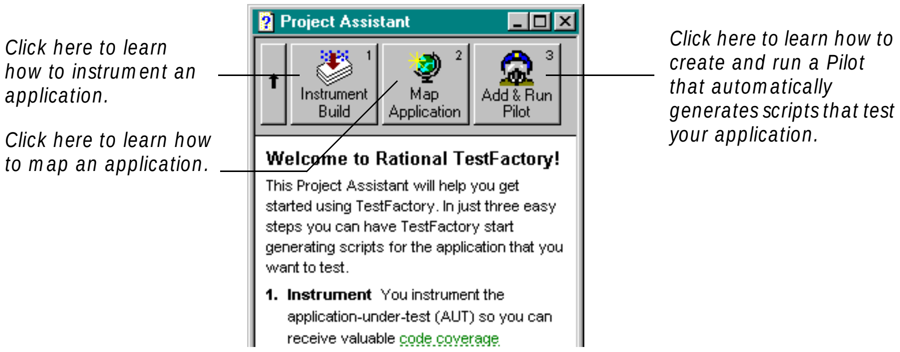
After you click Finish, you can begin using TestFactory for your new project.

If you are not familiar with TestFactory, review the information in the Project Assistant. You will see how to perform the three basic steps to instrument and map the AUT, and then generate scripts that test the AUT.

If you do not want to see this wizard for new projects, clear its check box on the General tab in the Options dialog box.

7. To close the wizard, click **Finish**.

After the New Project Wizard closes, the **Project Assistant** opens. You can use the Project Assistant to quickly get information about instrumenting the application, mapping the user interface, and running a Pilot.


Project Assistant
Instrument Build
1
Map Application
2
Add & Run Pilot
3
Click here to learn how to instrument an application.
Click here to learn how to map an application.
Click here to learn how to create and run a Pilot that automatically generates scripts that test your application.

Welcome to Rational TestFactory!

This Project Assistant will help you get started using TestFactory. In just three easy steps you can have TestFactory start generating scripts for the application that you want to test.

1. **Instrument** You instrument the application-under-test (AUT) so you can receive valuable code coverage

For now, close the Project Assistant.

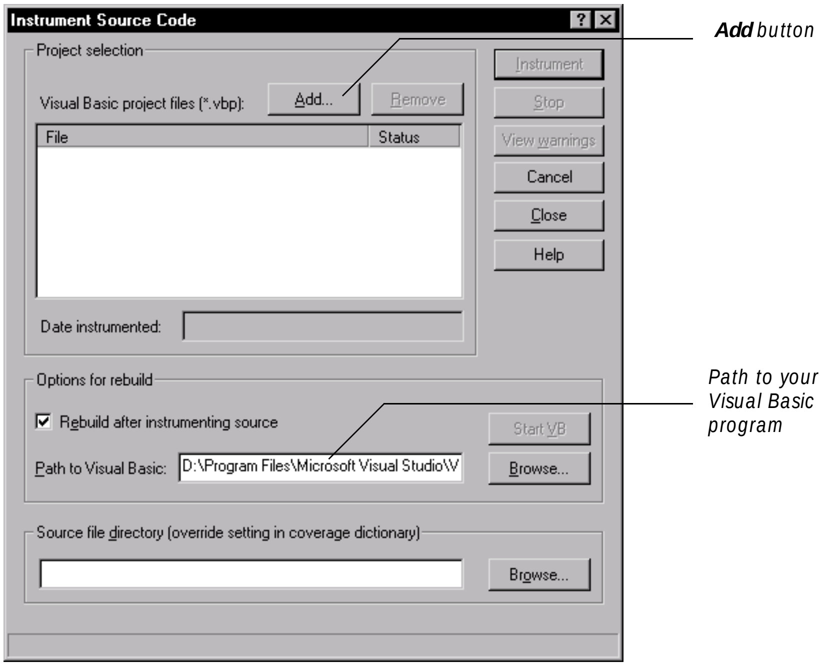
To open the Project Assistant later, just click **Help** → **Project Assistant**.

Instrumenting the Sample Application

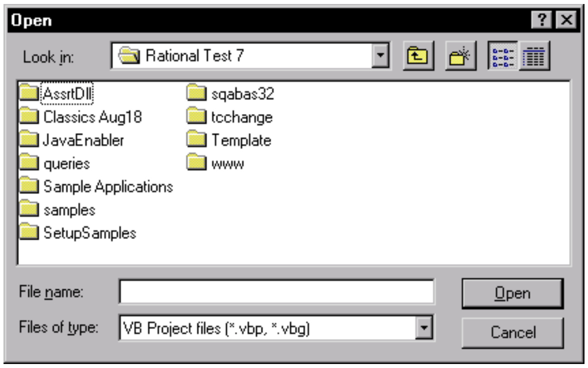
On step 3 of the New Project Wizard, you selected the **source code** method for instrumenting the Classics application. The following steps show you how to instrument the source files. Later, when you test the application, TestFactory Pilots will use the instrumented source files to calculate code coverage—the percentage of all source code that your scripts cover.

To instrument the Classics source files:

1. On the TestFactory toolbar, click **Instrument**.


Instrument Source Code
Project selection
Visual Basic project files (*.vbp):
Add...
Remove
File
Status
Date instrumented:
Options for rebuild
Rebuild after instrumenting source
Path to Visual Basic:
D:\Program Files\Microsoft Visual Studio\V
Start VB
Browse...
Source file directory (override setting in coverage dictionary)
Browse...
Instrument
Stop
View warnings
Cancel
Close
Help
Add button
Path to your Visual Basic program

2. Under **Project selection**, click **Add**.



3. In the Open dialog box, browse for and open the ClassC.vbp file. You can find this file in the following directory:

   \\Rational Test 7\Sample Applications\Classics Online\classics source\ClassC.vbp

4. Under **Options for rebuild**, leave the **Rebuild after instrumenting source** check box selected.
5. If the **Path to Visual Basic** value is incorrect, enter the correct path of the executable file for your Visual Basic program.
6. To instrument the source files and rebuild the ClassicsC.exe file, click **Instrument**.
7. After instrumentation is completed and the executable file is rebuilt, click **Close**.

Summary

In this example, you started TestFactory, used the New Project Wizard to provide TestFactory with basic project information, and then instrumented the Classics source files. TestFactory can now use the instrumented source files to calculate code coverage values for the scripts that you create later as you run Pilots to test the application.

What's Next

In the next example, you'll learn how to create and begin to develop a detailed application map of the Classics user interface.

▸ ▸ ▸ E X A M P L E 2

Mapping the Sample Application

Objectives

- Start the Classics application from TestFactory and explore its user interface.
- Map Classics to single-level depth.
- Assign a style with a required string case to an object in the application map so that TestFactory can map beyond a login dialog box.
- Map a region of the Classics application to full depth.
- Use an interaction object to map an alternative path in a functional area of the sample application.
- Map an unmapped image control.
- Exclude a control from mapping and testing.
- Map Classics to full depth.
- Use the Find Objects window to locate an object in the application map.
- Create TestFactory reports on objects in the application map.

TestFactory uses TrueMap technology to automatically gather detailed information about the user interface and its navigational pathways. TestFactory completely exercises an application and builds a comprehensive hierarchical application map. In TestFactory, a well-developed application map provides the elements for automatic script generation.

Scenario

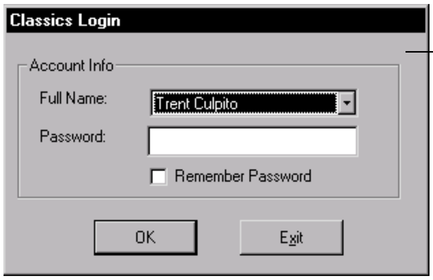
In this example, you'll learn how to map the Classics sample application in increments. Some of the exercises describe tasks such as creating a region object and excluding a control from mapping, that you might not need to perform when you work with your own application in TestFactory. These exercises are meant to demonstrate some of the methods you can use to fine-tune the application map.

Starting Classics from TestFactory

Before you begin mapping, start the Classics application from TestFactory and explore its user interface so that you can see what you'll be mapping.

To start and explore Classics from TestFactory:

1. On the Standard toolbar, click **StartAUT**.


Classics Login

Account Info

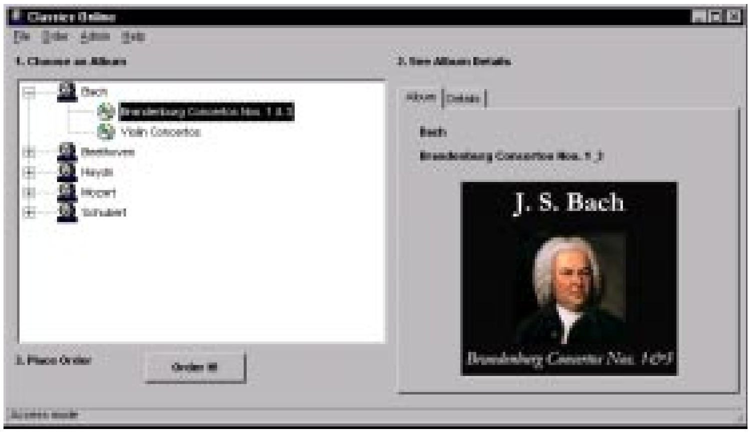
Full Name: Trent Culpito

Password:

☐ Remember Password

Classics Login dialog box

2. In the **Full Name** box under **Account Info**, type **Database Admin**.
3. In the **Password** box, type **5555**.
4. Click **OK**.



5. Try out the various options in the Classics Online window to see where they lead you and how the application functions.

   After you finish exploring the Classics user interface, and before you start mapping, be sure to quit Classics.

6. To quit Classics, click **File** → **Exit**.

Incremental Mapping

You can use TestFactory's Application Mapper to map an application to **full depth** or **single-level depth**. In full-depth mapping, the Application Mapper exercises and maps every control it encounters in the application until it has taken every path it can find. In single-level depth mapping, the Application Mapper exercises the first control it encounters and then maps the controls that this interaction exposes without exercising them. In this example, you'll map regions of the Classics user interface in increments by alternately using single-level and full-depth mapping.

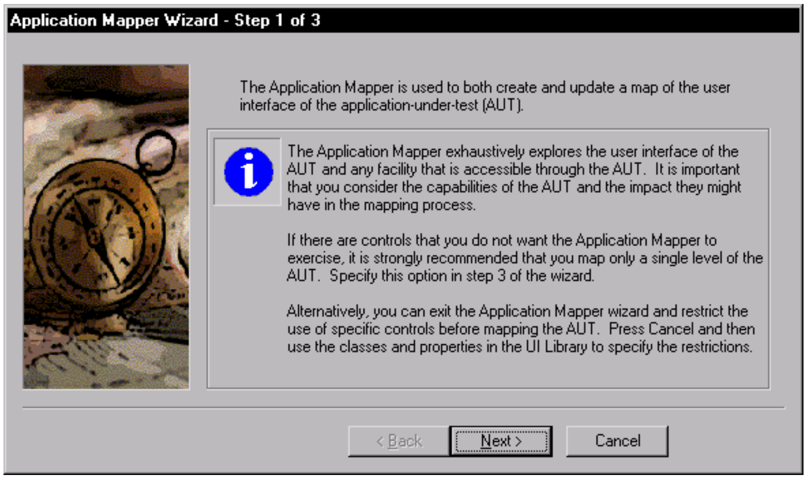
Mapping Classics to Single-Level Depth

When you map the application the first time, the Application Mapper can access and map the Classics Login dialog box, but it cannot exercise the controls in it. Without the correct login information to pass to the application, the Application Mapper cannot expose any more of the user interface. In this exercise, you'll start by mapping Classics to single-level depth. Later, you'll specify login information to expose more of the user interface for TestFactory to map.

To map Classics to single-level depth:

1. To start the Application Mapper Wizard, click **Application Mapper** on the Standard toolbar.

Step 1 of the Application Mapper Wizard


Application Mapper Wizard - Step 1 of 3

The Application Mapper is used to both create and update a map of the user interface of the application-under-test (AUT).

The Application Mapper exhaustively explores the user interface of the AUT and any facility that is accessible through the AUT. It is important that you consider the capabilities of the AUT and the impact they might have in the mapping process.

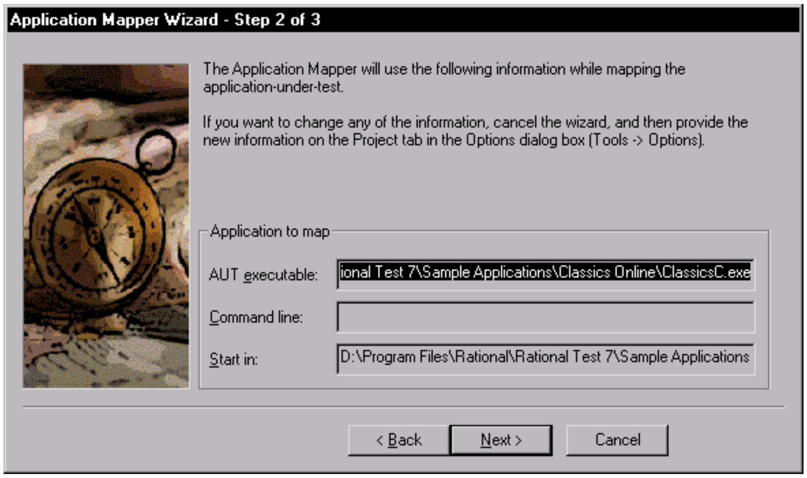
If there are controls that you do not want the Application Mapper to exercise, it is strongly recommended that you map only a single level of the AUT. Specify this option in step 3 of the wizard.

Alternatively, you can exit the Application Mapper wizard and restrict the use of specific controls before mapping the AUT. Press Cancel and then use the classes and properties in the UI Library to specify the restrictions.

2. Click **Next**.

Step 2 of the Application Mapper Wizard

Because you specified the Classics executable file in the New Project Wizard, TestFactory knows where to find the file for mapping.


Application Mapper Wizard - Step 2 of 3

The Application Mapper will use the following information while mapping the application-under-test.

If you want to change any of the information, cancel the wizard, and then provide the new information on the Project tab in the Options dialog box (Tools -> Options).

Application to map

AUT executable: ional Test 7\Sample Applications\Classics Online\ClassicsC.exe

Command line:

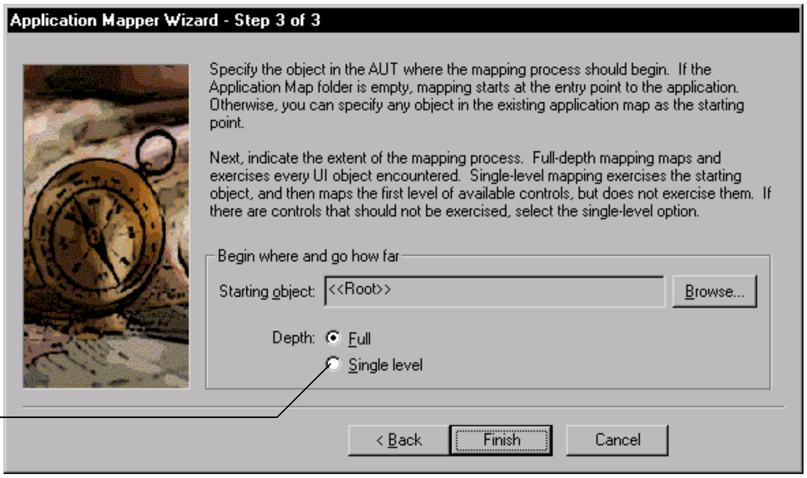
Start in: D:\Program Files\Rational\Rational Test 7\Sample Applications

3. Click **Next**.

Step 3 of the Application Mapper Wizard

The first time you map an application, the default "<<Root>>" is the appropriate starting point. After you map the AUT the first time, you can choose any UI object in the map as the starting object.

Click here to specify single-level depth mapping.


Application Mapper Wizard - Step 3 of 3

Specify the object in the AUT where the mapping process should begin. If the Application Map folder is empty, mapping starts at the entry point to the application. Otherwise, you can specify any object in the existing application map as the starting point.

Next, indicate the extent of the mapping process. Full-depth mapping maps and exercises every UI object encountered. Single-level mapping exercises the starting object, and then maps the first level of available controls, but does not exercise them. If there are controls that should not be exercised, select the single-level option.

Begin where and go how far

Starting object: <<Root>>

Depth: [✓ Full] Single level

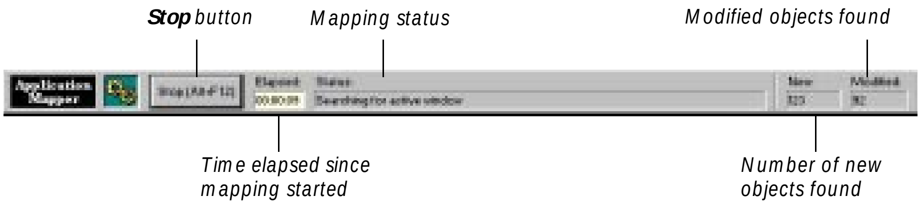
4. Under **Begin where and go how far**, click **Single level**.
5. To close the wizard and start mapping, click **Finish**.

As mapping begins, you'll see the following events on the screen:

- The TestFactory window closes and the Application Mapper progress bar opens at the bottom of the screen.
- A mask covers the screen and displays the *Running Application Mapper* message.

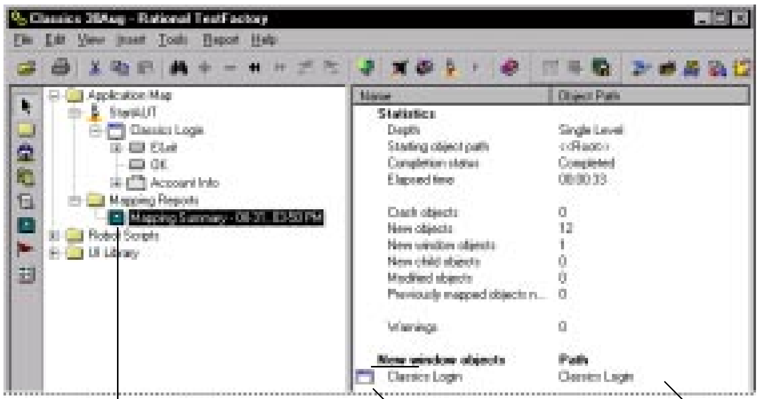
- TestFactory starts Robot and then minimizes the Robot window.
- TestFactory starts Classics, and then exercises its controls.

During the mapping process, the Application Mapper progress bar displays information about mapping activity.


Stop button
Mapping status
Modified objects found
Time elapsed since mapping started
Number of new objects found

NOTE: Do not try to use the computer while TestFactory is mapping. Because TestFactory uses the pointer to exercise controls in the application, it is especially important that you not move the mouse during mapping.

This first, single-level mapping session takes just a short time. After the session is completed, the restored TestFactory window displays the new application map in the left pane and the contents of the **Mapping Summary** report in the right pane.



Name	Object Path
Statistics	
Depth	Single Level
Starting object path	<<Root>>
Completion status	Completed
Elapsed time	00:00:33
Crash objects	0
New objects	12
New window objects	1
New child objects	0
Modified objects	0
Previously mapped objects n...	0
Warnings	0
New window objects	**Path**
Classics Login	Classics Login

Mapping Summary report object
Double-click here to select the Classics Login object in the application map.
Mapping Summary report contents

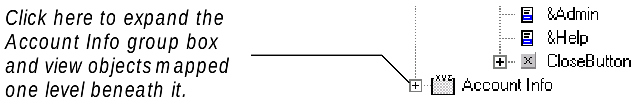
The Mapping Summary report can display the number of new objects mapped, previously-mapped controls not seen in the user interface, the starting object for the mapping session, and other information that you'll find useful in evaluating the new map and developing it further.

To jump to a mapped object listed in the Mapping Summary report, double-click the report item in the right pane.

Expanding the Application Map

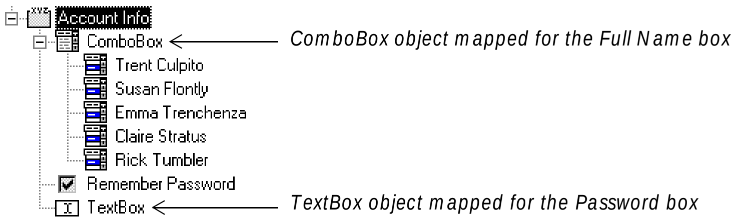
A plus (+) character next to an object in the application map indicates that child objects are mapped below it. To expand an object and see its child objects, do one of the following:

- To expand a branch of the map one level, click the plus (+) character next to an collapsed object.


Click here to expand the Account Info group box and view objects mapped one level beneath it.
&Admin
&Help
CloseButton
Account Info

- To fully expand a branch of the map and see all of the objects mapped beneath it:
  - Right-click the object, and then click **Expand All** on the shortcut menu.
  - Click the object, and then click **Expand** on the Standard toolbar.

For now, click the Account Info object in the application map, and then click **Expand** on the toolbar.



The ComboBox and TextBox objects mapped below Account Info represent the Full Name and Password boxes, respectively, in the Classics Login dialog box. To make these objects easier to identify in the application map, you'll rename them.

To rename the ComboBox and TextBox objects:

1. In the application map, click the ComboBox object.
2. Press F2, and then type **Full Name** in the active text box.
3. Next, click the TextBox object.
4. Press F2, and then type **Password** in the active text box.

Mapping Beyond the Classics Login Dialog Box

So far, you have mapped just the Classics Login dialog box and the controls that it contains. To map more of the Classics user interface, you must provide the Application Mapper with input to pass to the Full Name combo box and the Password text box. To supply this information, you'll assign data entry styles to the Full Name and Password objects in the application map, and then edit the styles so that they include required string cases.

Assigning a Data Entry Style to an Object

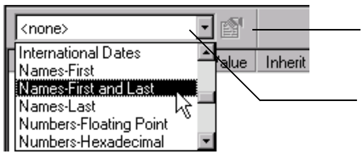
This exercise shows you how to:

- Assign data entry styles to input-type objects in the application map.
- Edit the data entry styles to include required string cases.

Assigning Data Entry Styles to Objects in the Application Map

To assign a data entry style to the Full Name object:

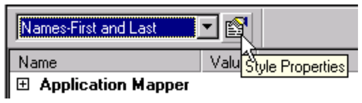
1. In the application map, click the Full Name combo box object.
2. In the Properties view in the top right pane, click the **Select Style** box on the Style toolbar.



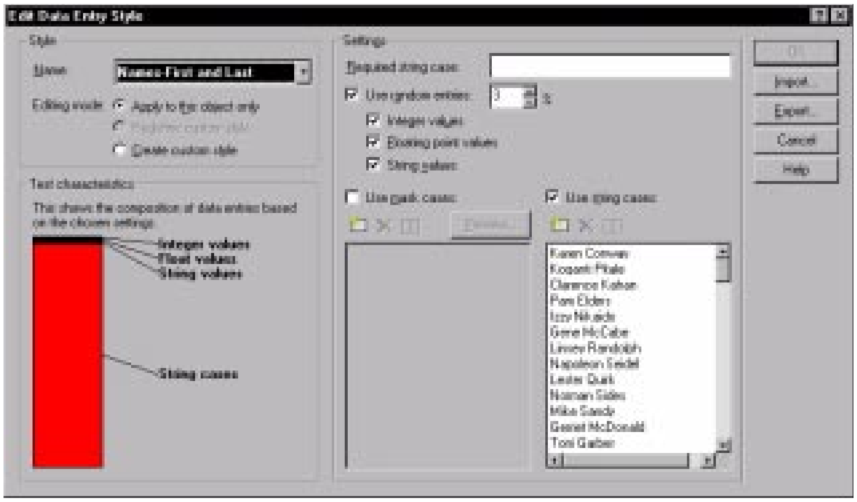
3. In the **Select Style** list, scroll down to and click **Names-First and Last**.

To edit the **Names-First and Last** style and specify a required string case:

1. To open the Edit Data Entry Styles dialog box, click **Style Properties** on the Style toolbar.



The Edit Data Entry Styles dialog box shows the data entry settings for the **Names-First and Last** style.



Edit Data Entry Style

Style

Name: Names-First and Last

Editing mode: Apply to this object only [✓] / Redefine custom style / Create custom style

Test characteristics

This shows the composition of data entries based on the chosen settings.

Settings

Required string case:

Use random entries: 3 % [✓]

Integer values [✓]

Floating point values [✓]

String values [✓]

Use mask cases: [ ]

Use string cases: [✓]

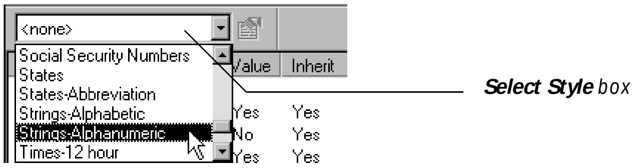
Karen Conway
Kopanti Plate
Clarence Kahan
Pam Elders
Izzy Nikaido
Gene McCabe
Linsey Randolph
Napoleon Seidel
Lester Dusk
Norman Sides
Mike Sandy
Garret McDonald
Toni Gaber

2. To specify a required string case for TestFactory to pass to the Full Name box during mapping and testing, under **Settings**, type **Database admin** in the **Required string case** box.

3. To save your setting and close the dialog box, click **OK**.

To assign a data entry style to the Password object:

1. In the application map, click the Password text box object.
2. In the Properties view in the top right pane, click the **Select Style** box on the Style toolbar.



3. In the **Select Style** list, scroll down to and click **Strings-Alphanumeric**.

To edit the **Strings-Alphanumeric** style and specify a required string case:

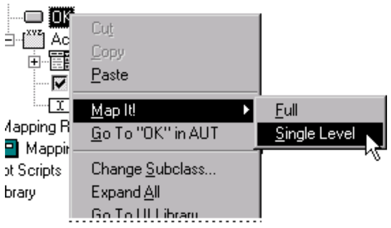
1. To open the Edit Data Entry Styles dialog box, click **Style Properties** on the Style toolbar.

   The Edit Data Entry Styles dialog box shows the data entry settings for the **Strings-Alphanumeric** style.
2. To specify a required string case for TestFactory to pass to the Password box during mapping and testing, under **Settings**, type **5555** in the **Required string case** box.
3. To save your setting and close the dialog box, click **OK**.

Now that you've supplied the Application Mapper with the information it needs to pass to the Classics Login controls, you can map the next level of controls.

To map the next level of controls beyond the Classics Login dialog box:

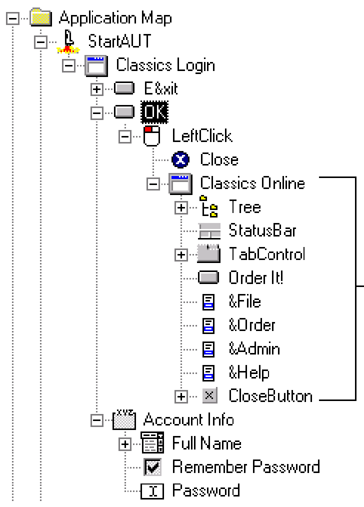
▶ In the application map, right-click the OK button object, point to **Map It!**, and then click **Single Level** on the shortcut menu.



This mapping session takes longer than the first one because TestFactory has access to and can map more controls.

To view the results of the mapping session after mapping is completed:

1. To jump to the OK button object in the application map, click **Previous Object** on the toolbar.
2. Click the plus character (+) next to the OK button object, click the plus character next to the LeftClick object, and then click the plus character next to the Classics Online window object.



The application map now includes objects that represent the first level of controls in the Classics Online window.

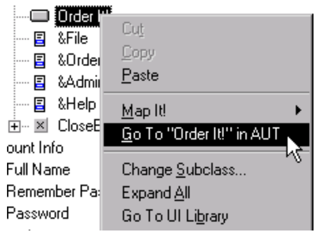
Comparing the Application Map to the Sample Application

After a mapping session is completed, you can start Classics from TestFactory and compare your mapping results against it. At the beginning of this example, you used the **StartAUT** toolbar button to start Classics so that you could explore its user interface. After mapping, you can use the **Go To "*Control*" in AUT** shortcut menu option to go to a specific control in the AUT.

Going to a Selected Control from the Application Map

The **Go To "*Control*" in AUT** option lets you select an object in the map and have TestFactory drive directly to the corresponding control in the application. To start Classics and navigate to the **Order It!** button in Classics:

- In the application map, right-click the Order It! button object, and then click **Go To "Order It!" in AUT**.



On your screen, you'll see:

- The TestFactory window closes and the Go To Control progress bar opens at the bottom of the screen.
- A mask covers the screen and displays the **Go To Control** message.
- TestFactory starts Robot and then minimizes the Robot window.
- TestFactory starts Classics, and then exercises the user interface until it exposes the **Order It!** button.

Once TestFactory locates the **Order It!** button, the Go To Control progress bar closes and the TestFactory window is restored. The Classics application stays open on top of the TestFactory window.

Take a few minutes to examine the application map hierarchy and check the accuracy of its structure by exploring the navigational paths you've mapped so far. Click **Order It!** and see where it takes you in the application. You'll map the Order It! button next.

NOTE: Remember to quit Classics before you start the next mapping session.

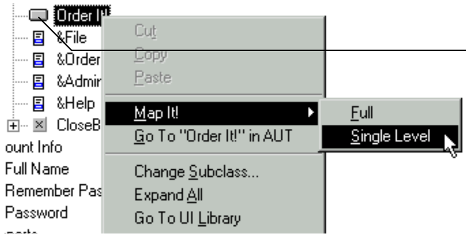
Mapping a Part of Classics to Full Depth

Now that you've mapped the first level of controls in the Classics user interface, you can fully map a functional area of the application. The Order It! command button in Classics opens the Make An Order dialog box, which a user can fill out to order albums. To correctly place an order, a user must enter quantity and payment information.

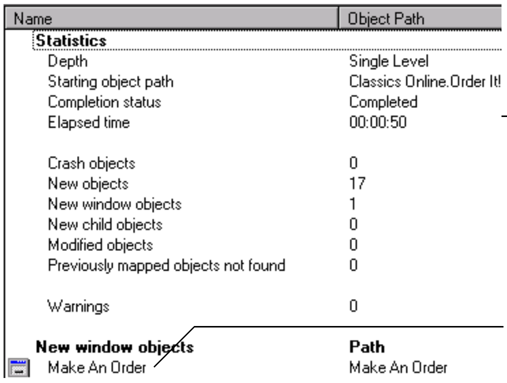
To map the Make An Order dialog box and its controls, you'll start by mapping the Order It! button to single-level depth.

To map the Order It! button to single-level depth:

- ▶ In the application map, right-click the Order It! button object, point to **Map It!**, and then click **Single Level** on the shortcut menu.



After mapping is completed, the Mapping Reports folder is selected in the left pane. The right pane displays the contents of the Mapping Summary report.



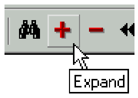
Name	Object Path
Statistics	
Depth	Single Level
Starting object path	Classics Online.Order It!
Completion status	Completed
Elapsed time	00:00:50
Crash objects	0
New objects	17
New window objects	1
New child objects	0
Modified objects	0
Previously mapped objects not found	0
Warnings	0
New window objects	**Path**
Make An Order	Make An Order

After a mapping session is completed, TestFactory displays the contents of the Mapping Summary report in the right pane.
To go to the Make An Order window object in the application map, double-click here.

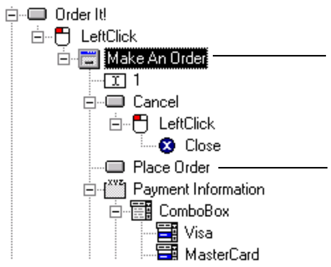
To jump to the new Make An Order window object in the application map:

- ▶ In the right pane, under **New window objects**, double-click **Make An Order**.

To see the child objects mapped below the Make An Order window object:



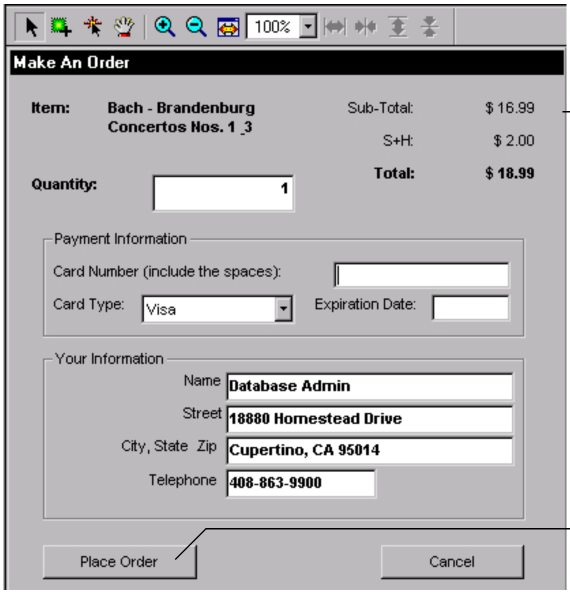
▸ Click **Expand** on the Standard toolbar.



TestFactory has mapped the Make An Order window and the first level of controls (child objects) in the window.

Place Order button object

The Image view in the lower right pane displays a bitmap image of the Make An Order window.



Make An Order

Item: Bach - Brandenburg Concertos Nos. 1_3

Sub-Total: $ 16.99

S+H: $ 2.00

Total: $ 18.99

Quantity: 1

Payment Information

Card Number (include the spaces):

Card Type: Visa

Expiration Date:

Your Information

Name: Database Admin

Street: 18880 Homestead Drive

City, State Zip: Cupertino, CA 95014

Telephone: 408-863-9900

Place Order

Cancel

The Image view in the lower right pane displays a bitmap image of the Make An Order window.

***Place Order** button*

The Place Order button leads to three possible paths in Classics. If a user correctly enters quantity and payment information, and then clicks **Place Order**, Classics displays the Order Confirmation message box. If the user clicks **Place Order** without entering the correct payment information first, Classics displays the Incomplete Order error message. If the user types a zero in the **Quantity** box, and then clicks **Place Order**, Classics displays the Invalid Quantity Ordered error message. In the following exercises, you'll map all three of these paths so that you can test them later.

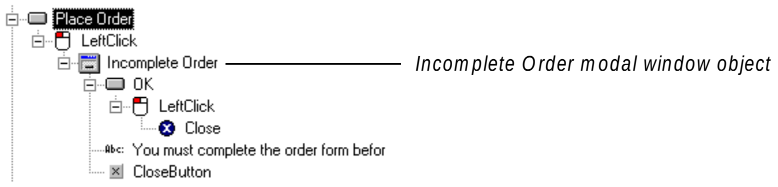
In this exercise, you'll map the path that includes the Incomplete Order message box.

To map the Incomplete Order message box:

▸ In the application map, right-click the Place Order button object, point to **Map It!**, and then click **Full** on the shortcut menu.

To view the results after mapping is completed:

▸ On the toolbar, click **Previous Object**, and then click **Expand**.


```
Place Order
  LeftClick
    Incomplete Order
      OK
        LeftClick
          Close
      You must complete the order form befor
      CloseButton
```

The application map now contains UI objects that represent the Incomplete Order message box and its child controls.

Next, you'll insert and set up interaction objects to map the paths that include the Order Confirmation message box and the Invalid Quantity Ordered error message box.

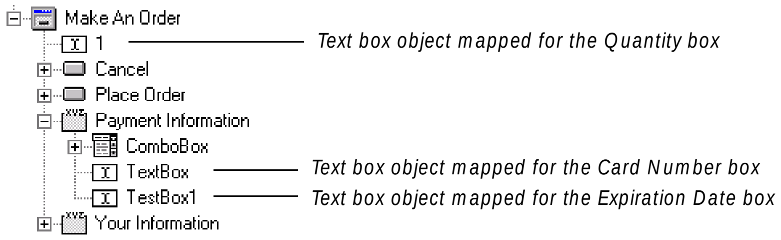
Mapping Alternative Paths in an Application

An **interaction object** lets you guide the Application Mapper through a path in the application that it cannot reach automatically. You can set up interaction objects to guide TestFactory through all of the available paths in a functional area of the application. If you don't use interaction objects, you can only map one of the paths available to users.

In the last exercise, you mapped the path to the Incomplete Order message box in Classics. In this exercise, you'll map two alternative paths — one that includes the Order Confirmation message box, and another that includes the Invalid Quantity Ordered message box.

To successfully order an album in Classics, a user must type text in the **Quantity**, **Card Number**, and **Expiration Date** boxes, and then click **Place Order**. To map the Confirmation Order message box, you'll set up an interaction object that includes components for these controls.

Before you insert the interaction object, rename the objects mapped for the text box controls to make them easier to identify in the application map.


Make An Order
1
Cancel
Place Order
Payment Information
ComboBox
TextBox
TestBox1
Your Information
Text box object mapped for the Quantity box
Text box object mapped for the Card Number box
Text box object mapped for the Expiration Date box

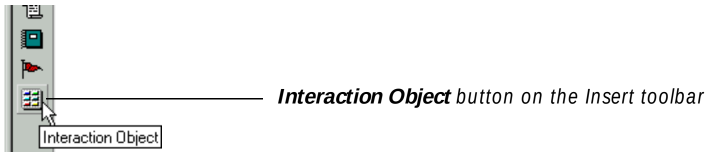
To rename three of the text box objects in the application map:

1. Click the 1 object.
2. Press F2, and then type **Quantity** in the active edit box.
3. Expand the Payment Information object in the application map.
4. Click the TextBox object.
5. Press F2, and then type **Card Number** in the active edit box.
6. Click the TextBox1 object.
7. Press F2, and then type **Expiration Date** in the active edit box.

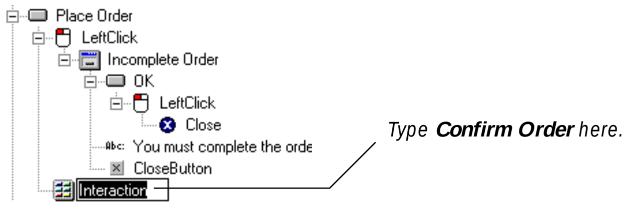
Inserting an Interaction Object from the Insert Toolbar

To insert an interaction object from the Insert toolbar:

1. On the left side of the TestFactory window, click **Interaction Object** on the Insert toolbar.

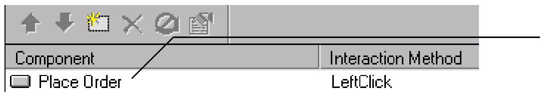


2. In the application map, click the Place Order button object.



3. To name the interaction object, type **Confirm Order** in the active text box.

The **Interaction Object view** in the upper right pane lists the Place Order button as a **component** of the Confirm Order interaction object.



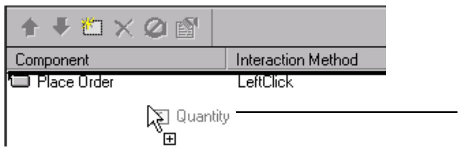
Component	Interaction Method
Place Order	LeftClick

Adding Interaction Object Components from the Application Map

In this exercise, you'll add the Quantity, Card Number, and Expiration Date components to the interaction object from the application map.

To add components from the application map:

1. In the application map, leave the focus on the Confirm Order interaction object, and then drag the Quantity object from the application map to the Interaction Object view.

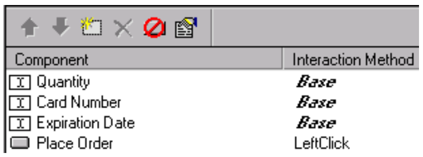

Drag the Quantity object from the application map to the Interaction Object view.
Quantity

Component	Interaction Method
Place Order	LeftClick

NOTE: Be sure that you *drag* the Quantity object, and that you don't click it. Otherwise, the Properties view replaces the Interaction Object view in the top right pane. If this happens, just click the Confirm Order interaction object and start again.

2. Drag the Card Number object, and then the Expiration Date object, from the application map to the Interaction Object view.

   The Confirm Order interaction object now contains the components that TestFactory needs to navigate to the Order Confirmation message box.

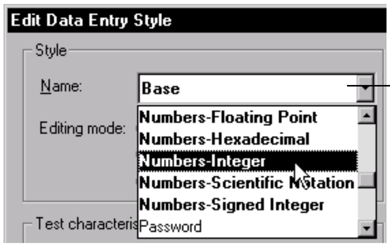


Component	Interaction Method
Quantity	*Base*
Card Number	*Base*
Expiration Date	*Base*
Place Order	LeftClick

After you add the text box components to the interaction object, TestFactory assigns the default *Base* style to each of them. Next, you'll assign new data entry styles to the components, and then specify required string cases to include in each.

To assign new styles to the interaction object components:

1. In the Interaction Object view, click the **Quantity** component, and then click **Style Properties** on the Interaction Object toolbar.



Edit Data Entry Style

Style

Name: Base

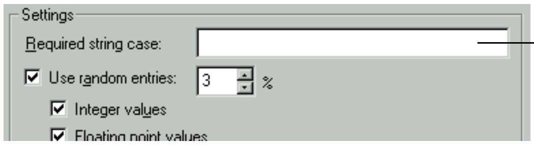
Editing mode:

Numbers-Floating Point
Numbers-Hexadecimal
Numbers-Integer
Numbers-Scientific Notation
Numbers-Signed Integer
Password

Test characteris

Style *box in the Edit Data Entry Styles dialog box*

2. Under **Style**, click the **Name** box, and then scroll down to and click **Numbers-Integer**.
3. Under **Settings**, type **2** in the **Required string case** box.



Settings

Required string case:

☑ Use random entries: 3 %

☑ Integer values

☑ Floating point values

Type ***2*** *here.*

4. To save your settings and close the Edit Data Entry Style dialog box, click **OK**.
5. Next, click the **Card Number** component in the Interaction Object view, and then click **Style Properties** on the Interaction Object toolbar.
6. Under **Style**, click the **Name** box, and then click **Credit Cards**.
7. Under **Settings**, type **5555 5555 5555 5555** in the **Required string case** box.
8. To save your settings and close the Edit Data Entry Style dialog box, click **OK**.
9. Now click the **Expiration Date** component, and then click **Style Properties** on the Interaction Object toolbar.
10. Under **Style**, click **Dates** in the **Name** list.
11. Under **Settings**, type **12/00** in the **Required string case** box.
12. To save your settings and close the Edit Data Entry Style dialog box, click **OK**.

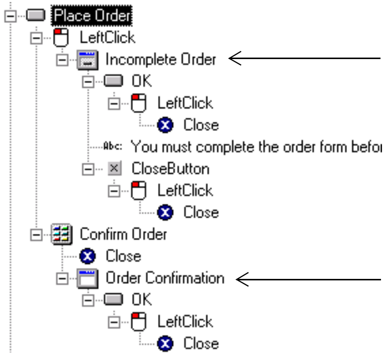
Now that you've specified styles and required string cases for the text boxes, you can map the path to the Order Confirmation message box.

To map the Order Confirmation message box:

▶ In the application map, right-click the Place Order object, point to **Map It!**, and then click **Full** on the shortcut menu.

To view the new objects in the application map after mapping is completed:

- On the Standard toolbar, click **Previous**, and then click **Expand**.


Place Order
LeftClick
Incomplete Order
OK
LeftClick
Close
You must complete the order form befo
CloseButton
LeftClick
Close
Confirm Order
Close
Order Confirmation
OK
LeftClick
Close

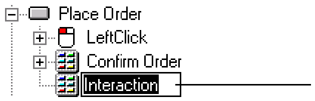
The application map now contains both the Incomplete Order message box and the Order Confirmation message box.

Next, you'll set up an interaction object that lets you map the third path in this part of Classics—the one that includes the Invalid Quantity Ordered message box. This time, you'll insert the interaction object using the **Insert** menu.

Inserting an Interaction Object Using the Insert Menu

To insert an interaction object using the **Insert** menu:

1. In the application map, click the Place Order button object.
2. Click **Insert** → **Interaction Object**.


Place Order
LeftClick
Confirm Order
Interaction

*Type **Invalid Quantity** here.*

3. To name the new interaction object, type **Invalid Quantity** in the active text box.

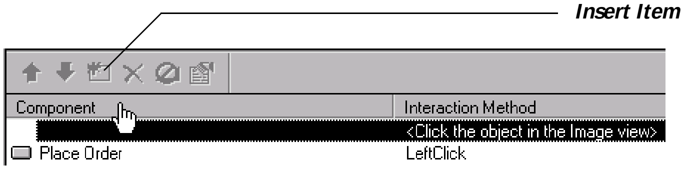
Inserting a Component from the Image View

Next, you'll add the Quantity component to the interaction object by selecting it in the Image view.

To add the Quantity component to the interaction object from the Image view:

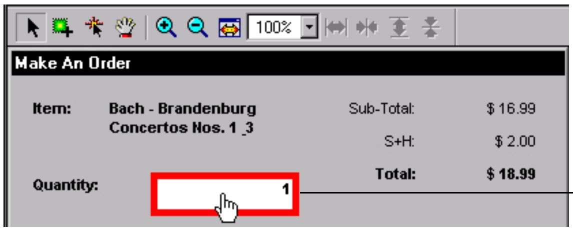
1. On the Interaction Object toolbar, click **Insert Item**.

Insert Item button



Component	Interaction Method
	<Click the object in the Image view>
Place Order	LeftClick

2. In the Image view in the bottom right pane, point to the image of the Quantity text box, and then, after TestFactory outlines the image in red, click the image.


Make An Order

Item: Bach - Brandenburg Concertos Nos. 1_3

Sub-Total: $ 16.99

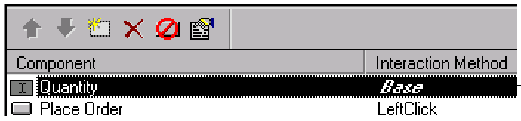
S+H: $ 2.00

Total: $ 18.99

Quantity: 1

In the Image view, point to and click the image of the Quantity text box.

The Interaction Object view now contains the Quantity component.

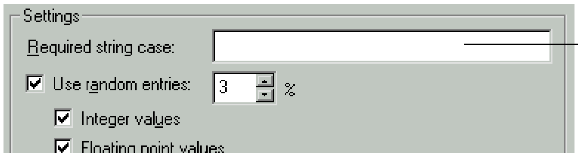


Component	Interaction Method
Quantity	Base
Place Order	LeftClick

Quantity component in the Interaction Object view

To specify a required string case value of zero for the Quantity component:

1. Leave the Quantity component selected in the Interaction Object view and click **Style Properties** on the Interaction Object toolbar.


Settings

Required string case:

☑ Use random entries: 3 %

☑ Integer values

☑ Floating point values

*Type **0** here.*

2. Under **Settings**, type **0** in the **Required string case** box.

3. To ensure that TestFactory uses only the required string case to exercise the Quantity text box during testing as well as during mapping, under **Settings**, clear the **Use random entries** check box.

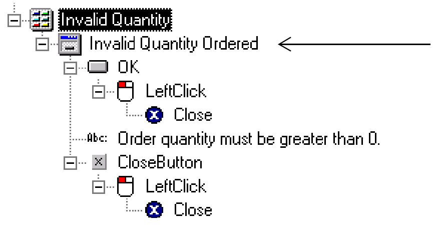
4. To save your settings and close the Edit Data Entry Style dialog box, click **OK**.

To map the path in Classics that includes the Invalid Quantity Ordered message box:

▸ Right-click the Invalid Quantity interaction object, point to **MapIt!**, and then click **Full** on the shortcut menu.

To view the results after the mapping session is completed:

▶ On the Standard toolbar, click **Previous Object**, and then click **Expand**.



```
Invalid Quantity
  Invalid Quantity Ordered
    OK
      LeftClick
        Close
    Order quantity must be greater than 0.
    CloseButton
      LeftClick
        Close
```

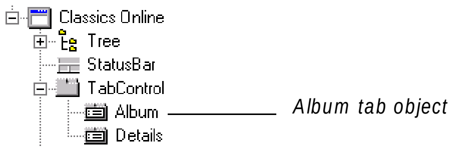
The application map now contains objects for the Invalid Quantity Ordered error message box and its child controls. You have successfully mapped the three paths available to a user from the Make An Order window.

Mapping an Unmapped Control

In some instances, the Application Mapper can fail to detect a control in the application. For example, the **Album** tab in Classics contains an image control that a user can double-click to order the album displayed in the image. TestFactory doesn't see this image control. If you map the Album tab to full depth, the Application Mapper creates no new objects. To get TestFactory to see and map the control, you have to create a **region object** for it first.

In this exercise, you'll insert and set up an image control that lets you map the Album tab to full depth.

To view the Album tab object in the application map, click the plus character (+) next to the Tab Control object.



```
Classics Online
  Tree
  StatusBar
  TabControl
    Album
    Details
```

To see the unmapped image control, click the Album tab object in the application map. The Image view in the lower right pane displays a bitmap image of the Album tab.

Creating a Region Object

To create a region object for the unmapped image control:

1. On the Image toolbar, click **Draw Region**.

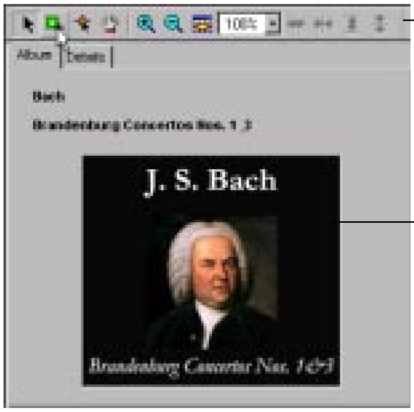
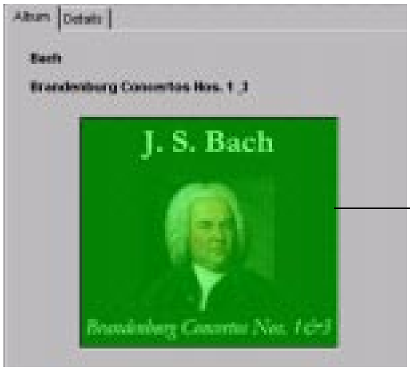

Album
Details
Bach
Brandenburg Concertos Nos. 1_3
J. S. Bach
Brandenburg Concertos Nos. 1&3

Image toolbar

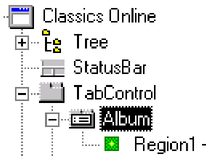
The Album tab containing the unmapped image control is displayed in the Image view.

2. To select the region, drag the mouse from one corner of the album cover image to the opposite corner.


Album
Details
Bach
Brandenburg Concertos Nos. 1_3
J. S. Bach
Brandenburg Concertos Nos. 1&3

A green tracker covers the region that you drew.

TestFactory places a green tracker on the area of the image you selected, and inserts the **Region1** object below the Album object in the application map.



New region object

3. To name the region object, click it, press F2, and then type **Album Cover** in the active text box.

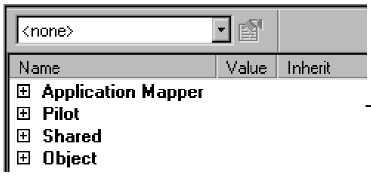
Setting the Action for Exercising a Region Object

To ensure that TestFactory uses the double-click action to exercise the image control on the Album tab, you'll set the **DoLeftDoubleClick** property for the region object.

To set the DoLeftDoubleClick property:

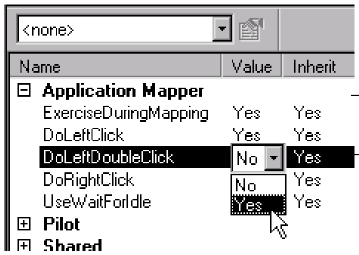
1. In the application map, click the Album Cover region object.

   TestFactory displays the object properties groups in the Properties view in the top right pane.



The Properties view in the top right pane

2. In the Properties view, click **Application Mapper** to view the application mapper properties for the region object.
3. In the **Value** list for the **DoLeftDoubleClick** property, select **Yes**.



Application Mapper properties in the Properties view

***DoLeftDoubleClick** property*

4. To save time during the mapping process, change the value set for the **DoLeftClick** property to **No**.

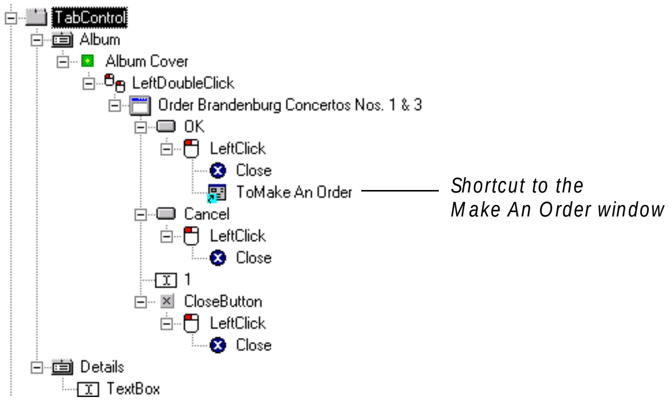
Exercising the album cover image exposes new parts of the Classics user interface. Before you can test this part of the application, you need to map the Album tab to full depth.

To map the Tab Control box, including the Album tab, to full depth:

▸ In the application map, right-click the Tab Control object, point to **MapIt!**, and then click **Full**.

To view the results after the mapping session is completed:

▶ On the Standard toolbar, click **Previous Object**, and then click **Expand**.



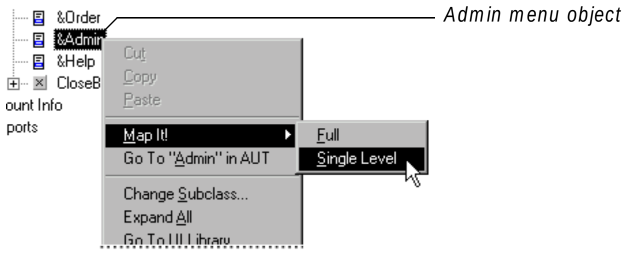
After it exercised the **OK** button, the Application Mapper encountered the Make An Order window. Because this window was already mapped, the Application Mapper placed a shortcut object that points to the window object originally mapped.

Excluding a Control from Mapping and Testing

The Classics **Admin** menu contains a command that we don't want to map or test. The **Restore Database** command resets all customer, product, and order data in the Classics database. In this exercise, you'll prevent TestFactory from exercising this control by excluding it from mapping and testing.

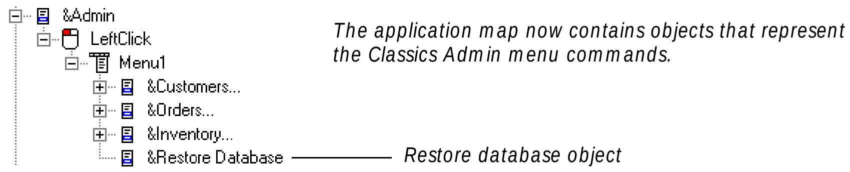
To expose the **Restore Database** command in the **Admin** menu:

▶ In the application map, right-click the Admin menu object, point to **Map It!**, and then click **Single Level** on the shortcut menu.



To see the Restore Database object after the mapping session is completed:

▶ Click **Previous Object**, and then click **Expand** on the toolbar.



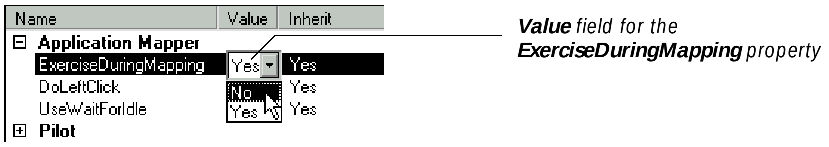
```
&Admin
  LeftClick
    Menu1
      &Customers...
      &Orders...
      &Inventory...
      &Restore Database
```

The application map now contains objects that represent the Classics Admin menu commands.

Restore database object

To exclude the **Restore Database** menu command from mapping and testing:

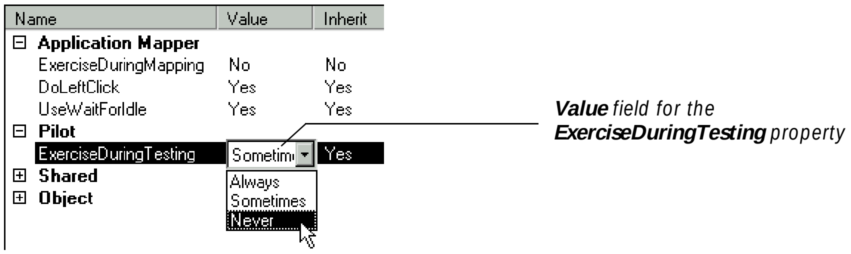
1. Click the Restore Database object in the application map.
2. In the Properties view, double-click the **Value** field for the **ExerciseDuringMapping** property, and then click **No**.



Name	Value	Inherit
⊟ **Application Mapper**		
ExerciseDuringMapping	Yes	Yes
DoLeftClick	No	Yes
UseWaitForIdle	Yes	Yes
⊞ **Pilot**		

Value field for the ExerciseDuringMapping property

3. To see the Pilot properties, click **Pilot** in the Properties view.



Name	Value	Inherit
⊟ **Application Mapper**		
ExerciseDuringMapping	No	No
DoLeftClick	Yes	Yes
UseWaitForIdle	Yes	Yes
⊟ **Pilot**		
ExerciseDuringTesting	Sometim	Yes
⊞ **Shared**	Always	
⊞ **Object**	Sometimes	
	Never	

Value field for the ExerciseDuringTesting property

4. To exclude the Restore Database control from testing, double-click the **Value** field for the **ExerciseDuringTesting** property, and then click **Never**.

When you fully map and test the Admin menu, TestFactory will ignore the **Restore Database** menu command.

Mapping Classics to Full Depth

Now that you have mapped alternative paths for the Order It feature, created a region object for the unmapped album image control, and excluded the Restore Database menu command from mapping, you can map the Classics application to full depth.

To map Classics to full depth:

▸ Right-click the Classics Online window object, point to **Map It!**, and then click **Full**.

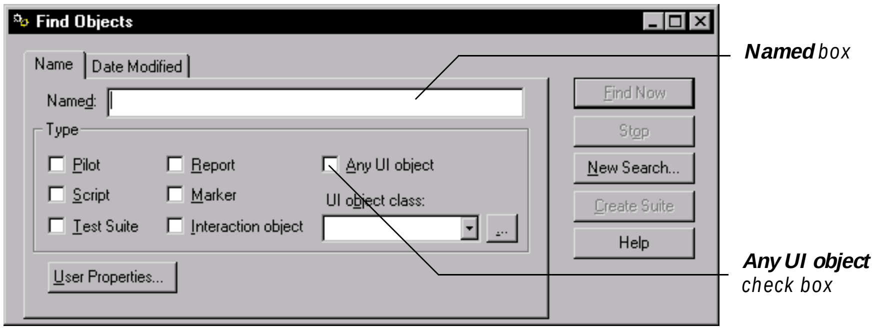
NOTE: This mapping session will take several minutes. While mapping is in progress, do not try to interact with your system.

Using the Find Objects Window to Locate Objects

If the application map that you create for your own application is complex, you'll find the **Find Objects** window useful for locating objects quickly. In this exercise, you'll use the Find Objects window to locate the View Existing Orders window object in the application map for Classics.

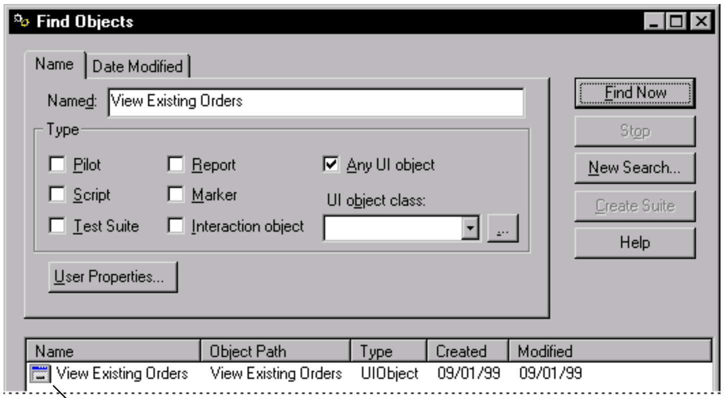
To find the View Existing Orders window object:

1. On the Standard toolbar, click **Find Objects**.



2. In the **Named** box, type **View Existing Orders**.
3. Under **Type**, select the **Any UI object** check box.

4. Click **Find Now**.



Named: View Existing Orders

Type

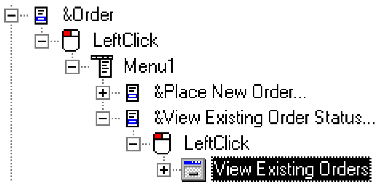
[ ] Pilot [ ] Report [✓] Any UI object
[ ] Script [ ] Marker
[ ] Test Suite [ ] Interaction object

UI object class:

Name	Object Path	Type	Created	Modified
View Existing Orders	View Existing Orders	UIObject	09/01/99	09/01/99

To select the View Existing Orders window in the application map, double-click here.

5. To go to the View Existing Orders object in the application map, double-click the **View Existing Orders** item in the **Name** column, and then close the Find Objects window.



The View Existing Orders window object is selected in the application map.

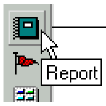
Creating TestFactory Reports

You can insert a TestFactory report object to configure and run a report on objects in the application map. Later, you can edit the report parameters, rerun the report, print it, and export it as a text file for editing in other applications.

Inserting a TestFactory Report Object

To insert a TestFactory report object in the application map:

1. On the Insert bar along the left side of the TestFactory window, click **Report**.



***Report** button on the Insert toolbar*

2. To select a location (a parent object) for the report and open the Edit Report Parameters dialog box, click an object in the application map.

   NOTE: The application map location you choose for a report does not affect the report contents.

Configuring TestFactory Reports

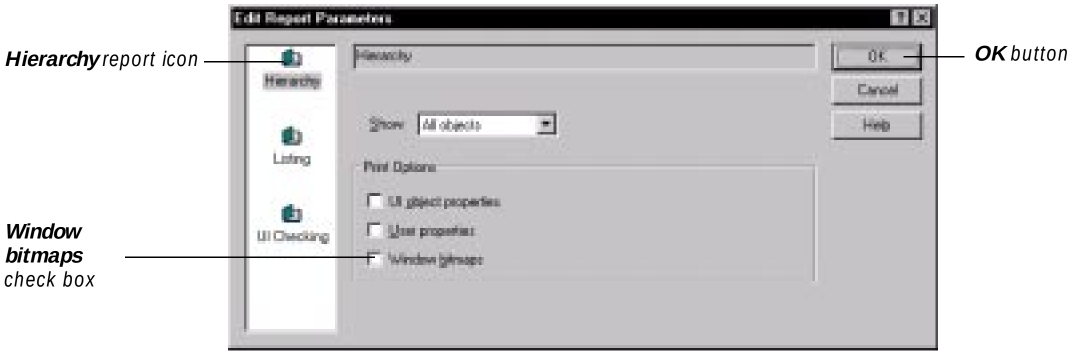
You can use the Edit Report Parameters dialog box to create three types of TestFactory reports; **Hierarchy** reports, **Listing** reports, and **UI Checking** reports. This exercise shows you how to configure and run a Hierarchy report and a UI Checking report.

Creating a Hierarchy Report

You can create and print a Hierarchy report to get a hard copy version of the application map hierarchy displayed in the left pane.

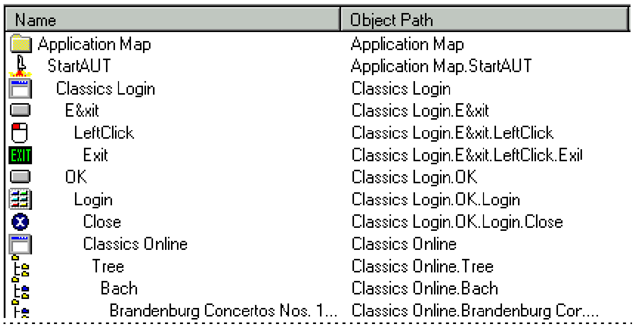
To create a Hierarchy report:

1. In the left portion of the Edit Report Parameters dialog box, leave the **Hierarchy** icon selected.



2. Under **Print Options**, select the **Window bitmaps** check box.
3. To run the report, click **OK**.

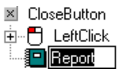
After running the report, TestFactory displays all of the application map objects in the right pane.



Name	Object Path
Application Map	Application Map
StartAUT	Application Map.StartAUT
Classics Login	Classics Login
E&xit	Classics Login.E&xit
LeftClick	Classics Login.E&xit.LeftClick
Exit	Classics Login.E&xit.LeftClick.Exit
OK	Classics Login.OK
Login	Classics Login.OK.Login
Close	Classics Login.OK.Login.Close
Classics Online	Classics Online
Tree	Classics Online.Tree
Bach	Classics Online.Bach
Brandenburg Concertos Nos. 1...	Classics Online.Brandenburg Cor....

In the Hierarchy report, the names of child objects are indented beneath the names of the parent object.

TestFactory inserts a report object below the object you clicked in the application map.



TestFactory report object

4. To name the report, type a name in the active text box.

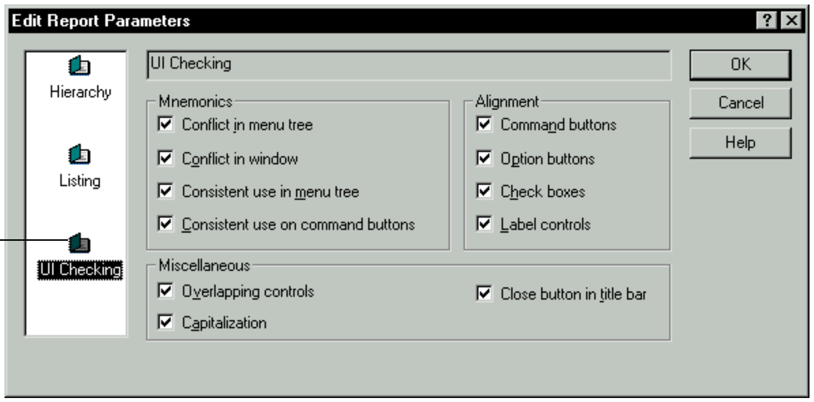
Creating a UI Checking Report

This exercise shows you how to create a UI Checking report. A UI Checking report provides information on mnemonics conflicts, incorrect alignment of controls, and other potential problems that TestFactory detects in the user interface.

To configure and run a UI Checking report:

1. Insert a report object at any location in the application map.
2. In the left region of the Edit Report Parameters dialog box, click the **UI Checking** icon.

UI Checking icon



UI Checking

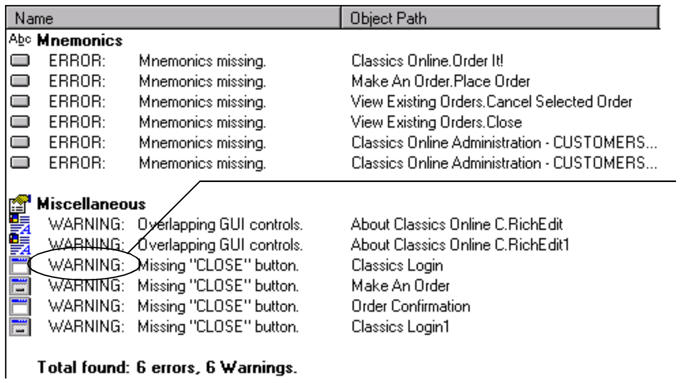
Mnemonics
- ☑ Conflict in menu tree
- ☑ Conflict in window
- ☑ Consistent use in menu tree
- ☑ Consistent use on command buttons

Alignment
- ☑ Command buttons
- ☑ Option buttons
- ☑ Check boxes
- ☑ Label controls

Miscellaneous
- ☑ Overlapping controls
- ☑ Capitalization
- ☑ Close button in title bar

3. Leave all of the check boxes selected and click **OK**.

TestFactory runs the report and displays the results in the right pane.



```
Name                                     Object Path
Abc Mnemonics
    ERROR:     Mnemonics missing.        Classics Online.Order It!
    ERROR:     Mnemonics missing.        Make An Order.Place Order
    ERROR:     Mnemonics missing.        View Existing Orders.Cancel Selected Order
    ERROR:     Mnemonics missing.        View Existing Orders.Close
    ERROR:     Mnemonics missing.        Classics Online Administration - CUSTOMERS...
    ERROR:     Mnemonics missing.        Classics Online Administration - CUSTOMERS...

Miscellaneous
    WARNING:   Overlapping GUI controls.  About Classics Online C.RichEdit
    WARNING:   Overlapping GUI controls.  About Classics Online C.RichEdit1
    WARNING:   Missing "CLOSE" button.    Classics Login
    WARNING:   Missing "CLOSE" button.    Make An Order
    WARNING:   Missing "CLOSE" button.    Order Confirmation
    WARNING:   Missing "CLOSE" button.    Classics Login1

    Total found: 6 errors, 6 Warnings.
```

UI Checking report results in the right pane
Double-click here to jump to the Classics Login object in the application map.

The UI Checking report lists every control in the Classics user interface that might have a problem, the type of problem found, and the application map path to the object mapped for the control.

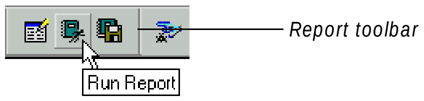
To jump to the object mapped for an item listed in the report, double-click its name.

Updating a TestFactory Report After an Application Changes

As changes are made to the application with which you are working, you can update an existing report by running it again.

To update a TestFactory report:

1. Click the report object in the application map.
2. On the Report toolbar, click **Run Report**.

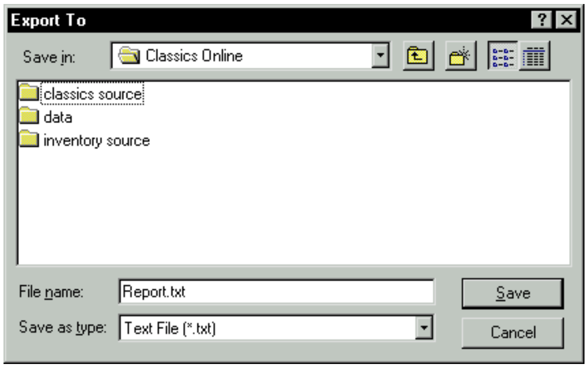


Exporting a TestFactory Report as a Text File

You can export a TestFactory report as a text file to edit in other applications.

To export a report as a text file:

1. Click a report object in the application map.
2. On the Report toolbar, click **Export Report**.



3. Specify a report file name and a destination directory, and then click **Save**.

Printing a TestFactory Report

To print the contents of a TestFactory report:

1. Click the report object in the application map.
2. Click anywhere in the right pane of the TestFactory window.
3. On the Standard toolbar, click **Print**.

Summary

By completing the exercises in this example, you've learned how to specify styles and string cases for input controls, use single-level and full-depth mapping to incrementally map areas of an application, use interaction objects to map alternative paths, create region objects for unmapped controls, create TestFactory reports, and more. This practice will help you develop an accurate and complete map of your own application in TestFactory.

What's Next

Now that you have incrementally mapped Classics to full depth, you can begin to test the application. The exercises in the next example show you how to run Pilots that generate scripts to test Classics.

▸ ▸ ▸ EXAMPLE 3

Automatically Generating Scripts

Objectives

- Set up a Pilot to test the Classics Order It! feature, and run the Pilot on your local machine.
- Examine the Pilot run results.
- Insert and run Pilots to test other functional areas of Classics.

Scenario

In Example 2, you incrementally mapped Classics to full depth. To develop the map, you excluded a control from mapping and testing, assigned styles to input controls, and mapped alternative paths in the Make an Order feature. The exercises in this example show you how to run Pilots to test the application, and how to examine the Pilot run results.

Running a Pilot to Test a Functional Area of the Sample Application

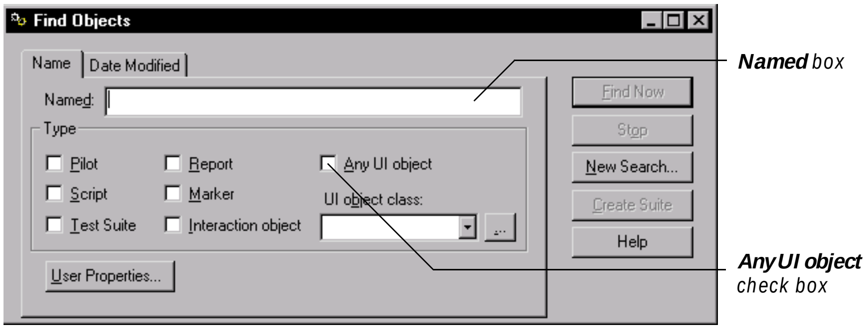
This section describes how to insert a Pilot at the Classics Order It! button, specify the duration of the Pilot run in the **Stop Criteria** tab, and then run the Pilot on your local machine to test the Order It! feature.

You'll start by inserting a Pilot at the Order It! button in the application map.

To locate the Order It! button:

1. Click **Find Objects** on the Standard toolbar.

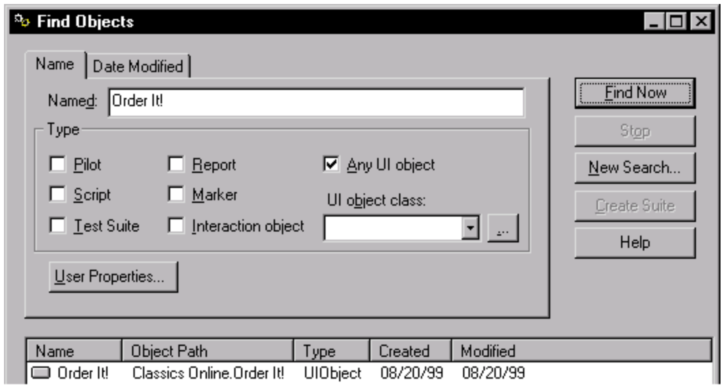
2. To clear the results of your last search, click **New Search**, and then click **Yes** to confirm that you want to clear the current search results.



3. In the **Named** box, type **Order It!**

4. Under **Type**, select the **Any UI object** check box.

5. Click **Find Now**.



Name	Object Path	Type	Created	Modified
Order It!	Classics Online.Order It!	UIObject	08/20/99	08/20/99

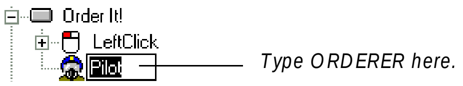
6. In the **Name** column, double-click **Order It!**

7. Close the Find Objects window.

The Order It! object is selected in the application map.

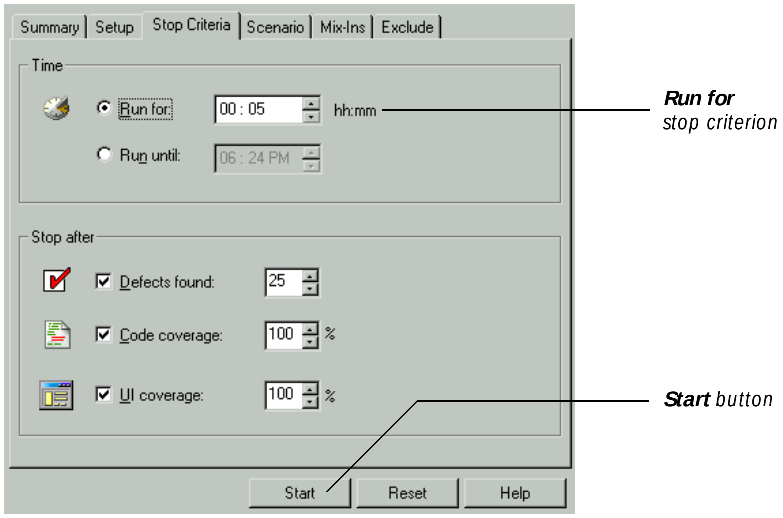
To insert and set up a Pilot:

1. In the application map, leave the Order It! object selected and click **Insert** → **Pilot**.



TestFactory inserts a Pilot object at the bottom of the application map branch below the Order It! window object.

2. Type **ORDERER** in the active text box for the new Pilot object, and then press ENTER.
3. In the right pane, click the **Stop Criteria** tab.



Time
- (●) Run for: 00 : 05 hh:mm
- ( ) Run until: 06 : 24 PM

Stop after
- ☑ Defects found: 25
- ☑ Code coverage: 100 %
- ☑ UI coverage: 100 %

4. To set a Pilot run duration of five minutes, leave **Run for** selected, and change the **hh:mm** value to **00:05**.

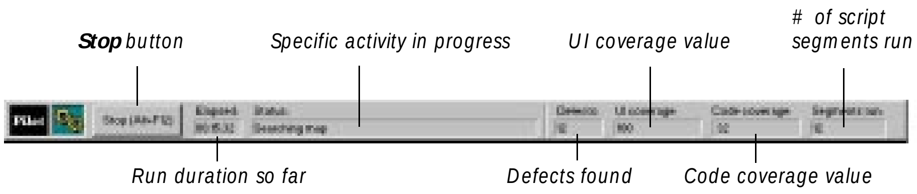
NOTE: For the Classics sample application, run Pilots for between five and ten minutes. For complex applications, you would run a Pilot for up to several hours.

5. To start the Pilot run, click **Start**.

After the run starts, you'll see the following occur on the screen:

- The TestFactory window closes.
- The Pilot progress bar opens at the bottom of the screen.
- A mask covers the screen and displays the *Running Pilot* message.
- TestFactory starts Robot, and then minimizes the Robot window.
- TestFactory starts and tests Classics.

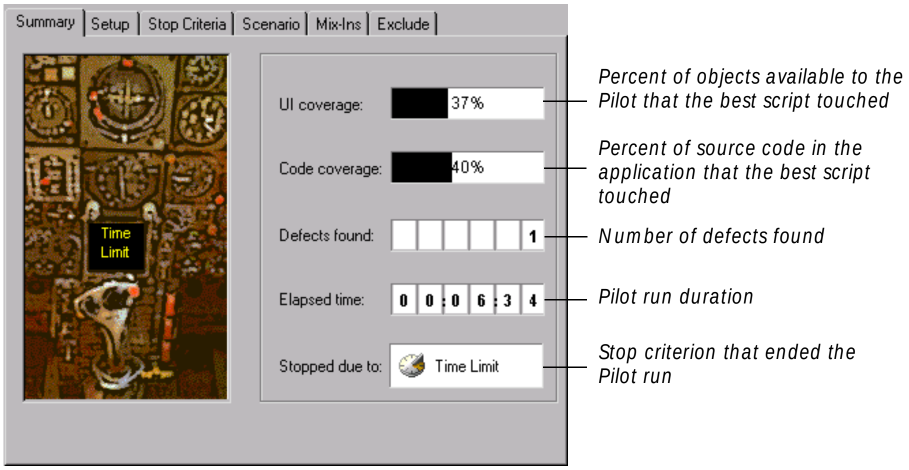
During the Pilot run, the Pilot progress bar displays testing information.



NOTE: Do not try to use the machine while the Pilot is running.

Examining the Pilot Run Results

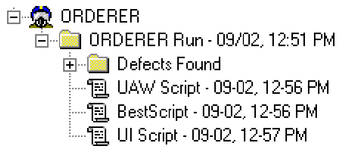
After the Pilot run is completed, the **Summary** tab displays summary run information in the right pane.


Summary
Setup
Stop Criteria
Scenario
Mix-Ins
Exclude
Time Limit
UI coverage: 37%
Code coverage: 40%
Defects found: 1
Elapsed time: 00:06:34
Stopped due to: Time Limit
Percent of objects available to the Pilot that the best script touched
Percent of source code in the application that the best script touched
Number of defects found
Pilot run duration
Stop criterion that ended the Pilot run

TestFactory's TestMaker technology generates scripts that exercise as much of an application as possible. A single Pilot run generates a **best script** and a **UI script**. The best script provides maximum coverage of the application's source code and user interface using the least redundant script code possible. The UI script is designed to exercise each and every control available to a Pilot just once. You can run the UI script later as a simple smoke test to check the controls in the user interface.

If a Pilot uncovers severe program defects such as crashes, Visual Basic run-time errors, or assertion failures, it generates **defect scripts** that make it easy for you to pinpoint defective source code. If a Pilot encounters controls that aren't mapped, or that are mapped on a path other than the one the Pilot has taken, the run results also include a **UAW (unexpected active window) script**.

To expand the Pilot run results folder and see the scripts your Pilot run produced, click **Expand** on the toolbar.



```
ORDERER
  ORDERER Run - 09/02, 12:51 PM
    Defects Found
    UAW Script - 09-02, 12-56 PM
    BestScript - 09-02, 12-56 PM
    UI Script - 09-02, 12-57 PM
```

Folder containing the Pilot run results

The scripts that the Pilot generated are in the ORDERER-Run-< date, time> folder below the Pilot object in the application map. This Pilot run generated a defect script, a UAW script, a best script, and a UI script.

In the following exercises, you'll examine the scripts that the Pilot run generated.

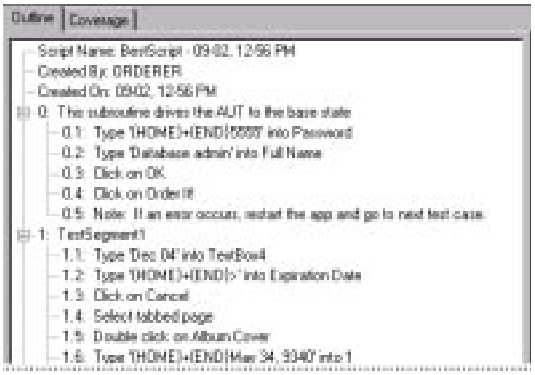
Don't be concerned if your Pilot run results look somewhat different than the results you see in the following figures. The Pilot that you run will take a slightly different path through the application. It will also exercise controls in the application using different random input.

Viewing the Outline for the Best Script

The script **Outline** tab lists all of the steps that a script took to test an area of the application.

To view the steps that the best script took during the Pilot run:

▶ Click the best script object in the application map.



```
Script Name: BestScript - 09-02, 12-56 PM
Created By: ORDERER
Created On: 09-02, 12-56 PM
0: This subroutine drives the AUT to the base state
    0.1: Type '{HOME}+{END}5555' into Password
    0.2: Type 'Database admin' into Full Name
    0.3: Click on OK
    0.4: Click on Order It!
    0.5: Note: If an error occurs, restart the app and go to next test case.
1: TestSegment1
    1.1: Type 'Dec 04' into TextBox4
    1.2: Type '{HOME}+{END}>' into Expiration Date
    1.3: Click on Cancel
    1.4: Select tabbed page
    1.5: Double click on Album Cover
    1.6: Type '{HOME}+{END}Mar 34, 3340' into 1
```

*The **Outline** tab lists all of the steps that the best script took to test the Order It! feature.*

TestFactory displays the steps in the **Outline** tab in the right pane.

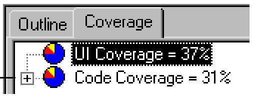
Viewing the Coverage Values for the Best Script

TestFactory calculates **UI coverage** values for all of the scripts a Pilot generates. If you run the Pilot against an instrumented application, TestFactory calculates **code coverage** for its best scripts and UI scripts. The coverage values indicate the thoroughness of testing and help you determine which features to test next. Code coverage tells you how well a script exercises the AUT and is an indirect indicator of the quality of the generated scripts. A Pilot calculates code coverage as it creates and runs new script segments. At the same time, it identifies and discards redundant script segments that do not increase code coverage.

The **Coverage** tab displays the UI coverage and code coverage values for the best script. To see these values:

1. In the right pane, click the **Coverage** tab.

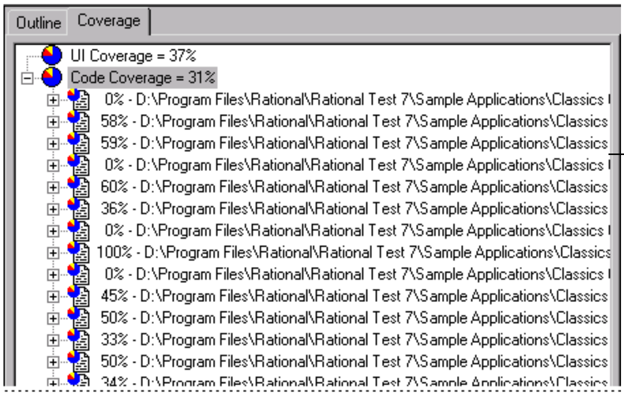
   The **UI Coverage** value shows the percentage of user interface available to the Pilot that the best script touched. The **Code Coverage** value shows the percentage of total source code that the best script exercised.



UI Coverage = 37%
Code Coverage = 31%

Click here to see the code coverage values for all Classics source files.

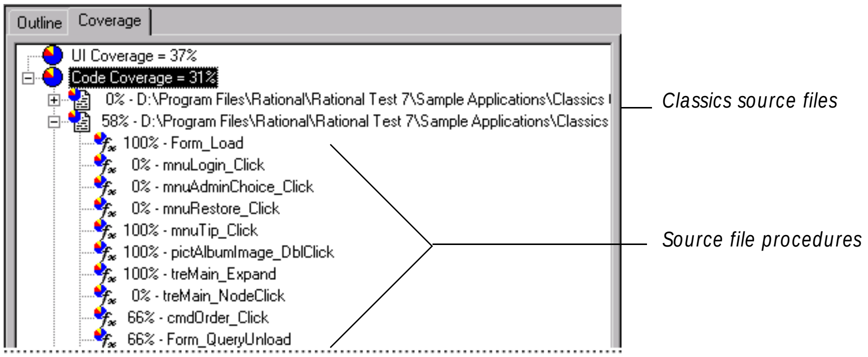
2. To see the code coverage values for every Classics source file, click the plus (+) character next to the **Code Coverage** item.



```
UI Coverage = 37%
Code Coverage = 31%
   0% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  58% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  59% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
   0% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  60% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  36% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
   0% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
 100% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
   0% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  45% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  50% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  33% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  50% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
  34% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
```

Source files

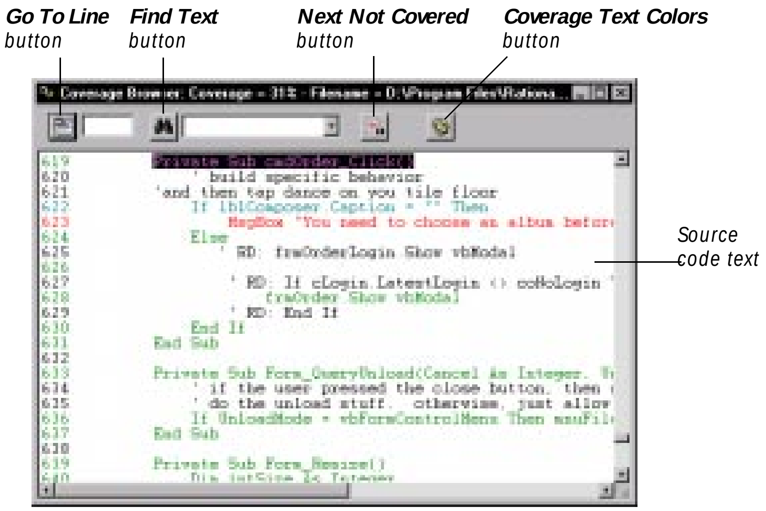
3. To see coverage values for procedures in a source file, expand the source file item.



```
UI Coverage = 37%
Code Coverage = 31%
   0% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
   58% - D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics
      100% - Form_Load
        0% - mnuLogin_Click
        0% - mnuAdminChoice_Click
        0% - mnuRestore_Click
      100% - mnuTip_Click
      100% - pictAlbumImage_DblClick
      100% - treMain_Expand
        0% - treMain_NodeClick
       66% - cmdOrder_Click
       66% - Form_QueryUnload
```

You can open the **Coverage Browser** window to see exactly how the best script exercised the source code for a procedure.

4. To view source code coverage details for a procedure in the Coverage Browser, double-click the procedure item listed in the **Coverage** tab.


Go To Line *button*
Find Text *button*
Next Not Covered *button*
Coverage Text Colors *button*
Coverage Browser: Coverage = 31% - Filename = D:\Program Files\Rationa...
Source code text

```
Private Sub cmdOrder_Click()
        ' build specific behavior
    'and then tap dance on you tile floor
        If lblComposer.Caption = "" Then
            MsgBox "You need to choose an album befor
        Else
           ' RD: frmOrderLogin.Show vbModal

           ' RD: If cLogin.LatestLogin <> coNologin '
              frmOrder.Show vbModal
           ' RD: End If
        End If
    End Sub

    Private Sub Form_QueryUnload(Cancel As Integer, U
        ' if the user pressed the close button, then
        ' do the unload stuff.  otherwise, just allow
        If UnloadMode = vbFormControlMenu Then mnuFil
    End Sub

    Private Sub Form_Resize()
        Dim intSize As Integer
```

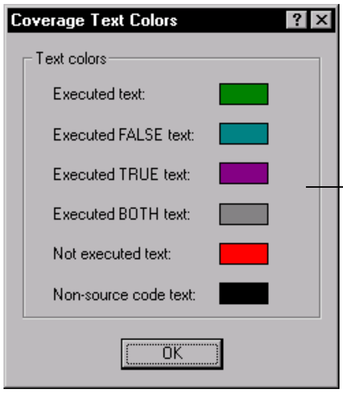
5. To jump to a specific line of source code, type the line number in the first edit box at the top of the browser, and then click **Go To Line**.

6. To jump to the first instance of a text string, type the text string in the second text box at the top of the browser, and then click **Find Text**.

7. To jump to the next line of source code that the script did *not* cover, click **Next Not Covered**.

The text in the Coverage Browser is color-coded so that you can see exactly what code the best script did and did not execute.

8. To see the current color coding scheme for source code text, click **Coverage Text Colors**.



Executed text:	🟩
Executed FALSE text:	🟦
Executed TRUE text:	🟪
Executed BOTH text:	⬜
Not executed text:	🟥
Non-source code text:	⬛

The Coverage Text Colors dialog box shows the current color coding scheme for code text displayed in the Coverage Browser.

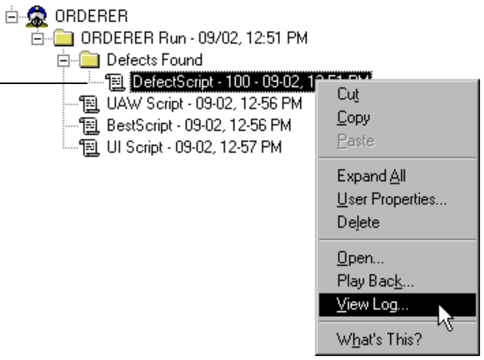
Viewing the Test Log for a Defect Script

You can locate the exact line of script code that uncovered a defect in the sample application by opening the log for the defect script in the Rational LogViewer.

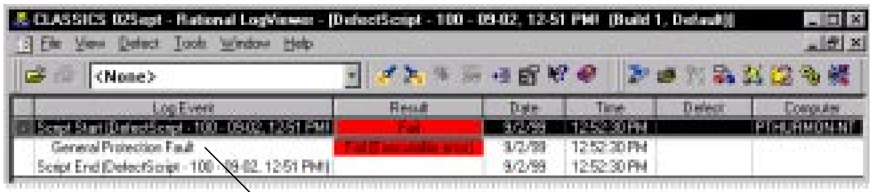
To open and examine the log for the defect script:

1. Expand the **Defects Found** folder.
2. Right-click the defect script, and then click **View Log** on the shortcut menu.

Defect script


ORDERER
ORDERER Run - 09/02, 12:51 PM
Defects Found
DefectScript - 100 - 09-02, 12:51 PM
UAW Script - 09-02, 12:56 PM
BestScript - 09-02, 12:56 PM
UI Script - 09-02, 12:57 PM
Cut
Copy
Paste
Expand All
User Properties...
Delete
Open...
Play Back...
View Log...
What's This?

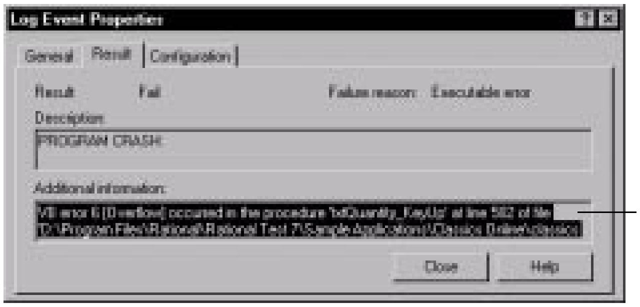
TestFactory starts the Rational LogViewer. The LogViewer displays the test log for the defect script.


CLASSICS 025ept - Rational LogViewer - [DefectScript - 100 - 09-02, 12:51 PM (Build 1, Default)]
File View Defect Tools Window Help
<None>

Log Event	Result	Date	Time	Defect	Computer
Script Start (DefectScript - 100 - 09.02, 12:51 PM)	Fail	9/2/99	12:52:30 PM		[illegible]
General Protection Fault	Fail (Executable error)	9/2/99	12:52:30 PM		
Script End (DefectScript - 100 - 09.02, 12:51 PM)		9/2/99	12:52:30 PM		

General Protection Fault *log event*

3. In the **Log Event** column, right-click the **General Protection Fault** item that has the failed result, and then click **Properties** on the shortcut menu.
4. On the Log Event Properties dialog box, click the **Result** tab.



Result: Fail
Failure reason: Executable error
Description: PROGRAM CRASH
Additional information: VB error 6 (Overflow) occurred in the procedure 'txtQuantity_KeyUp' at line 582 of file D:\Program Files\Rational\Rational Test 7\Sample Applications\Classics Online\classics

The ***Additional information*** *box displays the source code file and line number associated with the script failure.*

5. Under **Additional information**, find the line number and source code file associated with the script failure, and make a note of them.
6. Click **Close**.
7. Close the LogViewer window.

After you determine the line number and source code file associated with the defect script failure, play back the defect script in Robot to reproduce the failure.

To play back a defect script in Robot:

▸ In the application map, right-click the defect script object, and then click **Play Back** on the shortcut menu.

For instructions on debugging a script in Robot, see the *Using Rational Robot* manual.

Examining a UAW Script

If a Pilot encounters a window that isn't mapped on the path it has taken, or if it can't see a window that it expects to see, then the Pilot generates a **UAW script**. You can use a UAW script to trace the steps a Pilot took before losing its way in the application during testing.

To examine the UAW script that the ORDERER Pilot generated:

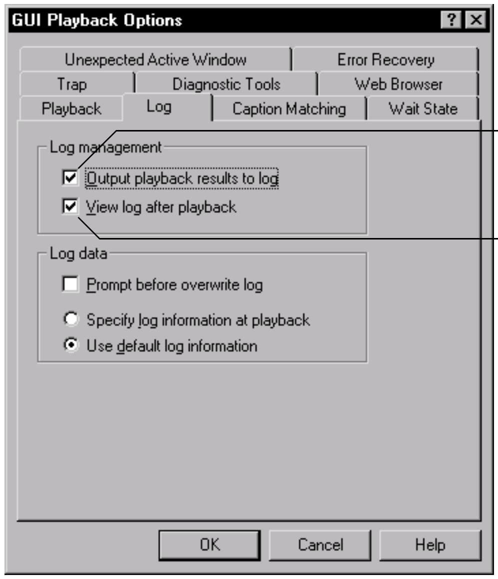
1. In the application map, click the UAW script.
2. Review the steps displayed on the **Outline** tab in the right pane to determine what you can do to improve the application map.

If a Pilot generates a UAW script for a window or dialog box that you have not yet mapped, you can insert and set up an interaction object that lets the Application Mapper navigate and map that path in the application. This allows you to test the new path when you run a Pilot again in the same area of the application.

Identifying the UAW

To see what part of the Classics application generated the UAW script:

1. In the application map, right-click the UAW script, and then click **Open** on the shortcut menu.
2. In Robot, click **Tools** → **GUI Playback Options**.
3. Click the **Log** tab.



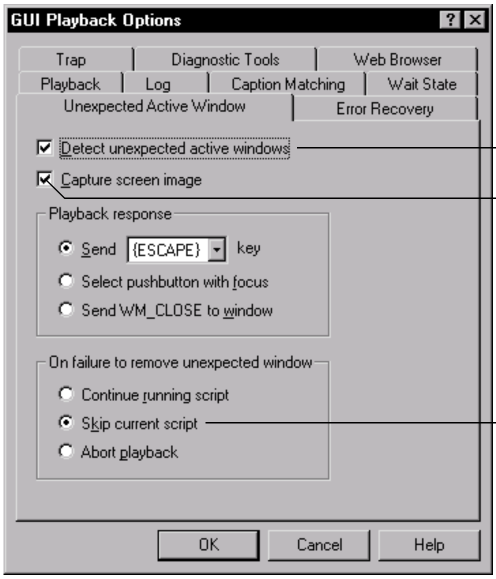
Log management
- [✓] Output playback results to log
- [✓] View log after playback

Log data
- [ ] Prompt before overwrite log
- ○ Specify log information at playback
- ● Use default log information

Output playback result to log *check box*

View log after playback *check box*

4. Under **Log management**, select the **Output playback result to log** and the **View log after playback** check boxes.
5. Click the **Unexpected Active Window** tab.



[✓] Detect unexpected active windows
[✓] Capture screen image

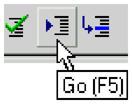
Playback response
[✓ Send] {ESCAPE} key
Select pushbutton with focus
Send WM_CLOSE to window

On failure to remove unexpected window
Continue running script
[✓ Skip current script]
Abort playback

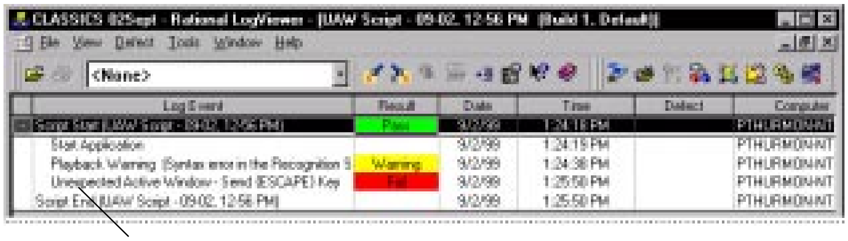
6. Select the **Detect unexpected active windows** and the **Capture screen image** check boxes.
7. Under **On failure to remove unexpected window**, click **Skip current script**.
8. Click **OK**.

To run the UAW script and view the run log in the LogViewer:

1. Click **Go** on the Robot Standard toolbar.



2. Robot plays back the script until the UAW opens. After the script playback ends, the LogViewer starts and displays the log for the UAW script run.


CLASSICS 02Sept - Rational LogViewer - [UAW Script - 09:02, 12:56 PM (Build 1, Default)]
File View Defect Tools Window Help
<None>

Log Event	Result	Date	Time	Defect	Computer
Script Start (UAW Script - 09:02, 12:56 PM)	Pass	9/2/99	1:24:18 PM		PTHURMOND-NT
Start Application		9/2/99	1:24:19 PM		PTHURMOND-NT
Playback Warning (Syntax error in the Recognition S	Warning	9/2/99	1:24:38 PM		PTHURMOND-NT
Unexpected Active Window - Send {ESCAPE} Key	Fail	9/2/99	1:25:50 PM		PTHURMOND-NT
Script End (UAW Script - 09:02, 12:56 PM)		9/2/99	1:25:50 PM		PTHURMOND-NT

Unexpected Active Window *event*

3. To view the properties of the unexpected active window, right-click the Unexpected Active Window item in the **Log Event** column of the LogViewer, and then click **Properties** on the shortcut menu.

4. To open the Image Comparator and view a bitmap of the screen that includes the UAW, double-click the **Unexpected Active Window** item in the **Log Event** column of the LogViewer.

   The Image Comparator shows a bitmap of the Order Confirmation message box, which is represented in the application map. The message box was identified as an unexpected active window because the Pilot took a path in the application map that did not include the message box. The Pilot activated the message box by using random data entries to exercise controls in its path.

5. Close the Image Comparator.

6. After you finish viewing the log information for the UAW script, quit the LogViewer, and quit the Classics sample application.

If a Pilot generates a UAW script for a window that is already mapped, as happened here, you can disregard the UAW script. To decrease the incidence of UAW scripts such as this, you can decrease the percentage of random data entry used to exercise a control that accepts typed input.

Testing More of the Classics Application

Now that you know how to set up and run a Pilot, run some more of them. Before you continue on to the next example, insert and run Pilots at the following destinations in the application map:

- Classics Online window object
- Customers menu object

Before you insert and run a Pilot at the Classics Online window object, exclude the Inventory menu item from testing. The **Inventory** menu command in Classics activates the secondary application Inventory, which you have not mapped yet.

To exclude the Inventory menu item from testing:

1. In the application map, click the Inventory menu object mapped beneath the Admin menu object.
2. In the top right pane, expand the **Pilot** properties group.
3. Double-click the **Value** field for **ExerciseDuringTesting**, and then click **Never**.

Remember to set a five- to ten-minute duration for these Pilot runs. For information about setting the Pilot run duration, see steps 3 and 4 on page 53.

After each Pilot run finishes, examine the scripts that the Pilot generated.

Summary

By completing the exercises in this example, you've learned how to run Pilots to test different functional areas of an application, and how to view the run results.

What's Next

The next example shows you how to map the secondary application *Inventory*.

▸▸▸ E X A M P L E 4

Mapping a Secondary Application

Objectives

- Include the secondary application *Inventory* in the mapping process.
- Map a path in the Inventory user interface to full depth.

Scenario

An application can consist of a main application and a set of secondary applications that the main application loads and executes. A secondary application can be one that is developed as part of the application, or it can be a third-party application. Although TestFactory excludes secondary applications from mapping by default, you can map them in addition to the main executable file. The following exercises show you how.

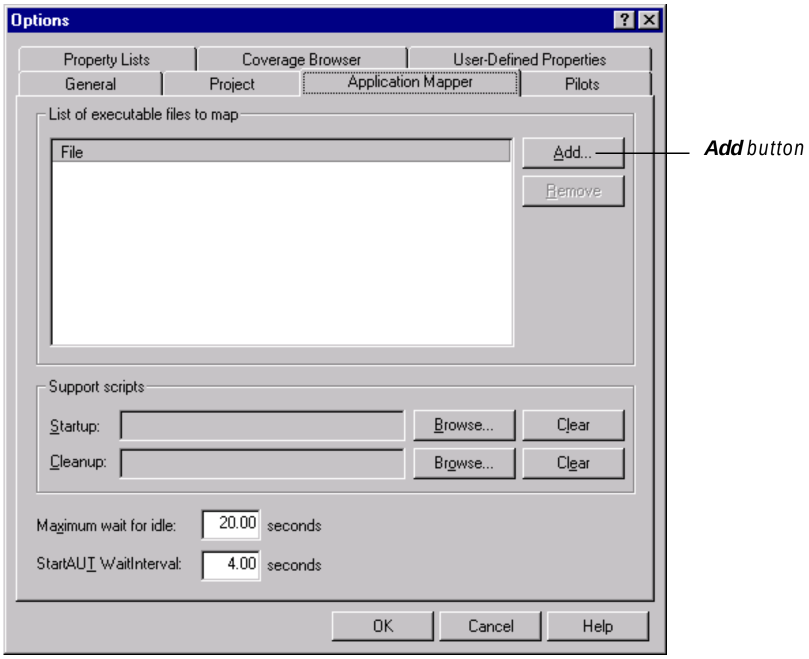
Mapping the Inventory Application

If a user logs on to Classics as a database administrator and selects the **Inventory** command in the Classics **Admin** menu, Classics loads and executes the secondary application *Inventory*. When you map an application that calls a secondary application, TestFactory maps the top level of controls it encounters in the secondary application, but does not map controls at deeper levels. To fully map a secondary application, you must add its name to the list of executable files on the **Application Mapper** tab.

NOTE: If you want to test a secondary application and get code coverage results for the scripts your Pilots create for it, instrument the source code for the secondary application before you map it.

To list the Inventory application in the **Application Mapper** tab:

1. Click **Tools** → **Options**, and then click the **Application Mapper** tab.

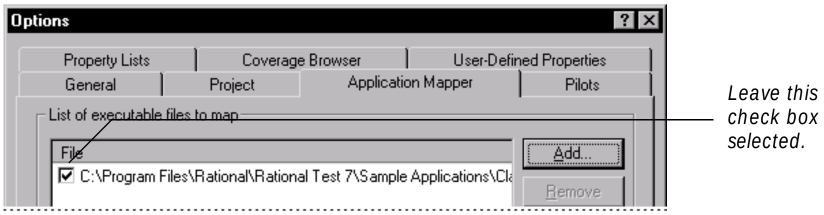


2. Under **List of executable files to map**, click **Add**.

3. To find the executable file for the secondary application from the Open dialog box, browse to the following directory:

   ...\Rational Test 7\Sample Applications\Classics Online\InventoryC.exe

4. Click **Open**.



☑ C:\Program Files\Rational\Rational Test 7\Sample Applications\Cl

5. In the **File** list, leave the check box next to the file name selected.

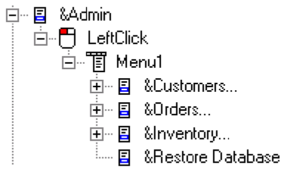
6. To save this addition and close the Options dialog box, click **OK**.

Mapping the Suppliers Feature of Inventory

When you mapped Classics to full depth in Example 2, TestFactory mapped the first level of controls in the Inventory application. Among the Inventory controls mapped during that mapping session is the **Suppliers** button. This exercise shows you how to map the Suppliers feature of the Inventory application to full depth.

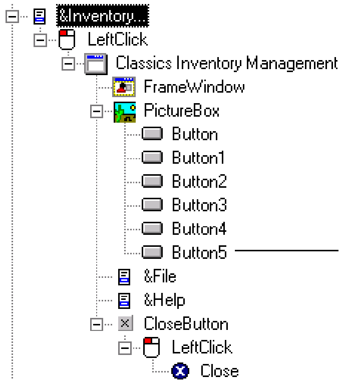
To view the first level of controls in Inventory, including the object mapped for the **Suppliers** button:

1. In the application map, expand the Admin menu object to display the menu command objects mapped beneath it.



```
&Admin
  LeftClick
    Menu1
      &Customers...
      &Orders...
      &Inventory...
      &Restore Database
```

2. Right-click the Inventory object, and then click **Expand All** on the shortcut menu.



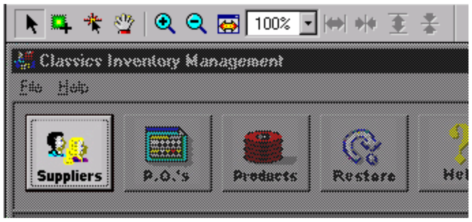
```
&Inventory...
  LeftClick
    Classics Inventory Management
      FrameWindow
      PictureBox
        Button
        Button1
        Button2
        Button3
        Button4
        Button5
      &File
      &Help
      CloseButton
        LeftClick
          Close
```

When you first map Classics, TestFactory maps only the top level of controls that it encounters in Inventory.

After you specify the InventoryC.exe file for mapping, you can map Inventory to full depth.

Button5 object

3. To see the Suppliers button highlighted in the Image view in the lower right pane, click the Button5 object in the application map.



Button5 in the application map represents the ***Suppliers*** *button in Inventory.*

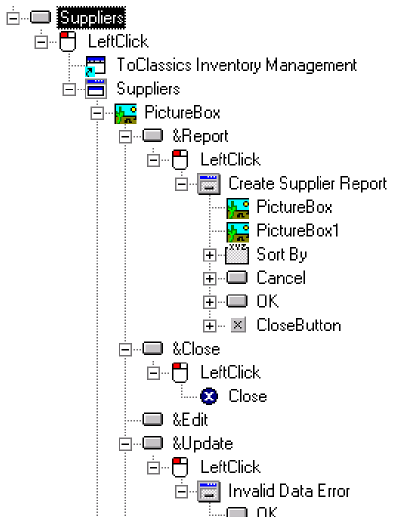
4. To rename the button object, click F2, and then type **Suppliers** in the text box.

To map the Suppliers button to full depth:

- Right-click the Suppliers object, point to **Map It!**, and then click **Full** on the shortcut menu.

To see the fully mapped Suppliers button after mapping is completed:

- Expand the Suppliers object one branch at a time by clicking the plus (+) character next to each new object displayed.



```
Suppliers
  LeftClick
    ToClassics Inventory Management
    Suppliers
      PictureBox
        &Report
          LeftClick
            Create Supplier Report
              PictureBox
              PictureBox1
              Sort By
              Cancel
              OK
              CloseButton
        &Close
          LeftClick
            Close
        &Edit
        &Update
          LeftClick
            Invalid Data Error
              OK
```

You can use the **Go To < Control> in AUT** command to start Inventory and automatically navigate to the **Suppliers** button. You can then compare your mapping results with the Inventory user interface.

To start Inventory and go to the Suppliers button:

- In the application map, right-click the Suppliers button object, and then click **Go To "Suppliers" in AUT** on the shortcut menu.

Summary

In this exercise, you've learned how to map beyond the first level of controls in a secondary application that the main application loads and executes.

What's Next

The next example shows you how to create a Test Suite and run the scripts it contains as a batch job on your local machine.

▸ ▸ ▸ E X A M P L E 5

Creating and Running a Test Suite

Objectives

- Use the Find Objects window to find scripts and create a Test Suite.
- Run the Test Suite on your local machine.
- Examine the results of the Test Suite run.

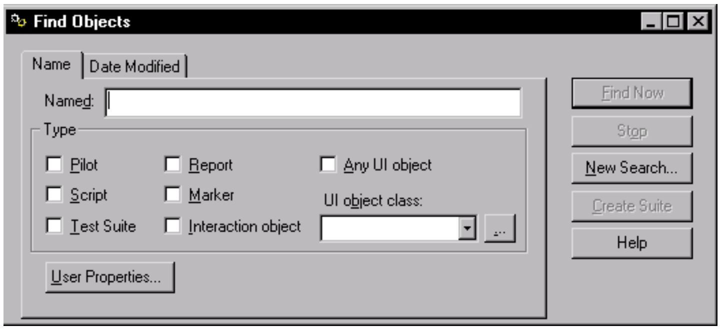
A Test Suite is the TestFactory object that you can use to organize scripts and other Test Suites, and run them as a batch job. The exercises in this example show you how to locate scripts from your Pilot runs, combine them in a Test Suite, and run the queued scripts on your local machine.

Creating and Running a Test Suite

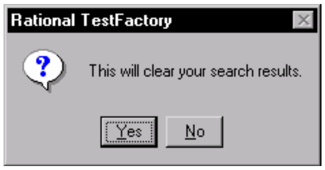
This section describes how to create a Test Suite using the Find Objects window, and to then run the Test Suite on your local machine.

To automatically create a Test Suite that includes all best scripts:

1. On the Standard toolbar, click **Find Objects**.

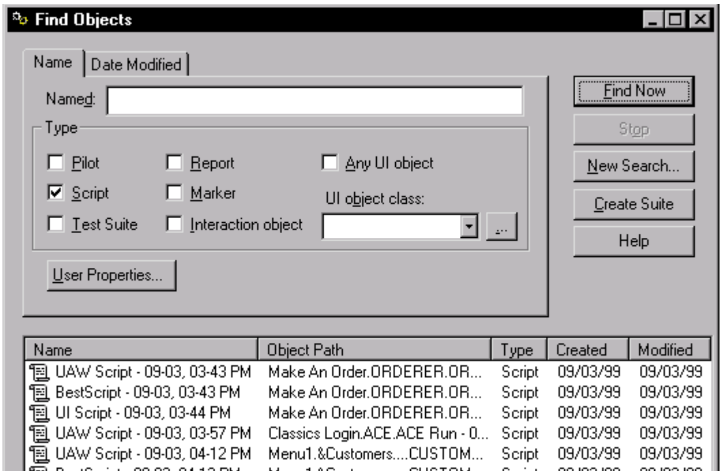


2. If the Find Objects window displays the results of a previous search, click **New Search**, and then click **Yes** to confirm that you want to clear the results.



This will clear your search results.

3. Under **Type**, select the **Script** check box.
4. To start the search for scripts, click **Find Now**.



Named:

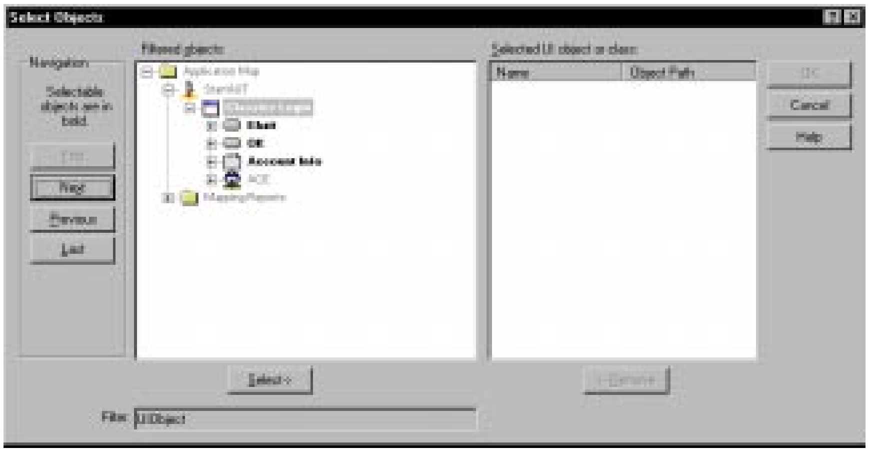
Type

- [ ] Pilot
- [✓] Script
- [ ] Test Suite
- [ ] Report
- [ ] Marker
- [ ] Interaction object
- [ ] Any UI object

UI object class:

Name	Object Path	Type	Created	Modified
UAW Script - 09-03, 03-43 PM	Make An Order.ORDERER.OR...	Script	09/03/99	09/03/99
BestScript - 09-03, 03-43 PM	Make An Order.ORDERER.OR...	Script	09/03/99	09/03/99
UI Script - 09-03, 03-44 PM	Make An Order.ORDERER.OR...	Script	09/03/99	09/03/99
UAW Script - 09-03, 03-57 PM	Classics Login.ACE.ACE Run - 0...	Script	09/03/99	09/03/99
UAW Script - 09-03, 04-12 PM	Menu1.&Customers....CUSTOM...	Script	09/03/99	09/03/99

5. Press CTRL+ and click every best script listed.
6. Click **Create Suite**.



Selectable objects are in bold.

Use the **Select Objects** dialog box to choose a parent object for the Test Suite in the application map. The parent object that you choose doesn't limit the scripts that you include in the Test Suite.

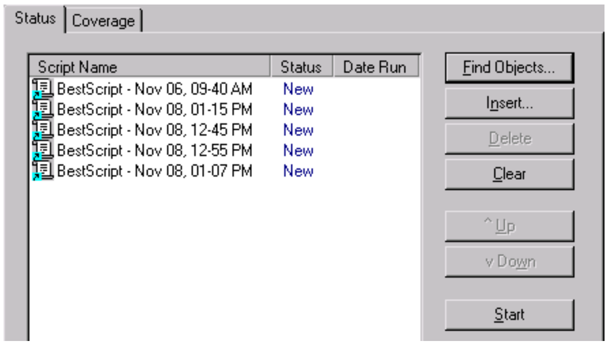
7. To choose a parent object for the Test Suite, click an object in the **Filtered objects** list, click **Select**, and then click **OK**.

   TestFactory inserts a Test Suite object in the application map under the parent object that you selected.

An Untitled Test Suite object is inserted in the application map.

8. Close the Find Objects window.
9. To name the Test Suite, press F2, and then type **Best Scripts**.

   The Test Suite **Status** tab in the right pane lists the scripts you selected.

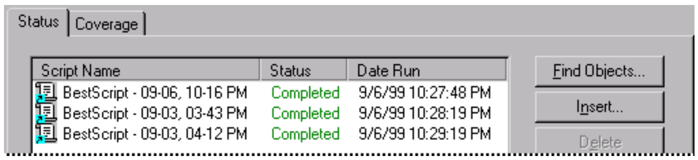


Script Name	Status	Date Run
BestScript - Nov 06, 09-40 AM	New	
BestScript - Nov 08, 01-15 PM	New	
BestScript - Nov 08, 12-45 PM	New	
BestScript - Nov 08, 12-55 PM	New	
BestScript - Nov 08, 01-07 PM	New	

*The **Status** tab lists the scripts that you selected.*

When you run a Test Suite on your local machine, TestFactory runs the scripts in the order in which they are listed on the **Status** tab.

10. To control the run order for a script, click the script name, and then use **Up** and **Down** to change its position in the list.
11. To start the Test Suite run, click **Start**.



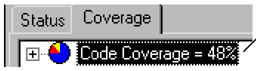
Script Name	Status	Date Run
BestScript - 09-06, 10-16 PM	Completed	9/6/99 10:27:48 PM
BestScript - 09-03, 03-43 PM	Completed	9/6/99 10:28:19 PM
BestScript - 09-03, 04-12 PM	Completed	9/6/99 10:29:19 PM

After the Test Suite run is completed, the values in the **Status** column show whether the script runs successfully completed or failed.

Examining the Code Coverage Results for a Test Suite

To view the coverage results for the Test Suite and the scripts it contains:

1. In the right pane, click the **Coverage** tab.



*The **Code Coverage** item shows the aggregate code coverage value for all of the scripts in the Test Suite.*

2. To see code coverage for the source files, expand the **Code Coverage** item. The tree lists every source file in Classics and the aggregate code coverage values for the scripts that touched them.
3. To see the coverage values for individual procedures within a source file, expand the source file.
4. To open the Coverage Browser and view the source code for a procedure, double-click the procedure.

NOTE: For information about viewing code coverage information, see *Viewing the Coverage Values for the Best Script* on page 56.

Summary

This example showed you how to use the Find Objects window to create a Test Suite of best scripts, and how to run the Test Suite on your local machine.

What's Next

The next example shows you how to use the AutoPilot to run multiple Pilots, scripts, and Test Suites as a batch job.

▸ ▸ ▸ EXAMPLE 6

Using the AutoPilot to Run Multiple Pilots, Scripts, and Test Suites

Objectives

- Start and add Pilots, scripts, and Test Suites to the AutoPilot window.
- Run the test objects listed in the AutoPilot window on your local machine.

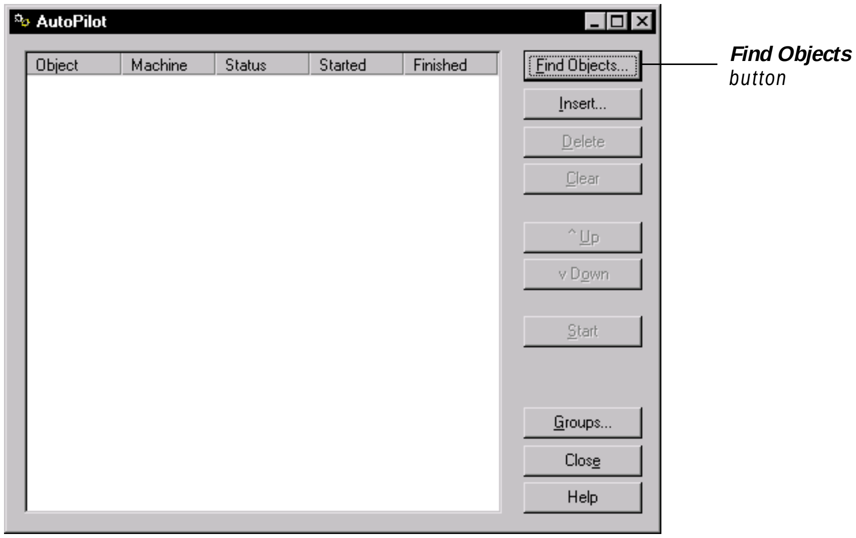
Scenario

The AutoPilot is the TestFactory tool that you use to run multiple Pilots, scripts, and Test Suites as a batch job on your local machine, or on Test Lab machines that have TestAccelerator installed on them. The AutoPilot helps you optimize your testing resources. You can use it to run test objects on your machine overnight. If you want to continue working on your local machine, you can use the AutoPilot to run tests on Test Lab machines. This example shows you how to run multiple test objects locally from the AutoPilot.

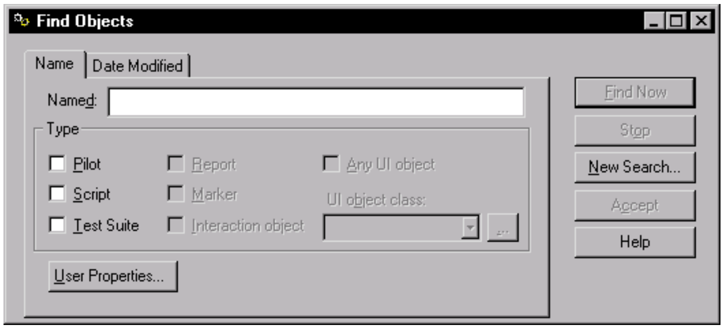
Adding Test Objects to Run in the AutoPilot

To open and add test objects to the AutoPilot window:

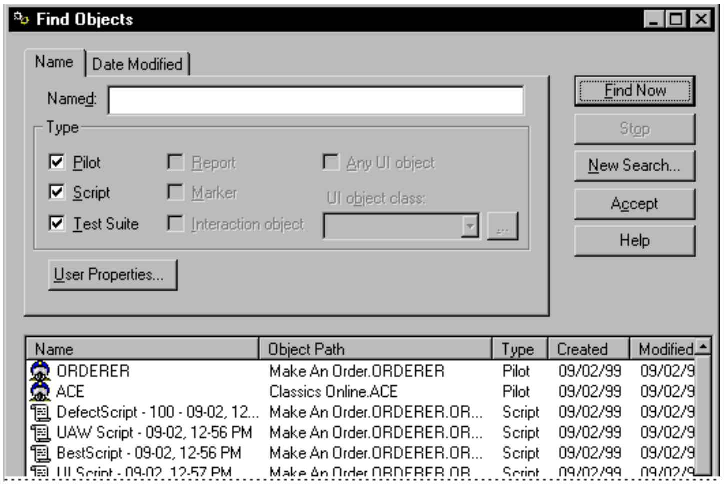
1. On the TestFactory toolbar, click **AutoPilot**.



2. Click **Find Objects**.



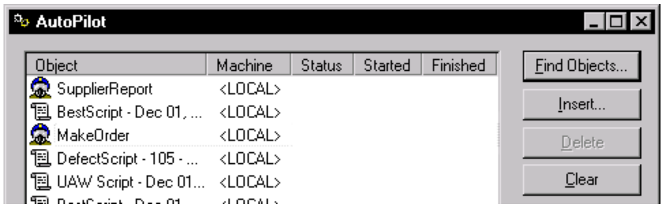
3. Select the **Pilot**, **Script**, and **Test Suite** check boxes, and then click **Find Now**.



Name	Object Path	Type	Created	Modified
ORDERER	Make An Order.ORDERER	Pilot	09/02/99	09/02/9
ACE	Classics Online.ACE	Pilot	09/02/99	09/02/9
DefectScript - 100 - 09-02, 12...	Make An Order.ORDERER.OR...	Script	09/02/99	09/02/9
UAW Script - 09-02, 12-56 PM	Make An Order.ORDERER.OR...	Script	09/02/99	09/02/9
BestScript - 09-02, 12-56 PM	Make An Order.ORDERER.OR...	Script	09/02/99	09/02/9
UI Script - 09-02, 12-57 PM	Make An Order.ORDERER.OR	Script	09/02/99	09/02/9

4. Press CTRL+ and select a few of the test objects listed.

5. To add these test objects to the **AutoPilot** window, click **Accept**.



Object	Machine	Status	Started	Finished
SupplierReport	<LOCAL>			
BestScript - Dec 01, ...	<LOCAL>			
MakeOrder	<LOCAL>			
DefectScript - 105 - ...	<LOCAL>			
UAW Script - Dec 01...	<LOCAL>			
BestScript - Dec 01	<LOCAL>			

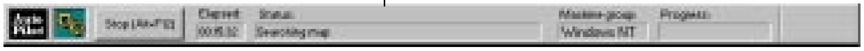
6. To change the run order for a test object, click the object name, and then use **Up** and **Down** to change its position in the list.

Running the AutoPilot on Your Local Machine

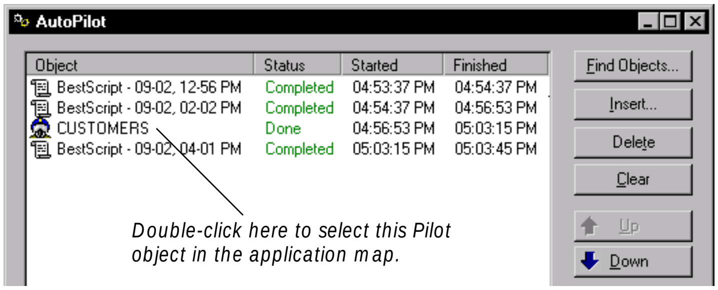
To begin running the listed test objects on your local machine, click **Start**.

As soon as you start the AutoPilot run, the TestFactory window minimizes and the AutoPilot progress bar displays run information at the bottom of the screen.

AutoPilot progress bar



After all of the listed test objects have been run, the AutoPilot displays the run result (**Completed**, **Done**, or **Failed**) for each test object.



Object	Status	Started	Finished
BestScript - 09-02, 12-56 PM	Completed	04:53:37 PM	04:54:37 PM
BestScript - 09-02, 02-02 PM	Completed	04:54:37 PM	04:56:53 PM
CUSTOMERS	Done	04:56:53 PM	05:03:15 PM
BestScript - 09-02, 04-01 PM	Completed	05:03:15 PM	05:03:45 PM

Double-click here to select this Pilot object in the application map.

To examine detailed run results for an individual test object, go to the object in the application map. To jump to a specific script, Pilot, or Test Suite in the application map, double-click its name in the **Object** column, and then close or minimize the AutoPilot window.

Summary

In this example, you learned how to add a selection of the test objects you've created for the CLASSICS project to the AutoPilot window and run them locally.

What's Next

The next example shows you how to set up a Test Lab machine running Rational TestAccelerator, and then run a Pilot on the machine.

▸▸▸ EXAMPLE 7

Running Tests on a Test Lab Machine

Objectives

- Start and set up Rational TestAccelerator on a Test Lab machine.
- Make the Test Lab machine available to the TestFactory machine.
- Create a Test Lab machine group.
- Run a Pilot on a Test Lab machine.

Scenario

As you develop more scripts, Test Suites, and Pilots, you can execute them on Test Lab machines running TestAccelerator. TestAccelerator is the agent application that manages the execution of scripts on the machines that make up the Test Lab.

Installing Rational TestAccelerator and Making the ClassicsC.exe File Available to the Test Lab Machine

To complete the exercises in this example, you must have already purchased Rational TestAccelerator and installed it on a Test Lab machine. For information about installing Rational TestAccelerator, see the *Rational Suite Installation Guide*.

From the Test Lab machine, you must have access to the exact same instrumented build of the ClassicsC.exe file that Test Factory uses. Before you start to set up TestAccelerator on the Test Lab machine, share the Classics Online folder on the TestFactory machine. If you have access to more than one machine for testing, you can also install TestAccelerator and Classics Online on those machines and include them in your Test Lab.

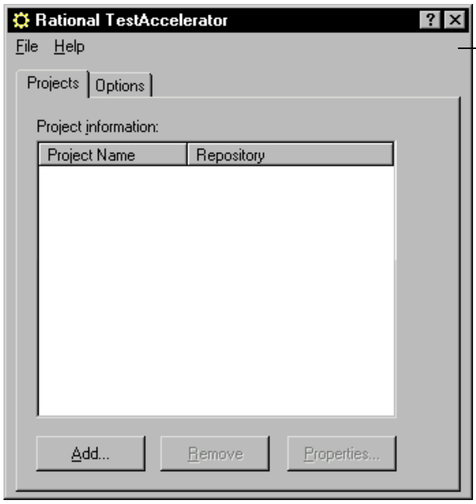
Starting and Setting Up TestAccelerator

The following exercises show you how to start and set up TestAccelerator on a Test Lab machine that has it installed.

Starting TestAccelerator

To start TestAccelerator:

1. Quit all open applications on the Test Lab machine and leave them closed until you quit TestAccelerator.
2. Click **Start** → **Programs** → **Rational TestAccelerator** → **Rational TestAccelerator**.



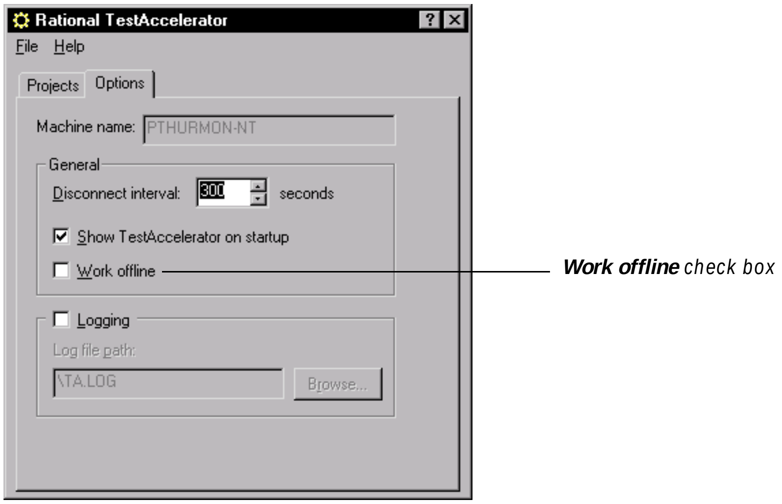
Rational TestAccelerator dialog box

Working Offline in TestAccelerator

After you start TestAccelerator, the machine that it runs on is available to machines running TestFactory. A TestFactory machine running scripts in the Test Lab can take control of a Test Lab machine while you are setting it up. To prevent this, you can work offline in TestAccelerator.

To work offline after you start TestAccelerator:

1. On the TestAccelerator dialog box, click the **Options** tab.



Rational TestAccelerator

Machine name: PTHURMON-NT

General

Disconnect interval: 300 seconds

☑ Show TestAccelerator on startup

☐ Work offline

☐ Logging

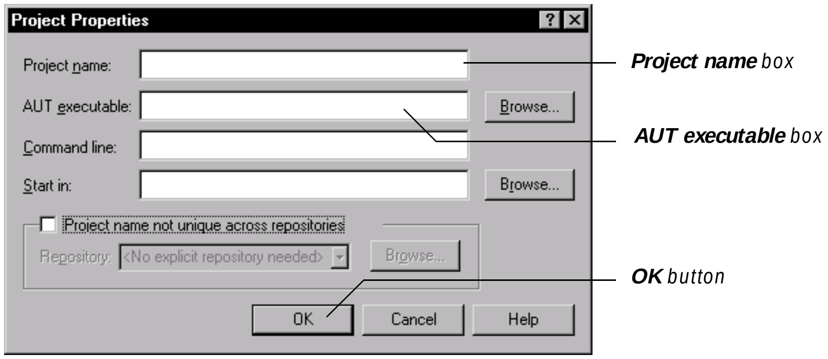
Log file path: \TA.LOG

2. Under **General**, select the **Work offline** check box.

Specifying the Project in TestAccelerator

To specify the CLASSICS project in TestAccelerator:

1. In the Rational TestAccelerator dialog box, click the **Projects** tab.
2. Click **Add**.



Project name:
AUT executable:
Command line:
Start in:
☐ Project name not unique across repositories
Repository: <No explicit repository needed>

3. In the **Project name** box, type **CLASSICS**.
4. In the **AUT executable** box, enter the path to the Classics Online folder on the shared drive of the TestFactory machine.
5. Click **OK**.

Working Online in TestAccelerator

To work online in TestAccelerator:

1. On the TestAccelerator dialog box, click the **Options** tab.
2. Under **General**, clear the **Work offline** check box.

This Test Lab machine is now ready to work.

Preparing to Test on a Test Lab Machine

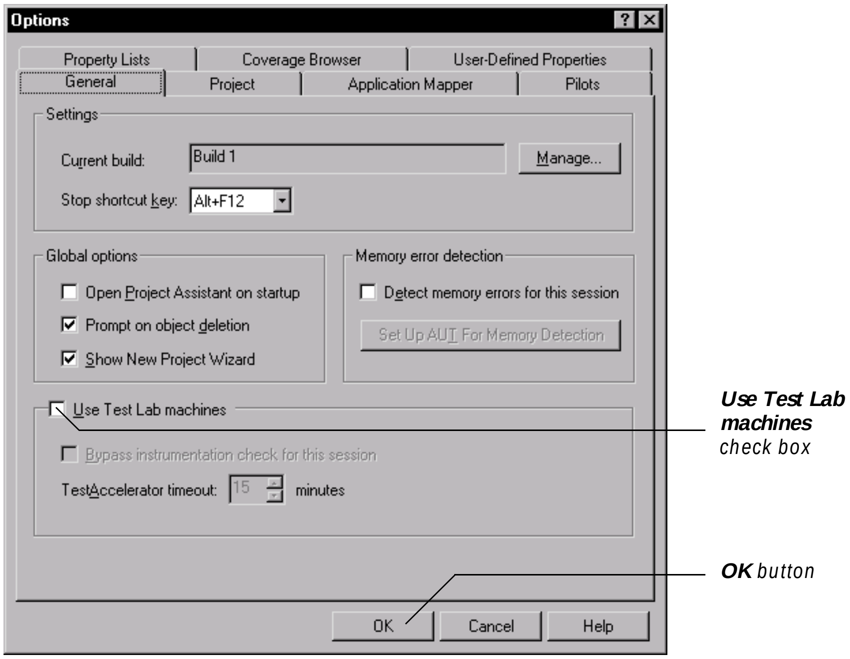
Now that you've set up TestAccelerator on a Test Lab machine, you can run Pilots, scripts, and Test Suites on it from the machine running TestFactory. Before you do, you first have to select the **Use Test Lab machines** option, and then create a machine group in TestFactory.

Making the Test Lab Machine Available to the TestFactory Machine

To make the Test Lab machine available to the TestFactory machine, you must select the **Use Test Lab machines** option in TestFactory.

To select the **Use Test Lab machines** option:

1. On the TestFactory machine, start TestFactory and log on to the CLASSICS project in the Classics Repository.
2. Click **Tools** → **Options**.



Settings
Current build: Build 1
Stop shortcut key: Alt+F12

Global options
☐ Open Project Assistant on startup
☑ Prompt on object deletion
☑ Show New Project Wizard

Memory error detection
☐ Detect memory errors for this session

☐ Use Test Lab machines
☐ Bypass instrumentation check for this session
TestAccelerator timeout: 15 minutes

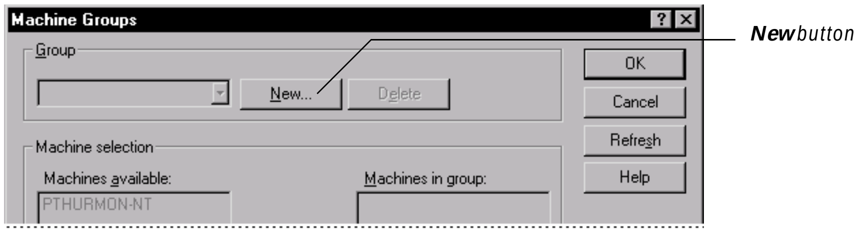
3. On the **General** tab, under **Global options**, select the **Use Test Lab machines** check box.
4. To close the Options dialog box, click **OK**.

Creating a Test Lab Machine Group

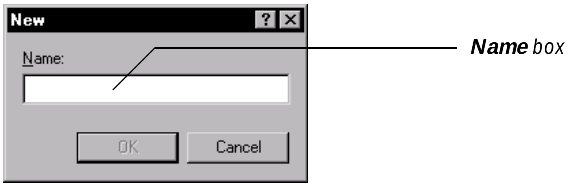
When you run test objects on Test Lab machines from TestFactory, you can assign the objects to one or more machine groups. A machine group can include one or several Test Lab machines.

To create a Test Lab machine group from within TestFactory:

1. To open the Machine Groups dialog box, click **Tools** → **Machine Groups**.



2. Under **Group**, click **New**.

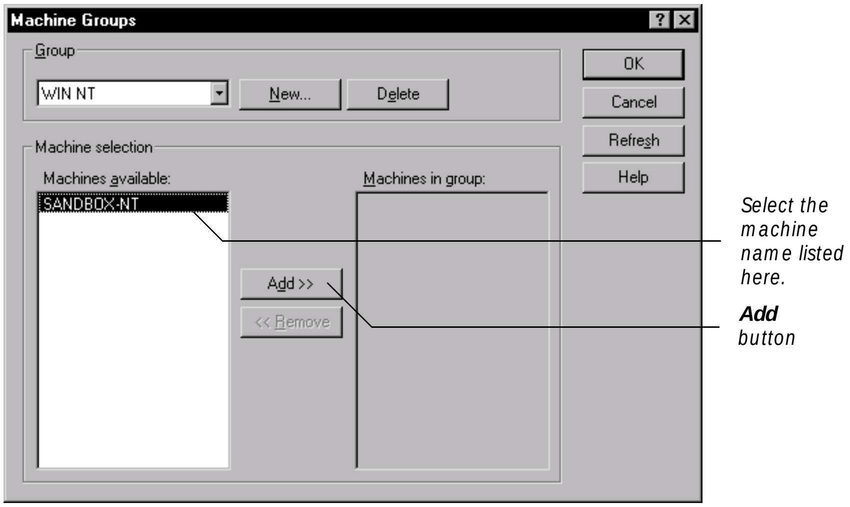


3. In the **Name** box, type a name such as **Win NT** or **Win 2000** for the machine group.

4. Click **OK**.

   The **Machines available** box displays the name of the Test Lab machine(s) that you set up for testing.

5. To add the Test Lab machine to the group, select its name, and then click **Add**.



6. Click **OK**.

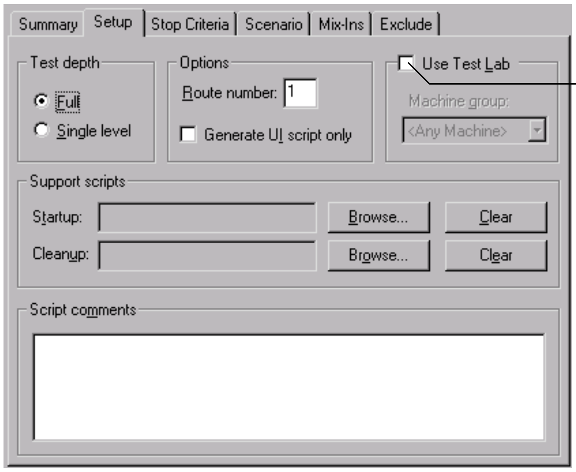
If you have installed and set up TestAccelerator on additional Test Lab machines, you can add these to the machine group that you created or you can create additional machine groups for them.

Now that you've successfully set up TestAccelerator and TestFactory, you're ready to run remote tests.

Running a Pilot on a Test Lab Machine

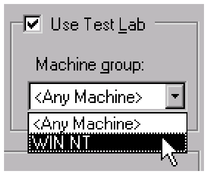
To run a Pilot on a Test Lab machine:

1. In the application map, click a Pilot object that you created in the exercises in Example 3.
2. Click the **Setup** tab.



Use Test Lab *check box*

3. Select the **Use Test Lab** check box.



4. In the **Machine group** list, select the name of the machine group that you created.
5. Click **Start**.

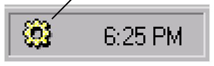
After you start the Pilot run, the Pilot progress bar displays run information at the bottom of the screen as the Pilot distributes scripts to the Test Lab machine. Although you can't continue to work in TestFactory, you can use the machine for other tasks during the Pilot run.

Once the Pilot run is completed, the restored TestFactory window displays summary results for the Pilot run in the **Summary** tab in the right pane. You can examine the Pilot run results just as you would if you had run the Pilot locally.

Quitting TestAccelerator

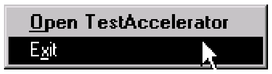
To quit TestAccelerator on the Test Lab machine after testing is completed:

1. In the status area of the Windows taskbar, right-click the TestAccelerator program applet.



TestAccelerator program applet in the status area of the taskbar.

2. Click **Exit** on the shortcut menu.



Summary

In the exercises in this example, you learned how to set up Rational TestAccelerator on a Test Lab machine, and then run a Pilot on the Test Lab machine from TestFactory.

For information about running Test Suites on Test Lab machines, see Chapter 6, *Developing and Running a Test Suite* in the *Using Rational TestFactory* manual. For information about running test objects listed in the AutoPilot on Test Lab machines, see Chapter 7, *Using the AutoPilot* in the *Using Rational TestFactory* manual.

What's Next

The next example shows you how to set up the TestCodeChanges add-in for Visual Studio, and then use the add-in to test changes that you make to Classics source code.

▸ ▸ ▸ EXAMPLE 8

Testing Changes to Source Code

Objectives

- Set up the TestCodeChanges add-in for Visual Studio.
- Start Visual Basic and make changes to Classics source code.
- Start the TestCodeChanges add-in.
- Run scripts to test changed source code files.

Scenario

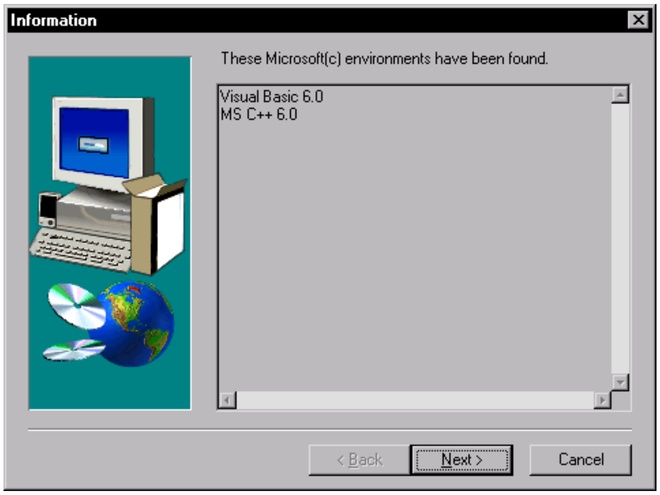
In this example, you'll learn how to set up the TestCodeChanges add-in for Visual Studio and use the add-in to test changes that you make to Classics source code.

Setting Up the TestCodeChanges Add-In for Visual Studio

After you set up the TestCodeChanges add-in, it is loaded in your Visual Studio development environment where it automatically tracks changes made to the source code files for the open project.

To set up the TestCodeChanges add-in:

1. Click **Start** → **Programs** → **Rational Suite TestStudio** → **Rational Test** → **Set Up Rational TestCodeChanges**.

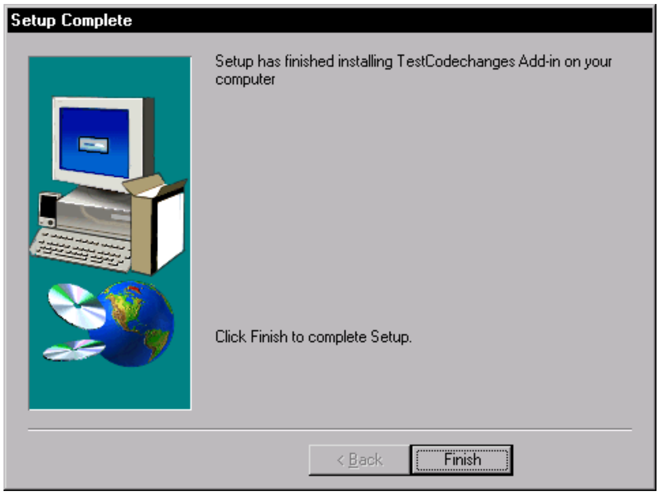


These Microsoft(c) environments have been found.

Visual Basic 6.0
MS C++ 6.0

The Information dialog box lists the Microsoft development environments detected on your system.

2. To continue with the setup, click **Next**.



Setup has finished installing TestCodechanges Add-in on your computer

Click Finish to complete Setup.

3. To complete the setup, click **Finish**.

Changing Source Code in Classics

In the following exercise, you'll change the source code for the Classics application, recompile the ClassC.exe file, and then save the changed source files.

To open the ClassicC.vbp file in Visual Basic 6:

1. Start Visual Basic 6.
2. In the New Project dialog box, click the **Existing** tab.
3. Browse to the following directory:

   \\Rational\Rational Test 7\Sample Applications\Classics Online\classics source
4. Open the ClassC.vbp file.

To change Classics source code:

1. In the Project window, expand the **Forms** folder.
2. Double-click **frmMain**.
3. On the frmMain form, click the Order It! button object.
4. In the Properties window, locate the **Caption** property, select the current value (Order It!) and type **Order Now!**
5. In the Project window, double-click **frmOrder**.

6. On the frmOrder form, click the Place Order button object.
7. In the Properties window, locate the **Caption** property, select the current value (Place Order) and type **Submit Order**.

To make the executable file and save the project:

1. Click **File** → **Make ClassicsC.exe**.
2. In the Make Project dialog box, click **OK**.
3. To confirm that you want to replace the existing executable file, click **Yes**.
4. To save your changes and the ClassicsC.exe file, click **File** → **Save Project**.

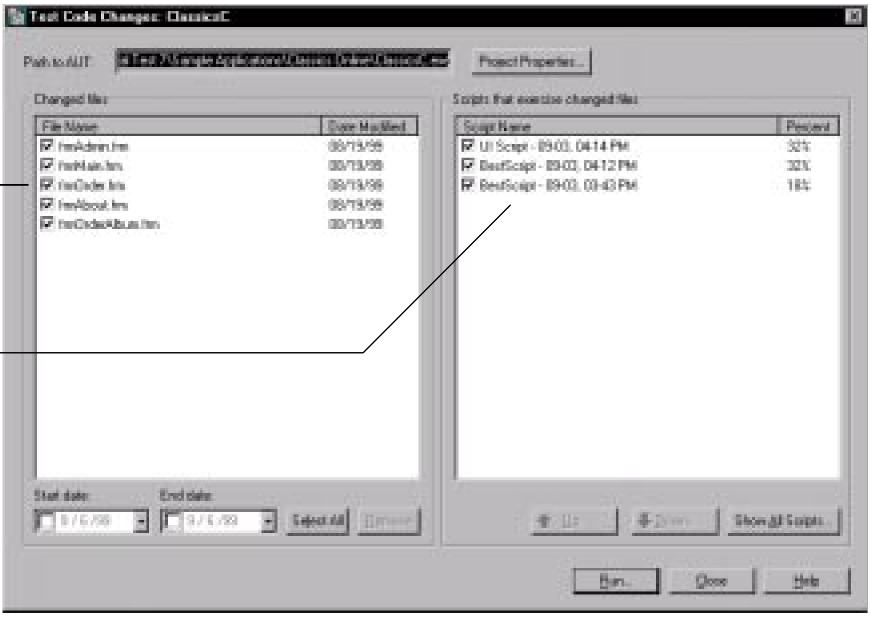
Testing Changes to Source Code Files

Now that you've made changes to Classics source code, you can start the TestCodeChanges add-in and run scripts that test the changed files.

Starting the TestCodeChanges Add-In

To start the TestCodeChanges add-in and log on to the CLASSICS project:

1. On the Visual Basic toolbar, click **TestCodeChanges**.
2. In the Rational Repository Login dialog box, click **OK**.



File Name	Date Modified
✓ frmAdmin.frm	08/19/98
✓ frmMain.frm	08/19/98
✓ frmOrder.frm	08/19/98
✓ frmAbout.frm	08/19/98
✓ frmOrderAlbum.frm	08/19/98

Script Name	Percent
✓ UI Script - 09/03, 04:14 PM	32%
✓ BestScript - 09/03, 04:12 PM	32%
✓ BestScript - 09/03, 03:43 PM	18%

All changed project files are listed in the left pane.

Scripts that exercise the changed files (and that have code coverage values) are listed in the right pane.

The **Changed files** list displays the names of all of the changed source code files and the date on which each was changed. Changed files are listed in descending order of the date they were changed, starting with the most recently changed file.

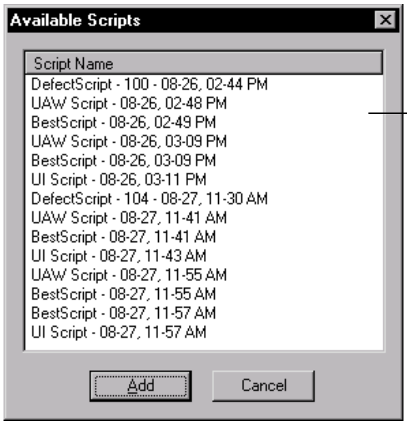
The **Scripts that exercise changed files** list displays all of the project scripts that exercise the files listed on the left, and for which TestFactory calculated code coverage. The value displayed in the **Percent** column indicates the percent of source code in the changed file that the script exercises. If a listed script exercises two or more of the changed source files, then its Percent value represents an average percent of code that the script exercises in all of the changed files it touches.

The order of a script in the list is based on the amount of code coverage it provides for the changed file relative to the other scripts. The script that provides the highest code coverage is listed first.

To add a script that does not exercise a changed file, or does not have a code coverage value, to the regression suite:

1. To view all available scripts, click **Show All Scripts**.

   The Available Scripts dialog box lists all of the scripts that you can add to the regression suite.



Script Name
DefectScript - 100 - 08-26, 02-44 PM
UAW Script - 08-26, 02-48 PM
BestScript - 08-26, 02-49 PM
UAW Script - 08-26, 03-09 PM
BestScript - 08-26, 03-09 PM
UI Script - 08-26, 03-11 PM
DefectScript - 104 - 08-27, 11-30 AM
UAW Script - 08-27, 11-41 AM
BestScript - 08-27, 11-41 AM
UI Script - 08-27, 11-43 AM
UAW Script - 08-27, 11-55 AM
BestScript - 08-27, 11-55 AM
BestScript - 08-27, 11-57 AM
UI Script - 08-27, 11-57 AM

All of the scripts that you can add to the regression suite are listed in the Available Scripts dialog box.

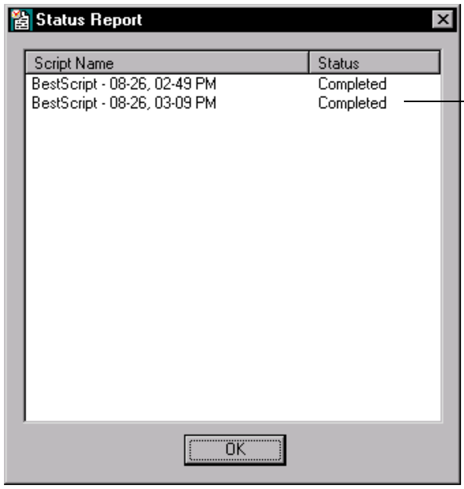
2. To add a listed script to the suite, click the script name, and then click **Add**.
3. To add a multiple scripts, hold down the CTRL or SHIFT key, select multiple script names, and then click **Add**.
4. The add-in runs scripts in their listed order. To change the run order for a script, click its name, and then use **Up** and **Down** to change its position in the list.

5. To start testing your code changes, click **Run**.

After you start the run, the add-in window closes, Robot starts, the AUT starts, and the Script progress bar displays testing status at the bottom of the screen.

Viewing Run Results

After the regression suite run is completed, the Status Report dialog box displays the run status for each script in the **Status** column. A **Completed** status indicates that the run completed and that the script encountered no defects. A **Failed** status indicates that the script encountered one or more defects (unless the script run was interrupted).



Script Name	Status
BestScript - 08-26, 02-49 PM	Completed
BestScript - 08-26, 03-09 PM	Completed

If the run results for scripts that exercise a changed file for your project are successfully completed, you can check in the modified code. If a run result fails, examine the logs, make necessary changes to the source code, and then rerun the regression suite.

To close the Status Report dialog box and quit TestCodeChanges, click **OK**.

Summary

In these exercises, you used the new TestCodeChanges add-in to access and run scripts that tested changes you made to Classics source code.

▸ ▸ ▸ Index