Using
Rational TestFactory

VERSION 2001.03.00

PART NUMBER 800-023884-000

Rational®
the **e-development** company™

▸ ▸ ▸ Contents

2   The TestFactory Interface

3   Instrumenting the Application-Under-Test

4 Developing and Working with the Application Map

7 Using the AutoPilot

8 Testing Code Changes in Visual Studio

9 Using TestFactory Command-Line Arguments

▸ ▸ ▸ Preface

Rational TestFactory® is the next-generation software quality tool that automatically generates scripts for testing applications written in Microsoft Visual Basic, C++, and Java, as well as Java applets. TestFactory amplifies the productivity of developers and testers by reducing the manual effort required to test software. Because it models an application, builds regression test suites, and finds defects, TestFactory is easily adopted at any phase in the development cycle. Scripts that flush out defects and provide extensive product coverage can be generated as soon as a user interface is available to test.

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

This manual explains how to use Rational TestFactory and create, analyze, and manage automated test procedures.

Audience

This manual is intended for application developers, quality assurance managers, and quality assurance testers.

Other Resources

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

▶ All manuals are available online, either in HTML or PDF format. These manuals are on the *Rational Solutions for Windows* Online Documentation CD.

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

Contacting Rational Technical Publications

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

Contacting Rational Technical Support

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

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

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

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

▸▸▸ C H A P T E R   1

Introduction

This chapter provides an overview of Rational TestFactory. It introduces some key concepts, describes the tasks that you can use TestFactory to perform, and provides information about what to do before you start working with your Java, C++, or Visual Basic application, or your Java applet in TestFactory. This chapter includes the following topics:

▸ About Rational TestFactory

▸ Key TestFactory concepts

▸ Testing software with TestFactory

▸ Preparing to work with the application-under-test in TestFactory

About Rational TestFactory

Rational TestFactory automates software quality testing to an unprecedented extent. By substantially reducing the manual intervention required to create and maintain testing assets, TestFactory significantly shortens the product development cycle. TestFactory maps the application-under-test, uncovers severe defects, and creates scripts that provide extensive source code coverage. In addition to its automatic mapping and scripting capabilities, TestFactory provides a flexible interface with tools that help you organize and manage project test assets. The test assets include the scripts, Test Suites, folders, reports, and so on, that you add to your project.

TestFactory handles much of the drudgery involved in software testing so you can focus on planning, development, and requirements testing. The high level of automation that TestFactory provides lets you incorporate automation early in the testing cycle; you do not have to wait until the application-under-test stabilizes.

TestFactory is integrated with the following Rational Suite TestStudio™ components:

- ► Rational Robot
- ► Rational TestManager
- ► Rational Administrator
- ► Rational ClearQuest
- ► Rational TestCodeChanges add-in for Visual Studio

Together, these products provide a full array of tools for team testing within Windows® NT®, Windows 95®, Windows 98®, and Windows 2000® environments.

TestFactory offers the following features:

- ► Automatically creates and maintains a detailed map of controls and actions in the user interface of the application-under-test.
- ► Lets you map and test multiple paths in a functional area of an application.
- ► Automatically generates scripts that provide maximized product coverage.
- ► Tracks executed and unexecuted source code and reports its findings.
- ► Automatically generates regression Test Suites containing scripts that uncover serious defects in the application-under-test.
- ► Simplifies maintenance of test assets, which means that testing new builds requires minimal supervision.
- ► Lets you organize scripts into Test Suites and run them as batch jobs.
- ► Lets you 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 TestFactory Test Suites.
- ► Together with the TestCodeChanges 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.

Key TestFactory Concepts

The Application Map

One of the first tasks that you perform in a new TestFactory project is to map the application-under-test (the **AUT**) using the Application Mapper. The Application Mapper explores the AUT to produce a detailed, hierarchical **application map** that graphically depicts all of the controls in the user interface and the actions used to exercise them. The application map models all possible states of the AUT and the transitions between those states.

As it remaps new builds, the Application Mapper looks for, flags, and resolves changes in the AUT. For information about application mapping and application map components, see Chapter 4, *Developing and Working with the Application Map*.

The UI Library

The **UI library** is an archive that contains all classes and subclasses of objects that commonly occur in the user interface of an application. When you open a project in TestFactory for the first time, TestFactory builds the UI library and places it in the UI Library folder. TestFactory uses the UI library to identify and reconcile objects in the AUT interface during the mapping process. Controls identified in the AUT are represented in the application map by user interface objects, or UI objects.

UI Objects

UI objects are the objects in the application map that represent the controls in the user interface of the AUT. Each UI object is an instance of a UI object class in the UI library. UI objects and their properties are described in Chapter 4, *Developing and Working with the Application Map*.

Pilots and Automatic Script Generation

In TestFactory, you use **Pilots** to automatically generate scripts. A Pilot uses the application map to build scripts that test the AUT. You can drop a Pilot at any functional area of the application map. From there, the Pilot generates scripts that go progressively deeper into the source code for the area of the AUT to which you give it access.

After a script is generated, it is independent of the Pilot that generated it. You can move or copy the script into a folder and include it in Test Suites. For information about using Pilots to automatically generate scripts, see Chapter 5, *Automatically Generating Scripts*. For information about Test Suites, see *Test Suites* on page 1-5.

Script Segments

A Pilot uses the application map to create **script segments**, which are the building blocks for scripts. A script segment consists of a sequence of operations that TestFactory performs in the AUT.

The Best Script

Each Pilot run produces, at most, one **best script**. To generate the best script, the Pilot first creates and runs a large number of script segments. The Pilot then evaluates every script segment based on the amount of user interface coverage and source code coverage each provides. The resulting best script provides high coverage of the AUT source code and user interface using the smallest possible amount of nonredundant script code.

Defect Scripts

The script segments that a Pilot creates sometimes uncover severe defects such as AUT crashes, run-time errors, assertion failures, as well as defects related to memory access. If a script segment uncovers a defect, the Pilot retains it as a single-segment **defect script** in the Pilot run results.

UAW Scripts

During a Pilot run, if script segments exposed controls in the user interface that are not represented in the application map, TestFactory saves the script segments in a single **UAW** (unexpected active window) **script**. After a Pilot run, you can use the UAW script to trace the steps that the Pilot took to reach unmapped controls, and use that information to improve the application map.

The UI Script

Each Pilot run generates one **UI script** that "attempts" to touch every control in the tested area once. The UI script serves as a simple smoke test that you can run to check the controls in a region of the AUT user interface. You can run a Pilot to quickly generate just a UI script.

Code Coverage and UI Coverage

TestFactory calculates **code coverage** and **UI coverage** metrics for the best script, defect scripts, and Robot scripts that you run against an instrumented AUT. The coverage values indicate how well a script exercises the AUT and help you determine which features to test next. Code coverage 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. For information about instrumentation, see Chapter 3, *Instrumenting the Application-Under-Test*.

Test Suites

A **Test Suite** is a container object that you can insert in the application map to group scripts and other Test Suites to run as a batch job. A Test Suite is also a convenient tool for organizing and tracking scripts. For information about creating and running a Test Suite, see Chapter 6, *Developing and Running a Test Suite*.

The AutoPilot

The **AutoPilot** is the TestFactory tool that automatically runs multiple Pilots, Test Suites, and scripts as a batch job. You can use the AutoPilot to run batch tests on your local machine. For information about using the AutoPilot, see Chapter 7, *Using the AutoPilot*.

The Test Datastore

All TestFactory project data is contained in the **test datastore**—which stores application testing information. After you create a new project in the Rational Administrator, you can see and access the project from all TestStudio programs. For information about working with projects and the test datastore, see the *Using the Rational Administrator* manual.

Testing Software with TestFactory

How you use TestFactory depends on whether you are the sole engineer on a project or a member of a QA team, as well as the testing practices you have in place. How you use TestFactory also depends on where you are in the testing cycle. For instance, in the early phases of the product development cycle, you can use TestFactory to automatically find defects in the AUT. As the AUT matures and stabilizes, you can use TestFactory to build regression suites. The tasks that you might perform in a typical TestFactory project are as follows:

Task	Manual Chapter
Instrument the AUT so that TestFactory can calculate code coverage for scripts.	Chapter 3: *Instrumenting the Application-Under-Test*
Use the Application Mapper to map the AUT.	Chapter 4: *Developing and Working with the Application Map*
Run several Pilots in functional areas of the AUT to generate best scripts and defect scripts that test those areas.	Chapter 5: *Automatically Generating Scripts* Chapter 7: *Using the AutoPilot*
Examine failures uncovered by the defect scripts.	Chapter 5: *Automatically Generating Scripts*
Examine the outlined steps and detailed code coverage results for best scripts.	Chapter 5: *Automatically Generating Scripts*
Create a Test Suite that contains pointers to all of the best scripts and defect scripts and run the Test Suite against future builds of the AUT.	Chapter 6: *Developing and Running a Test Suite*
Create customized TestFactory scripts to augment the coverage your Test Suites provide.	Chapter 5: *Automatically Generating Scripts*
Use the TestCodeChanges add-in for Visual Studio to access and run scripts that test changes to the source code of the AUT.	Chapter 9: *Testing Code Changes in Visual Studio*
Run Pilots to test the AUT for defects related to memory access.	Chapter 5: *Automatically Generating Scripts*

With each new build, you instrument the AUT, update the application map, run scripts (regression Test Suites) from the previous build, track and resolve defects, create and run new scripts, and reorganize test material.

Preparing for Automated Testing

To optimize the use of TestFactory, it is a good idea to begin by organizing the following information:

▶ Assess the AUT thoroughly. Understand the various system states it assumes, as well as the transitions between the states.

▶ Determine which areas of the AUT are the most stable and focus your testing efforts on these early in the development cycle. Identify functional areas that do not work and exclude them from initial testing.

▶ Plan the order in which functional areas are to be tested based on feature availability, stability, and risk.

▶ Make sure that you fully understand how functional areas of the AUT are designed to interact.

▶ Define data entry types (zip codes, phone numbers) and specific values (for example, passwords) for key controls and try to anticipate how they might affect the AUT during testing.

▶ Define typical user scenarios that you can simulate in TestFactory by creating Pilot scenarios.

▶ Devise a scheme for naming the test assets (scripts, Pilots, Test Suites, folders, and so on) you develop in TestFactory.

▶ Break down your testing product into manageable units or modules and assign these to appropriate team members.

▶ Make sure that you have standard methods in place for organizing project material.

Before You Map the AUT

Decide whether to map the AUT fully in one pass, or incrementally. Several factors can influence your decision, including the size and complexity of the AUT and the presence of unstable source code. For most applications, we recommend that you map incrementally. Incremental mapping makes it easier for you to control the direction and actions the Application Mapper takes through different functional areas of the AUT. For information about incremental mapping, see Chapter 4, *Developing and Working with the Application Map*.

After You Map the AUT

After a mapping session, start the AUT and compare it to the application map. Take steps to correct inaccuracies before you begin generating scripts. The quality of the scripts you generate is directly related to the quality of the application map. For information about analyzing and preparing the application map for testing, see Chapter 4, *Developing and Working with the Application Map*.

Testing the AUT

After you create and fine-tune the application map, you can:

▶ Insert and run Pilots in functional areas of the AUT to uncover severe defects.

▶ Compose Pilot scenarios that simulate user action sequences in the AUT.

▶ Compose Pilot mix-ins to test the random interaction of different functional areas in the AUT.

▶ Create Test Suites to organize and track the scripts that your Pilots generate.

▶ Use the AutoPilot to automatically run multiple Pilots, Test Suites and scripts as a batch job — on your local machine.

▶ If the AUT is written in Visual Basic or Visual C++, use the TestCodeChanges add-in for Visual Studio to run scripts that test the changes that you have made to the AUT source code files.

Preparing to Work with the Application-Under-Test in TestFactory

Before you begin working with the AUT in TestFactory or Robot, you must first "enable" the AUT, and then specify the correct IDE (integrated development environment) extensions to load for it in Robot. This section describes what enabling does for Java applications and applets and C++ applications, and provides instructions on how to specify the IDE extensions to load in Robot.

Enabling Java and C++ Applications

Rational Test Enablers provide specialized support for mapping and testing controls in Java and C++ applications and in Java applets. To successfully map and test controls, you must do the following before you start working with the AUT in TestFactory.

▶ Install the Rational Test Enabler for your IDE. The following table shows the correct Rational Test Enabler to install for each IDE.

IDE	Rational Test Enabler to Install
Java	Rational Test Java Enabler
C++	Rational ActiveX Test Control

▶ Open the AUT in the development environment and add the ActiveX Test Control to the forms.

You can install the Rational Test Enablers from the Rational Software Setup program. For information about how to install the Rational Test Enablers, see the *Rational Suite Installation Guide*.

Enabling Java Applications and Applets

Before you work with a Java application or applet in TestFactory, run the Java Enabler to have it search your hard drive for Java environments such as Web browsers and Sun JDK installations that Robot supports. The Java Enabler only enables environments that are currently installed. For more information, see the *Using Rational Robot* manual.

Enabling C++ Applications

To test the properties and data of ActiveX controls in C++ applications, manually add the ActiveX Test Control to each OLE container (Window) in your application. For information about how to add the ActiveX Test Control, see the documentation that comes with your C/C++ development environment.

Selecting the IDE Extensions to Load for the AUT

Before you can work with a Visual Basic application, Java application, or Java applet in TestFactory or Robot, you must start Robot and select the IDE extensions to load for the programming language. (IDE extension support for C++ is always loaded.) This ensures that the correct extensions are loaded for mapping and testing the AUT.

To select the IDE extension to load:

1. Click **Start → Programs → Rational Suite TestStudio → Rational Robot**.

2. Log on to the project.

3. Click **Tools → Extension Manager**.
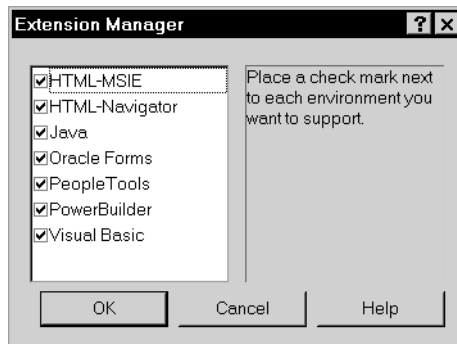
4. Leave the **Visual Basic** or **Java** check box selected. To improve performance, clear the remaining check boxes.

5. Click **OK**.

6. To close the message box, click **OK**.

7. Quit Robot.

Once you enable your application and specify the IDE extensions to load, you can starting working with it in TestFactory.

Working with C++ Applications in TestFactory and Robot

If you are working with an application written in C++, we recommend that you change the default **object recognition method order** setting in Robot before you start to work with the application in TestStudio. The default recognition method order that is set in Robot optimizes control recognition in Visual Basic applications. You can change this setting to optimize control recognition in C++. This improves the quality and stability of the scripts you generate to test the C++ application as changes are made to the user interface of the AUT during development.

To change the object recognition method order setting in Robot:

1. Click **Start > Programs > Rational Suite TestStudio > Rational Robot**.

2. Log on to the project.

3. Click **Tools > GUI Record Options**, and then click the **Object Recognition Order** tab.

4. In the **Object order preference** list, click **C++ Recognition Order**.

5. Click **OK**.

6. Quit Robot.

For information about Robot recognition methods, see *Selecting an Object Order Preference* in the *Using Rational Robot* manual.

▸ ▸ ▸ C H A P T E R   2

The TestFactory Interface

This chapter explains how to start TestFactory and describes the basic elements of the user interface. This chapter includes the following topics:

▸ Starting TestFactory

▸ Using the New Project Wizard

▸ Using the Project Assistant

▸ The TestFactory workspace

Starting TestFactory
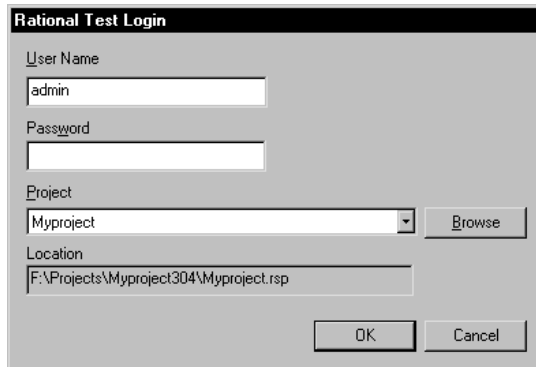
Before you start TestFactory for the first time, you must start the Rational Administrator, and create a project. For information about creating a project, see the *Using the Rational Administrator* manual.

Before you start TestFactory, quit all applications on the desktop. During mapping and testing, TestFactory actively explores the AUT. Other programs that are running can interfere with these processes.

To start TestFactory:

1. Click **Start > Programs > Rational Suite TestStudio > Rational TestFactory.**

2. In the **User Name** and **Password** boxes, type your user name and password.

3. In the **Project** list, select the project.

4. In the **Location** box, check to make sure that is the correct location.

5. Click **OK**.

TestFactory initializes the project, builds a UI library of object classes, and places the Application Map folder, the Robot Scripts folder, and the UI Library folder in the left pane of the window.

Starting TestFactory from the Command Line

You can start TestFactory from the command line by typing a command in the Run dialog box, or by adding the command-line argument in the Properties dialog box of Windows Explorer. If you start TestFactory from the command line, you can specify arguments to control the behavior of the program on start-up. You can also specify command-line arguments in the Properties dialog box for a TestFactory shortcut. For a list of command-line arguments supported by TestFactory and for information about using them, see the Appendix *Using TestFactory Command-Line Arguments* in this manual.

Using the New Project Wizard

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

1. Read the information in step 1 of the New Project Wizard, and then click **Next**.
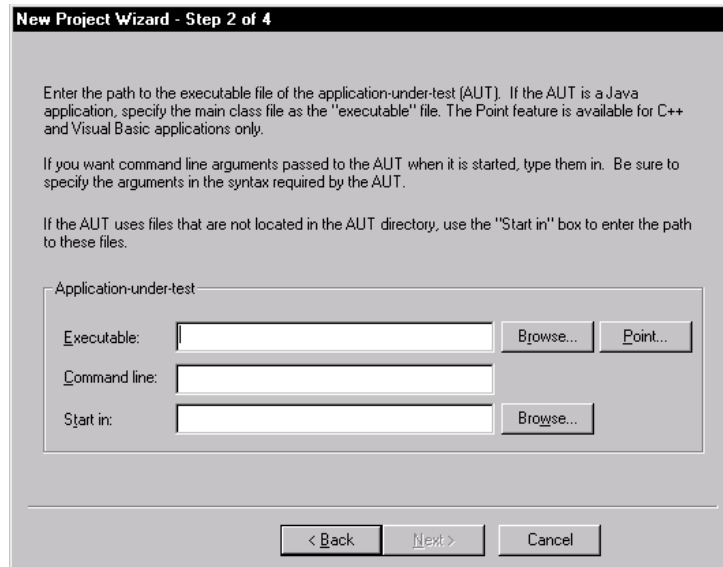
2. In the **Executable** box in step 2, enter the path to the executable file for the AUT. (You can specify an executable file that has an .exe , .class, .jar, .htm, .html, or .bat extension.)

3. To pass one or more command-line arguments to the AUT when it is started from TestFactory, type the argument in the **Command line** box. Be sure to specify the command-line argument in the syntax that the AUT requires.

4. In the **Start in** box, enter the full path to the working directory. If you do not enter a start path, TestFactory uses the AUT executable path.

5. Click **Next**.
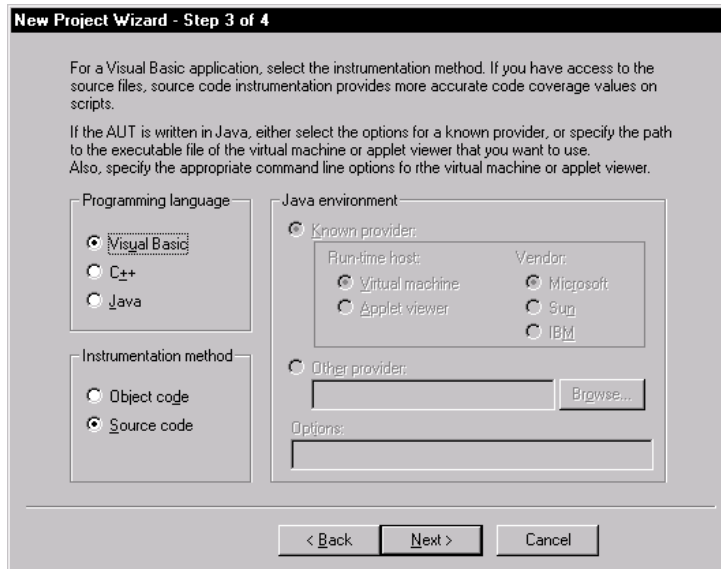
6. Under **Programming language**, check to see that the correct programming language for the AUT is selected. If it is not, click the correct programming language.

7. If the AUT is a Visual Basic application and you plan to instrument its source code files, then under **Instrumentation method**, leave **Source code** selected. If you plan to instrument the object code instead, click **Object code**.

Instrumenting the AUT gives TestFactory the information it needs to determine how well your scripts exercise the AUT source code during testing. If the AUT is a Visual Basic application, and you have access to source code, you can instrument it using either the object code method or the source code method. If the AUT is written in Java or C++, you must instrument it using the object code method.

If the AUT is a Visual Basic application and you have access to its source code files, we recommend that you select **Source code**. For information about the differences between object code and source code instrumentation, see Chapter 3, *Instrumenting the Application-Under-Test*.

8. If the AUT is a Java application or applet, then under **Java environment**, do the following:

– If the AUT is a Java application, then under **Runtime host**, leave **Virtual machine** selected.

– If the AUT is a Java applet, then under **Runtime host**, click **Applet viewer**.

– If you plan to run a Java application or applet using a Microsoft virtual machine or applet viewer, then under **Vendor**, leave **Microsoft** selected.

– If you plan to run a Java application or applet on a Sun virtual machine or applet viewer, then under **Vendor**, click **Sun**.

– If you plan to run a Java application or applet on an IBM virtual machine or applet viewer, then under **Vendor**, click **IBM**.

– If you are running a Java AUT on a virtual machine or applet viewer from a provider other than Microsoft, Sun, or IBM, click **Other provider**, and then browse to find and select the virtual machine or applet viewer.

9. If the AUT is a Java application or applet, then in the **Options** box, type arguments to pass to the Java virtual machine or applet viewer.

10. Click **Next**.


```
New Project Wizard - Step 4 of 4
```

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.

< Back   Finish   Cancel

11. To complete project initialization and close the wizard, click **Finish**.

Using the Project Assistant

After the New Project Wizard closes, the **Project Assistant** opens on top of the TestFactory window. You can use the Project Assistant to get information about how to instrument the AUT, map the user interface, and run a Pilot.

Click here to collapse the Project Assistant window.

Click here to learn how to instrument an application.

Click here to learn how to map an application.

Click here to learn how to insert and run a Pilot.
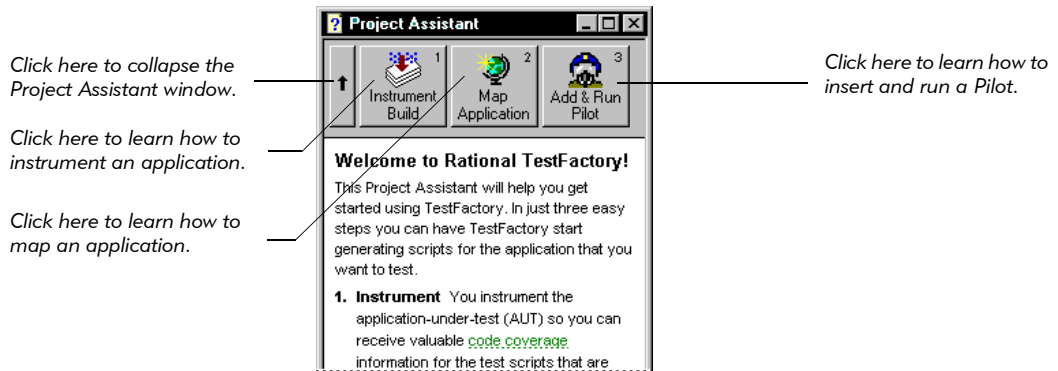
The following table shows how to use the buttons in the Project Assistant window.

Click	To
Instrument Build	Learn how to instrument the AUT
Map Application	Learn how to map the AUT
Add & Run Pilot	Learn how to insert and run a Pilot in the application map
↑	Display just the toolbar of the Project Assistant window
↑	Restore the full Project Assistant window

Every time you start TestFactory, the Project Assistant opens by default. To prevent the Project Assistant from opening after you start TestFactory:

1. Click **Tools → Options**.
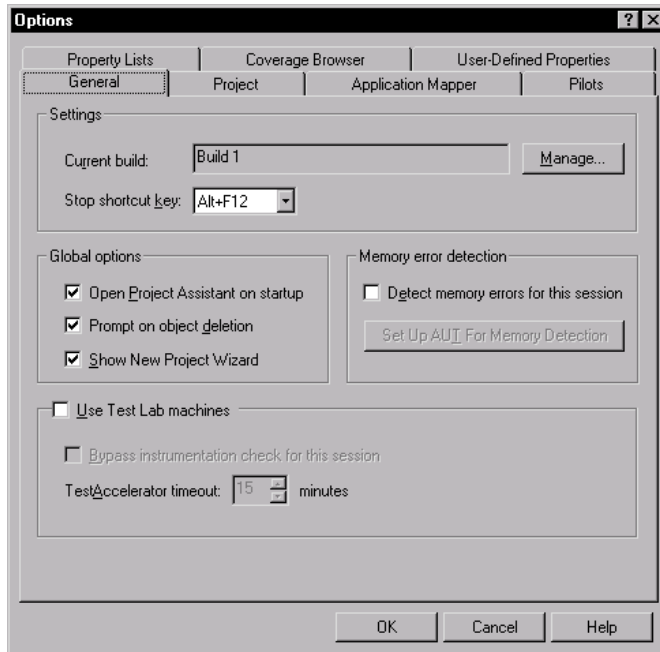
2. On the **General** tab, under **Global options**, clear the **Open Project Assistant on startup** check box.

3. Click **OK**.

The TestFactory Workspace

TestFactory features an intuitive, easy-to-navigate user interface. The following figure shows the TestFactory window for an open project.
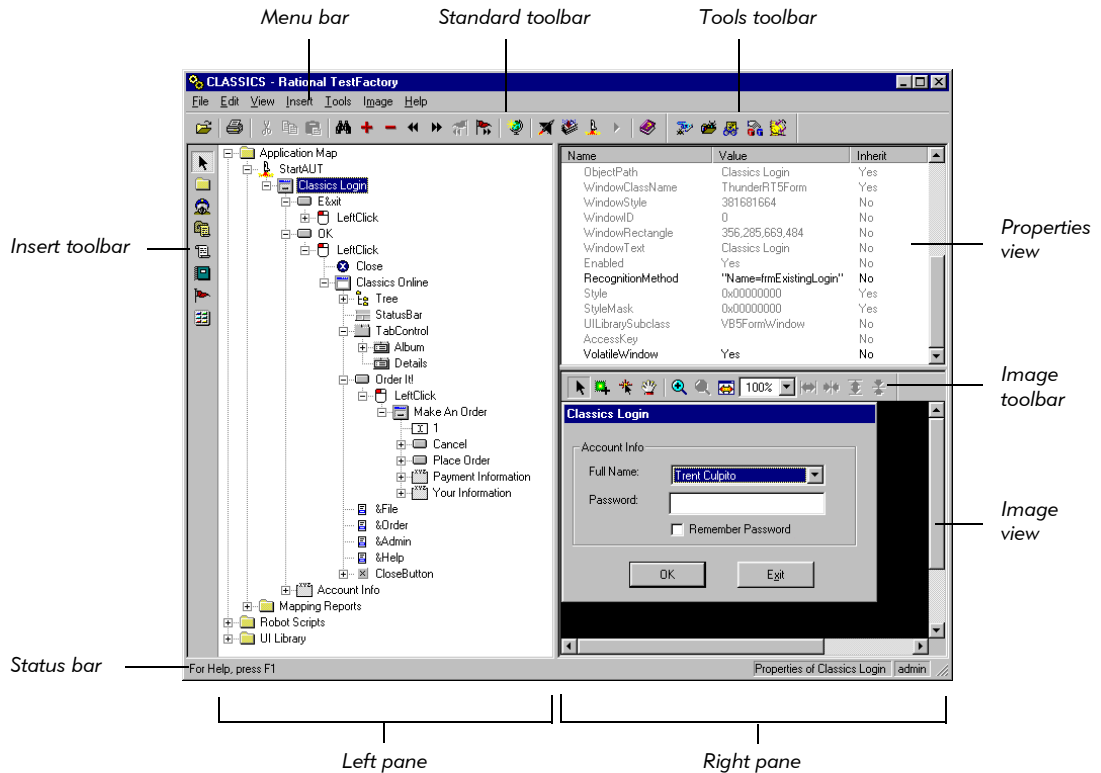
This section describes the components of the TestFactory window.

TestFactory Toolbars

The TestFactory window has seven separate toolbars. This section describes each toolbar.

To toggle the display of a toolbar:

▶   Click **View**, point to **Toolbars**, and then click the toolbar name.

NOTE: The Insert toolbar is always visible; you cannot toggle its display.

The Standard Toolbar

The **Standard toolbar** is displayed under the menu bar. A Standard toolbar button is available only after you select an appropriate object in the TestFactory window. For example, **Play Back Script** is available only after you select a script object in the application map.

Use the Standard toolbar buttons to do the following:

Click	To
	Close the open project and open a different project.
	Print the properties of a selected object in the application map or print information displayed in the right pane.
	Remove a selected object from the application map.
	Copy the selected object in the application map.
	Paste a copied object to a selected location in the application map.
	Find objects in the application map.
	Fully expand the branch under the object selected in the application map.
	Fully collapse the branch under the object selected in the application map.

Click	To
	Jump to the object previously selected in the application map.
	Jump to the next object in a sequence of selected objects in the application map.
	Jump to a marker that you inserted higher up in the application map hierarchy.
	Jump to a marker that you inserted lower in the application map hierarchy.
	Open the Application Mapper Wizard.
	Open the AutoPilot window.
	Open the Instrument Source Code or Instrument Object Code dialog box.
	Start the application-under-test.
	Play back a selected script in Robot.
	Access TestFactory Help.

The Insert Toolbar

The **Insert toolbar** is on the left side of the TestFactory window. It provides buttons that you can use to insert several types of **TestFactory objects** in the application map. Except for **Select**, each button corresponds to a command on the **Insert** menu.

Use the Insert toolbar buttons to do the following:

Click	To
	Restore the pointer to the Select Item mode.
	Insert a folder to organize project material. After TestFactory initiates a project, it inserts the Application Map folder, the Robot Scripts folder, and the UI Library folder in the left pane. A Pilot run inserts a run results folder. You can insert folders to hold best scripts, defect scripts, Test Suites, or TestFactory reports.
	Insert a Pilot to generate scripts for a functional area of the AUT. A Pilot proactively explores the UI objects to which it has access, automatically detects severe defects in the AUT, and generates defect scripts and a coverage-optimized best script. For information about setting up and running a Pilot, see Chapter 5, *Automatically Generating Scripts*.
	Insert a Test Suite to group scripts and other Test Suites to run as a batch job. For details on creating and running a Test Suite, see Chapter 6, *Developing and Running a Test Suite*.
	Insert a script to create a customized TestFactory script in the application map that you can then record or write manually in Robot. Although TestFactory gives you access to scripts created in Robot, the customized TestFactory script is useful if you want to place the script in a TestFactory folder.
	Insert a report to create a TestFactory report.
	Insert a marker as a place-holder in the application map or to capture comments on a specific area of the application map.
	Insert an interaction object to control the path that TestFactory takes through the AUT during mapping and testing.

The Tools Toolbar

Use the **Tools toolbar** to start other Rational TestStudio programs. Buttons on the Tools toolbar correspond to commands in the **Rational Test** submenu of the **Tools** menu.

Use the Tools toolbar buttons to do the following:

Click	To
	Start Rational Robot.
	Start Rational TestManager.
	Start the Rational Administrator.
	Start Rational ClearQuest.

The Report Toolbar

TestFactory displays the **Report toolbar** after you click a report object in the left pane. Buttons on the Report toolbar correspond to commands in the **Report** menu.
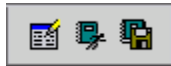
Use the Report toolbar buttons to do the following:
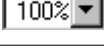
Click	To
	Edit the report parameters before rerunning the report.
	Run the selected report.
	Export the selected report as a text file.

The Image Toolbar

TestFactory displays the **Image toolbar** in the Image view after you select a UI object in the application map.

Use the Image toolbar buttons to do the following:

Click	To
	Select, resize, or reposition a region tracker in the Image view.
	Draw the area for a region object in the Image view.
	Reposition the responsive region, or hot spot, of a region object or other UI object.
	Move the image in the Image view.
	Increase the magnification of the displayed image.
	Decrease the magnification of the displayed image.
	Toggle between sizing the image to fit in the Image view and viewing the image at the currently selected magnification.
100% ▼	Display the image in the Image view at a selected magnification.
	Expand the region object tracker horizontally.
	Shrink the region object tracker horizontally.
	Expand the region object tracker vertically.
	Shrink the region object tracker vertically.

The Interaction Object Toolbar

TestFactory displays the **Interaction Object toolbar** in the top right pane after you insert or click an interaction object in the application map.

Use the Interaction Object toolbar buttons to do the following:

Click	To
⬆	Move the selected component up in the list to change its interaction order.
⬇	Move the selected component down in the list to change its interaction order.
	Insert a component in the interaction object.
✕	Delete the selected component from the interaction object.
⊘	Make a component unavailable for mapping and testing in the interaction object.
	Edit the data entry style for the selected interaction object component.

The Style Toolbar

After you click an edit box, combo box, or region object in the left pane, TestFactory displays the **Style toolbar** in the top right pane.

Use the Style toolbar to do the following:

Click	To
Sty	Select an existing data entry style for the selected object.
	Open the Edit Data Entry Style dialog box to edit an existing entry style or to create a new entry style for the selected object.

The Left Pane

The left pane is the primary workspace in TestFactory. TestFactory places the **Application Map folder**, the **Robot Scripts folder**, and the **UI Library folder** in the left pane after you open a project for the first time.

Until you map the AUT, the Application Map folder is empty. After you map the application, the folder contains all of the AUT components, including various UI objects, user actions, and transition elements.

The Robot Scripts folder holds project scripts created in Robot. You can include Robot scripts in your Test Suites and run them with TestFactory scripts. TestFactory can calculate UI coverage and code coverage values for your Robot scripts.

The UI Library folder contains all of the object classes and subclasses known to occur in graphical user interfaces. TestFactory uses it to reconcile and classify the controls it encounters as it maps the AUT and creates scripts. The types of objects found in the Application Map and UI Library folders are described in Chapter 4, *Developing and Working with the Application Map*.

From the left pane of the TestFactory window, you can:

▶   Start mapping the AUT.

▶   Click a UI object to display its UI object properties and bitmap image in the right pane.

▶   Start the AUT and drive to a selected control.

▶   Create folders and organize your work.

▶   Select scripts to add to a Test Suite.

▶   Create a new UI object subclass.

▶   Reclassify a generic object.

▶   Change the subclass of a UI object.

▶   Access Robot scripts for the open project.

▶   Click a script to display its coverage data and steps in the right pane.

▶   Open or run a script in Robot.

▶   Open the Test Log window of TestManager to see the log for a script run.

▶   Insert TestFactory objects in the application map.

The Right Pane

The right pane of the TestFactory window displays information about the current object of focus in the left pane. The following figure shows the right pane after the Classics Login window object is clicked in the application map. The **Properties view** displays the UI object properties for the selected object. The **Image view** displays a bitmap of the selected UI object.

Properties view

Image toolbar

Image view
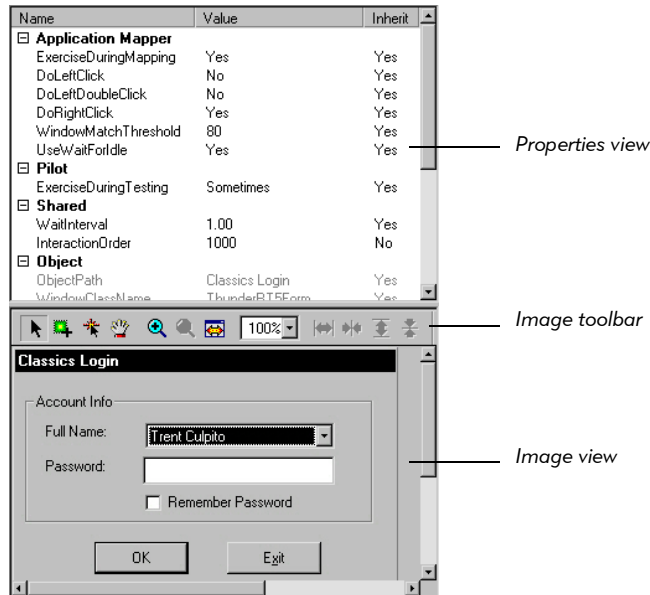
From the right pane, you can:

► View and edit the UI object properties of UI objects.

► See where a mapped control is located in the AUT user interface.

► Double-click a region of the bitmap to jump to the corresponding object in the application map.

► Configure and run a Pilot.

► Add UI object components to an interaction object and configure the object.

► View the outlined steps and coverage results for a script.

► Create and run a Test Suite and view the results.

► View report results.

► Create and modify a region object.

► View marker information.

2-17

Progress Bars

TestFactory has five **progress bars**, each of which opens at the bottom of the screen after TestFactory begins a specific task. The progress bars display status information about the process underway. Each contains a **Stop** button that you can use to stop the active process.

The **Application Mapper progress bar** opens after you start mapping the AUT.
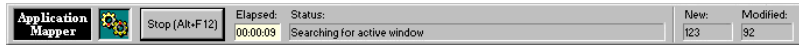
The **Pilot progress bar** opens after you start a Pilot run.

The **Script progress bar** opens after you start a script or Test Suite run.
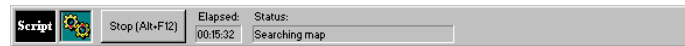
The **AutoPilot progress bar** opens after you start the AutoPilot to run multiple scripts, Test Suites, and Pilots.
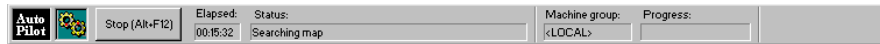
The **Go To Control progress bar** opens after you right-click a UI object and select **Go To "Control" in AUT** on the shortcut menu.
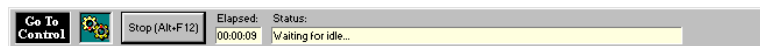
The Status Bar

The **status bar** at the bottom of the TestFactory window displays information about the activity in progress, the current object of focus, and the name of the user who is logged on. To toggle the status bar display, click **View > Status Bar**.

► ► ► C H A P T E R 3

Instrumenting the Application-Under-Test

This chapter addresses the first task you perform in TestFactory—instrumenting the AUT so that you can obtain code coverage information for your scripts when you start testing. This chapter describes what is required to instrument the AUT, how to instrument the AUT, and how to use the stand-alone instrumentor to instrument the source files of a Visual Basic application before build time. This chapter includes the following topics:

► About instrumentation

► Adding the Rational Test Enablers to the AUT before instrumenting

► Requirements for instrumenting the AUT

► Instrumenting C++, Visual Basic, and Java object code

► Instrumenting Visual Basic source code

About Instrumentation

If you want to obtain code coverage values for the scripts that you generate in testing, then you first need to instrument the AUT. TestFactory uses information in the instrumented files to calculate code coverage values for the best scripts that your Pilots generate, as well as for the Robot scripts that you run from TestFactory.

TestFactory calculates a **code coverage** value and a **UI coverage** value for a best script. The code coverage value represents the percentage of all AUT source code that the script exercises. The UI coverage value represents the percentage of unique UI objects that the script touches. Without instrumentation, TestFactory can use only UI coverage as a metric to generate a best script. Because instrumentation significantly improves testing results, we highly recommend that you instrument the AUT before you map it.

To instrument an application in TestFactory, you use either **object code instrumentation** or **source code instrumentation**. The object code method instruments the executable file using debug information. The source code method instruments the source code files of the AUT.

You can instrument the object code of an AUT written in C++, Java, and Visual Basic 5 and 6. If the AUT is written in Visual Basic and you have access to the source code files, you can instrument the source code files. If the AUT is written in Visual Basic 4, you can *only* instrument the source code files. Regardless of the instrumentation method you use, TestFactory maintains the integrity of your original source files or executable file.

Adding the Rational Test Enablers to the AUT Before Instrumenting

If you installed the Rational Test Enablers, you can add the ActiveX Test Control to the files of your AUT. The ActiveX Test Control enhances the control-recognition capabilities of TestFactory and Robot.

If you have not already done so, we highly recommend that you do the following:

▶ Install the Rational Test Enabler appropriate for your IDE.

▶ Open the AUT in its IDE and add the ActiveX Test Control to your application files before you instrument.

For information about installing the Rational Test Enablers, see the *Rational Suite Installation Guide*. For information about adding the ActiveX Test Control to your application, see the *Using Rational Robot* manual.

Requirements for Instrumenting the AUT

This section describes which instrumentation method (object code or source code) to use with Java, C++, and Visual Basic applications and what is required for instrumentation.

The method you use to instrument the AUT depends on the following:

▶ The IDE used to develop the AUT

▶ For Visual Basic applications only:

– The version of Visual Basic used to develop the AUT

– Access to source code

The following table lists the requirements for instrumenting the AUT using the source code and object code methods:

	Requirements for instrumenting source code	Requirements for instrumenting object code
VB 4.0	A .vbp file and all source files needed to build the .exe (can be read-only)	N/A
VB 5.0 and VB 6.0	A .vbp file and all source files needed to build the .exe (can be read-only)	An .exe file and a .pdb file (with debug information) located in the same directory
C++	N/A	An .exe file and a .pdb file (with debug information) located in the same directory
Java	N/A	None

NOTE: Regardless of how you instrument the AUT, you need access to source code if you want to view source code coverage details in the Coverage Browser for a C++ or Visual Basic application after you create scripts. For information about the Coverage Browser, see Chapter 5, *Automatically Generating Scripts*.

Java Applications and Applets

You can only instrument a Java application or applet using the object code method. There are no other specific requirements for instrumenting a Java AUT.

C++ Applications

You can only instrument a C++ application using the object code method. To do this, you must have the .exe file for the AUT and an associated .pdb file (containing debug information) located in the same directory.

Visual Basic Applications

If the AUT is written in Visual Basic 4, you can only instrument its source code files. If the AUT is written in Visual Basic 5 or 6, you can instrument either the source code files or the executable file.

If you have access to source code files for the AUT, we recommend that you instrument these files instead of the executable file. TestFactory instruments Visual Basic source code at the branch level, and instruments an executable file at the statement level. Because TestFactory takes into account the *if*, *then*, and *else* conditions in instrumented source code, it displays more detailed code coverage values for instrumented source files. You can also instrument the source files of secondary applications that the main application calls and executes. This lets you get coverage information for scripts that exercise the secondary applications.

Instrumenting C++, Visual Basic, and Java Object Code

If you want to instrument the object code of an AUT written in C++ or Visual Basic 5 or 6, you must first make an .exe file and a .pdb file that is located in the same directory as the .exe file. There are no specific requirements for instrumenting the object code of a Java application or applet.

Setting the Compiler Option for a Visual Basic 6 Application

If the AUT is written in Visual Basic 6, and you did *not* install Rational PureCoverage as a Visual Basic add-in, you need to set the compiler option before you make the .exe and .pdb files.

To set the compiler option for an AUT written in Visual Basic 6:

1. Open the .vbp file in Notepad.

2. Add the following lines at the end of the .vpb file:

    ```
    [VBCompiler]
    LinkSwitches=-Fixed:no
    ```

3. Save the .vbp file and quit Notepad.

Setting the Compiler Option for a C++ Application

Before you make the .exe and .pdb files for an AUT written in C++, you need to set the compiler option.

To set the compiler option in Visual C++:

1. Open the project file in Visual C++.

2. Click **Project > Options**, and then click the **Link** tab.

3. In the **Project options** box, scroll to the end and add **/fixed:no** to the options.

If you are working with the C++ application in an IDE other that Visual C++, make sure that you set the equivalent option to indicate that there are relocations in the executable file before you compile.

Instrumenting Object Code

To instrument the object code for the AUT:

1. Click **Tools > Instrument**, or click **Instrument** on the Standard toolbar.

Add button
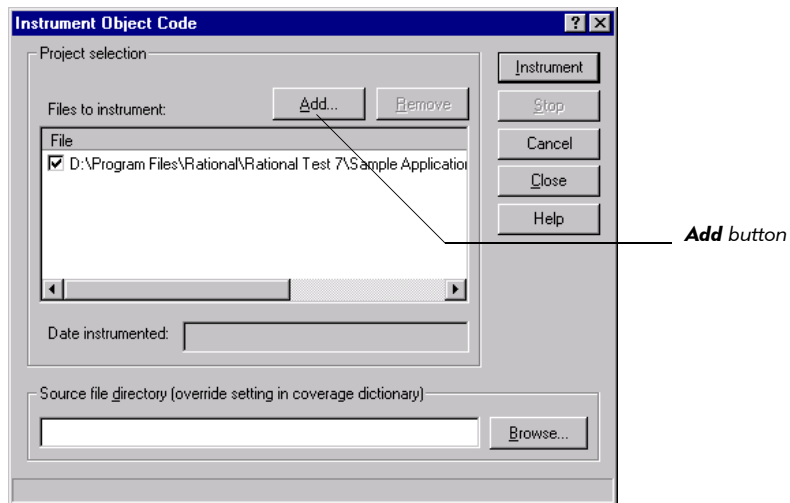
If the AUT has dependent .dll files, and you want to get coverage information on the code written in these files, you must add them to the **Files to instrument** list. If you specified the executable file for the AUT on step 2 of the New Project Wizard, then the **Files to instrument** list already contains the path of the executable file, as well as any dependent .dll files that TestFactory found in the same directory. If you did *not* specify the executable file for the AUT in the New Project Wizard, then add it to the **Files to instrument** list now.

2. To add the .exe file and dependent .dll files, or the .class or .jar file to the **Files to instrument** list, click **Add**, and then browse to and select the files for the AUT.

   TestFactory lists the selected file and the dependent .dll files that it can detect. Make sure that all of the files that you want to instrument are listed.

3. If the AUT has a dependent .dll file that is not in the same directory as the executable file, and you want to instrument the .dll file, click **Add**, and then browse to and select the file.

4. Leave the check box next to the .exe or .class file path selected. If the AUT is a C++ or Visual Basic application, select the check boxes for all of the .dll files that you want to instrument.

5. To leave a file uninstrumented, clear its check box.

6. To remove a file from the list, click the file path, and then click **Remove**.

7. Click **Instrument**.

8. After instrumentation is completed, click **Close**.

Instrumenting Visual Basic Source Code

To instrument Visual Basic source code files:

1. Click **Tools > Instrument**, or click **Instrument** on the Standard toolbar.

Add button
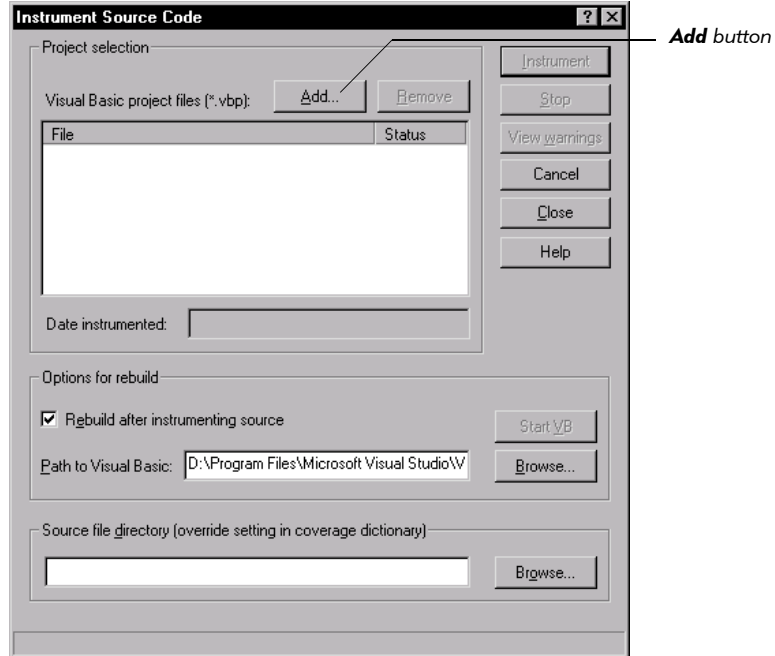
If you did not specify the source code method in step 3 of the New Project Wizard, then the Instrument Object Code dialog box opens instead of the Instrument Source Code dialog box. If this happens, close the Instrument Object Code dialog box, and then do the following:

a. Click **Tools > Options**, and then click the **Project** tab.
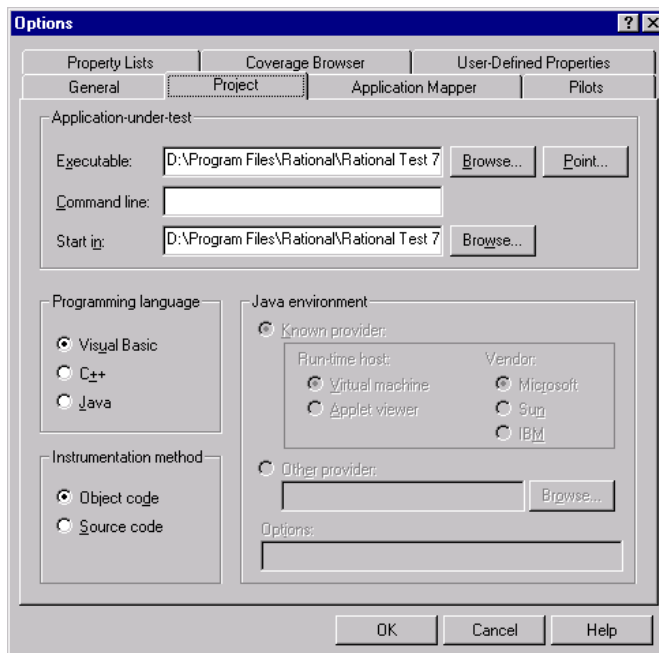
b. Under **Instrumentation method**, click **Source code**.

c. Click **OK**.

d. To open the Instrument Source Code dialog box, click **Tools > Instrument**, or click **Instrument** on the Standard toolbar.

2. To list the main project file for the AUT in the **Visual Basic project files** box, click **Add**, and then browse to and select the .vbp file. Leave the check box next to the file path selected.

An AUT 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.

3. To instrument a secondary application in addition to the main application, click **Add** and select its .vbp file. Leave the check box next to the file path selected.

If you add secondary files to instrument, be sure to add the .vbp file for the main application first, so it is at the top of the **Visual Basic project files** list.

> **NOTE:** If you specify a secondary application to instrument, TestFactory automatically adds its executable file to the list of executable files to map. For information about mapping secondary applications, see *Mapping Secondary Applications* on page 4-69.

4. To leave a listed .vbp file uninstrumented, clear its check box.

5. After you instrument the source files, you must rebuild the executable file. To rebuild the executable file automatically from instrumented source files, leave the **Rebuild after instrumenting source** check box selected.

> **NOTE:** We strongly recommend that you use the automatic rebuild option. If you clear the **Rebuild after instrumenting source** check box, you must rebuild the .exe file in Visual Basic while the Instrument Source Code dialog box is open. Otherwise, TestFactory does not save the instrumentation.

6. If the path shown in the **Path to Visual Basic** box is incorrect, enter the correct path to your Visual Basic executable file.

7. To remove a .vbp file for a secondary application from the **Visual Basic project files** list, click the file path, and then click **Remove**.

8. Make sure that all of the files that you want to instrument, and no others, are listed and checked, and then click **Instrument**.

If Visual Basic cannot rebuild the executable file, do the following:

a. Clear the **Rebuild after instrumenting source** check box.

b. Click **Start VB**.

c. Determine and correct the cause of the problem.

d. Rebuild the executable file manually.

e. Save the project, and then quit Visual Basic.

> **NOTE:** If you still have problems making the executable file, make sure that you can build the project using just Visual Basic.

9. After instrumentation is completed and the executable file is rebuilt, click **Close**.

The Coverage Dictionary

As TestFactory instruments code, it creates a **coverage dictionary** that it uses to calculate code coverage for scripts. If every member of your testing team shares the same build of the AUT, and saves test data to the same project, then each has access to the coverage dictionary after the AUT is instrumented.

If team members share the same build of the AUT, but have independent projects and save test data to separate repositories, then they can move the coverage dictionary between repositories to share it. A developer can instrument AUT source files and export the coverage dictionary for use by testers who only have access to AUT executable files. A tester can then import the coverage dictionary and use it to obtain code coverage data on scripts (as long as the scripts run against the executable file made from the same instrumented AUT). To move a coverage dictionary between projects, you must export it from one project and import it to another project.

NOTE: The coverage dictionary is visible only after you instrument source files or preinstrument the Visual Basic AUT using the stand-alone instrumentor.

Exporting and Importing the Coverage Dictionary

To export the coverage dictionary after you instrument source files:

1. Click **Tools > Import/Export Dictionary**.
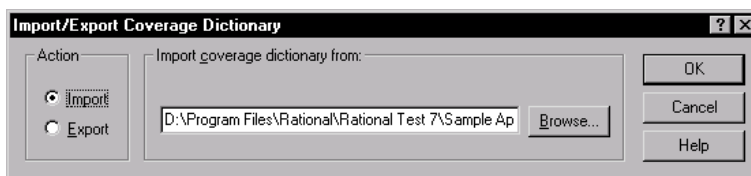
2. Under **Action**, click **Export**.

3. In the **Export coverage dictionary to** box, enter the path for the exported dictionary file, including the .vcd file name extension.

4. Click **OK**.

Importing the Coverage Dictionary

To import the coverage dictionary:

1. Click **Tools > Import/Export Dictionary**.

2. In the **Import coverage dictionary from** box, enter the path of the file to import, including the .vcd file name extension.

3. Click **OK**.

After you import the coverage dictionary, TestFactory can calculate code coverage metrics for the scripts that you generate and run.

Specifying the Source File Directory for Browsing Coverage Data

If you import a coverage dictionary so that you can view coverage details in the Coverage Browser, you must specify the directory for the instrumented source files for the project.

To specify the instrumented source files path:

1. Click **Tools > Instrument**.

2. In the **Source file directory** box, enter the source file directory.

3. Click **OK**.

Using the Stand-Alone Instrumentor to Instrument Visual Basic Source Code Files

If you have access to the Visual Basic source files for the AUT, you can instrument them at build time using the TestFactory stand-alone instrumentor. Every developer and tester who installs the new build can use TestFactory to get code coverage data without having to instrument the source files. The TestFactory stand-alone instrumentor is intended for use in a makefile.

To instrument the AUT outside of the main TestFactory program, type the following command in the makefile:

```
sqa7sci ...\<AUT name>.vbp
```

where `...\<AUT name>` is the full path of the project.

To instrument a secondary Visual Basic application that the AUT calls, add the full path of its .vbp file to the command-line argument, as shown in the following example:

```
sqa7sci ...\<Main AUT name>.vbp <Secondary app name>.vbp
```

TestFactory creates just one coverage dictionary, regardless of whether you instrument just the main project file for the AUT or instrument multiple files.

After you instrument the AUT, you can proceed to the next task—mapping the AUT.

Developing and Working with the Application Map

This chapter describes how to develop an application map for the AUT. It includes guidelines on how to prepare for mapping, procedures for mapping and evaluating the application map, and methods you can use to fine-tune the application map. This chapter includes the following topics:

▶   About TrueMap Technology

▶   Mapping the AUT for the first time

▶   Viewing and evaluating the Mapping Summary report and the application map

▶   Application map objects and their properties

▶   Excluding specific functions in the AUT from mapping and testing

▶   Improving the application map

▶   Using interaction objects to guide the Application Mapper through the AUT

▶   Using UI object properties to specify input and interaction order for mapping

▶   Creating and mapping a region object for an unmapped control

▶   Mapping similar windows

▶   Timing events during mapping

▶   Reclassifying a generic object

▶   Handling error message and crash transition objects in the application map

▶   Mapping new builds

▶   Mapping secondary applications

▶   Inserting TestFactory objects in the application map

▶   Creating and working with TestFactory reports

About TrueMap Technology

The application map is the foundation for automatic script generation in TestFactory. To create the application map, TestFactory uses its **TrueMap** technology to thoroughly explore the user interface of the AUT and exercise every control it finds. The **Application Mapper** tool uses the UI library of object classes to identify and reconcile the controls it finds to build a detailed, hierarchical application map that models the graphical user interface of the AUT. The application map gives you access to all user interface objects, scripts, reports, object properties, and coverage data for a TestFactory project.

The Pilots that you insert use the application map to generate scripts that test the AUT. It is important that you run Pilots in areas of the application map that are fully developed and accurately represent the user interface of the AUT. With a well-developed application map, your Pilots can generate scripts that thoroughly test your product.

Mapping the AUT for the First Time

Before you map the AUT for the first time, consider the functional areas it contains, whether those areas have stabilized, and the complexity of the interactions between the functional areas.

The time it takes to create a full application map can vary from minutes to hours, depending on the size and complexity of the AUT. If your AUT is large and complicated, consider mapping just the first level of controls on the first pass, and then focus your mapping efforts on individual regions. For information about depth of mapping, see *Using the Application Mapper Wizard to Map the AUT for the First Time* on page 4-5.

Setting Up the Mapping Environment

During the mapping process in TestFactory, the more closely the working environment resembles that of an actual use situation, the more complete and accurate the resulting application map. A real person using an application changes the system environment to provide appropriate input and to otherwise get the application to function as it is designed to function. The user might change certain system settings, import a bitmap file to edit, or populate a form with data from a spreadsheet program. Without this input, the application cannot perform all of the functions it is capable of performing.

A user also determines the best time to exercise a control in an application. For example, a user clicks a specific command button only when it is enabled, and not while the application is responding to a previous event. If an action is not properly timed, the application cannot respond correctly.

The **Application Mapper** tab in TestFactory contains controls that you can use to set up a realistic use environment before you map the AUT. This section describes how to specify **support scripts** for TestFactory to run before and after mapping, and how to set the maximum amount of time the Application Mapper waits to perform the next action after it exercises a control.

> **NOTE**: In addition to the main application, you can map secondary applications that the AUT loads and executes. For information about mapping secondary executable files, see *Mapping Secondary Applications* on page 4-69.

Specifying Support Scripts

To set up the system environment before mapping and then restore the system environment after mapping is completed, you can use support scripts. For example, if you want to configure a database and give the AUT access to it during mapping, you can create a Robot script that configures the database, and then specify the script as a **startup script** for mapping. To restore the system to its previous state after mapping, you can create and specify another Robot script as a **cleanup script**.

To specify support scripts for mapping:

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

Application Mapper tab

Startup box
Cleanup box
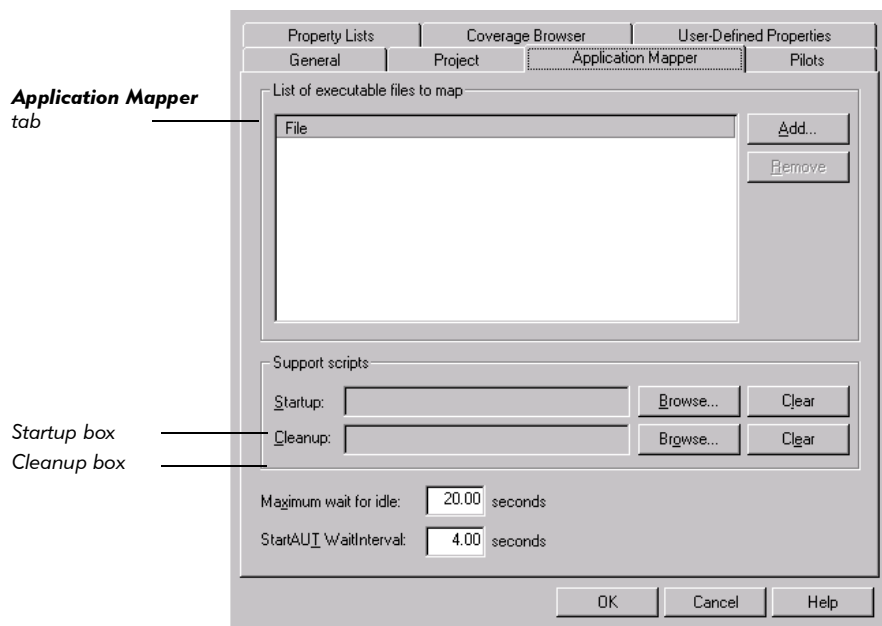
2. To specify a startup script, use the **Browse** button next to the **Startup** box.
3. To specify a cleanup script, use the **Browse** button next to the **Cleanup** box.
4. To remove a script from the **Startup** box or **Cleanup** box, click the adjacent **Clear** button.
5. After you specify support scripts, click **OK**.

Using the Application Mapper Wizard to Map the AUT for the First Time

If you specified the path to the executable file in step 2 of the New Project Wizard, then you can use the **Application Mapper Wizard** to start the Application Mapper for the first time. (For information about the New Project Wizard, see *Using the New Project Wizard* on page 2-3.) Otherwise, you must first enter the executable file path in the **Executable** box on the **Project** tab.

To enter the path to the executable file on the **Project** tab:

1. Click **Tools > Options**, and then click the **Project** tab.
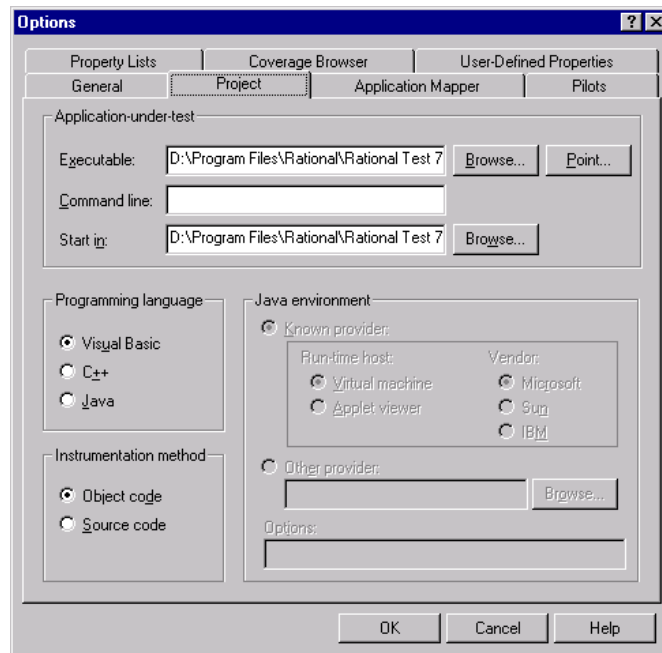
2. Under **Application-under-test**, enter the path to the .exe, .class, or .jar file for the AUT in the **Executable** box.

3. To pass one or more command-line arguments to the AUT when it is started from TestFactory, type the argument in the **Command line** box. Be sure to specify the command-line argument in the syntax that the AUT requires.

4. In the **Start in** box, enter the full path to the working directory. If you do not enter a start path, TestFactory uses the AUT executable path.

5. Under **Programming language**, click the programming language for the AUT.

6. If the AUT is written in Visual Basic, and you want to instrument the executable file (object code) instead of source code, under **Instrumentation method**, click **Object code**.

7. If the AUT is written in Java, then under **Java environment**, click the option for the Java virtual machine or applet viewer on which you plan to run the AUT.

8. If the AUT is written in Java, then under **Java environment**, type command-line arguments that you want to pass to the Java virtual machine or applet viewer in the **Options** box.

9. To save your settings and close the Options dialog box, click **OK**.

NOTE: We strongly recommend that you instrument the AUT before you begin to map it.

To map the AUT for the first time using the Application Mapper Wizard:

1. Click **Tools > Application Mapper**, or click **Application Mapper** on the Standard toolbar.

2. Click **Next**.
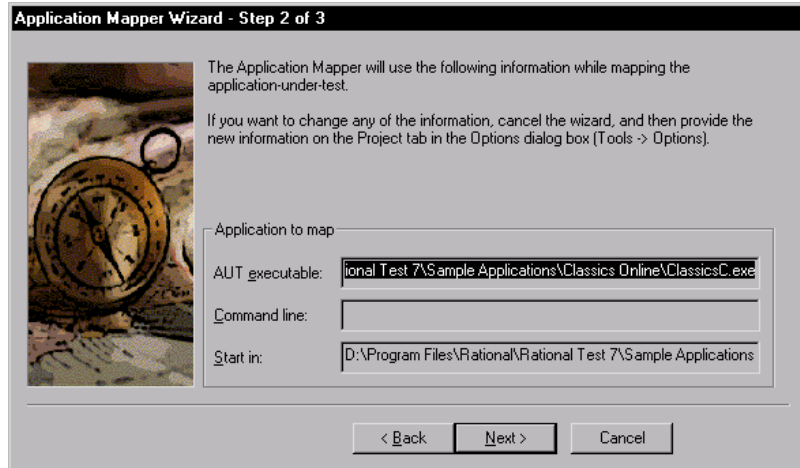
3. If any of the project information displayed under **Application to map** is incorrect, click **Cancel**, and then do the following:

   a. Click **Tools > Options**, and then click the **Project** tab.

   b. Under **Application-under-test**, modify the incorrect value(s).

   c. Click **OK**.

   d. Repeat steps 1 and 2.

4. Click **Next**.
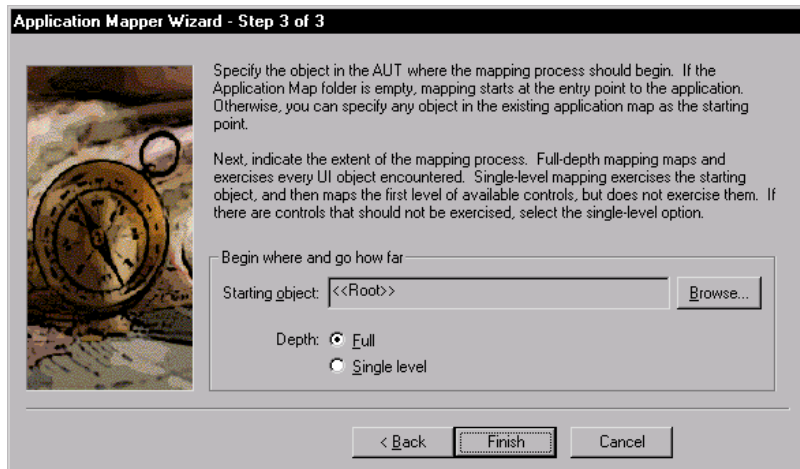
The Application Mapper uses the **starting object** as its base for exploring the AUT user interface. On the first mapping of the AUT, the default <<Root>> is the appropriate starting point. After you map the AUT the first time, you can choose any UI object in the map as the starting object.

You can map the AUT to **full depth** or **single-level depth**. To map the AUT to full depth, the Application Mapper begins at the starting object and explores all levels of the AUT, exercising every control it encounters until it has taken every path available. To map the AUT to single-level depth, the Application Mapper drives to the starting object, and then exercises the control it represents. TestFactory then maps the controls that are exposed, but does not exercise them. Before you choose the depth of mapping, consider the size, complexity, and stability of the AUT.

> **NOTE:** If the AUT is large or complex, or contains controls that you do not want TestFactory to exercise, map to single-level depth on the first pass. You can then select a starting object in the area of the application map where you want to start testing and map to full depth from that object. For instructions, see *Mapping New Builds* on page 4-66.

5. To select the depth of mapping, click **Full** or **Single level**.

6. To start mapping, click **Finish**.

> **NOTE:** Be sure you do not use the computer while mapping is in progress.

On-Screen Events During Mapping

After you start mapping, the following events occur:

▶ The TestFactory window closes.

▶ The Application Mapper progress bar appears at the bottom of the screen and displays information about the mapping process.
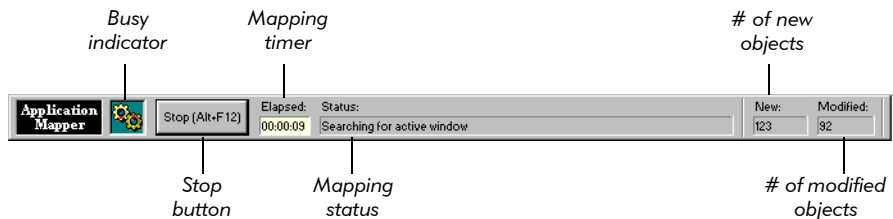
▶ A mask darkens the screen and displays the message *Running Application Mapper*.

▶ TestFactory starts Robot, and then minimizes the Robot window.

- ▶ TestFactory starts the AUT and maps it.

- ▶ After the mapping session is completed, the TestFactory window is restored.

Stopping the Mapping Process

If mapping takes longer than you expected, or you need access to something on your computer, you can stop mapping at any time.

To stop mapping:

- ▶ Press ALT+F12 or click **Stop (Alt+F12)** on the Application Mapper progress bar.

After you stop the mapping process, allow TestFactory time to stop Robot activity, quit the AUT, and restore its own window. This can take a few minutes.

> **NOTE**: If you stop mapping, then the next time you map, use the same starting object that you used before you stopped mapping. Mapping will pick up where it left off.

Viewing and Evaluating the Mapping Summary Report and the Application Map

After TestFactory completes a mapping session, it displays the resulting application map in the left pane. TestFactory displays the contents of the **Mapping Summary** report in the right pane.

Viewing the Mapping Summary Report

Every time you map the AUT, TestFactory creates a **Mapping Summary -** *<date time>* report and places it in the Mapping Reports folder. The Mapping Summary contains information that you can use to evaluate and improve the application map.

Mapping Summary report object in the left pane
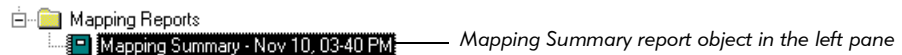
Name	Object Path
Statistics:	
Depth	Single Level
Starting object path	<<Root>>
Completion status	Completed
Elapsed time	00:00:19
Crash objects	0
New objects	12
New window objects	1
New child objects	0
Modified objects	0
Previously mapped objects ...	0
Warnings	0
New window objects:	**Path**
Classics Login	Classics Login

Mapping Summary report contents displayed in the right pane

Double-click here to jump to the Classics Login window object in the application map.

The Mapping Summary report lists the starting object for mapping, the depth of mapping, the number of new and modified objects found, executable files that were excluded from mapping, deleted objects, new windows mapped, and more.

To jump to a UI object listed in the Mapping Summary report, double-click the object name. TestFactory places the focus on the selected UI object in the application map.

To return to the Mapping Summary report contents in the right pane, click **Previous Object** on the Standard toolbar.

Expanding and Navigating the Application Map

TestFactory provides several ways for you to view the entire application map or selected areas of the application map.

To view sections of the completed application map:

▶ To expand an individual object in the application map just one level:

    – Click the plus character (**+**) next to the object.

    – Double-click the object.

▶ To jump to and expand objects in the map, use the up, down, and right arrow keys on your keyboard.

▶ To fully expand an object in the application map:

    – Click the object, and then click **Expand** on the Standard toolbar.

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

> **NOTE**: If the AUT is large, complete expansion of the entire application map can take several minutes.

As long as you click objects in the application map, TestFactory keeps track of where you click. This means that if you are comparing different regions of the application map, and lose track of the objects you previously selected, you can retrace your steps.

To jump to objects that you have recently selected in the application map:

▶ To jump to the last object you selected, click **Previous Object** on the Standard toolbar.

▶ To jump to the next object in a series of objects you selected, click **Next Object** on the Standard toolbar.

After mapping is completed, you must check the results for completeness and accuracy. Look for unmapped controls, replicate windows, and other discrepancies between the AUT user interface and the application map. To help you compare the application map with the AUT, you can use the **StartAUT** feature, or you can use the **Go To "Control"** feature.

To use the StartAUT feature to start the AUT from TestFactory:

▶ Click **StartAUT** on the Standard toolbar.

To use the Go To "Control" feature to start the AUT and go to a specific control in the user interface:

▶ In the application map, right-click the UI object mapped for the control you want to access, and then click **Go To *"Control"* in AUT** on the shortcut menu.
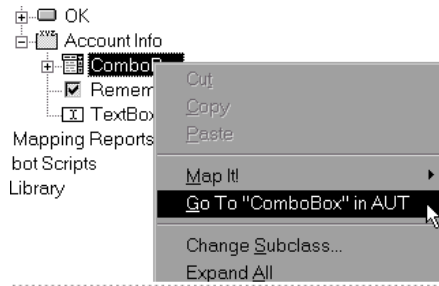
After you click **Go To *"Control"* in AUT**:

▶ The TestFactory window closes.

▶ The Go To Control progress bar opens at the bottom of the screen.

Busy indicator   *Time elapsed*

Stop (Alt+F12) button   *Status*
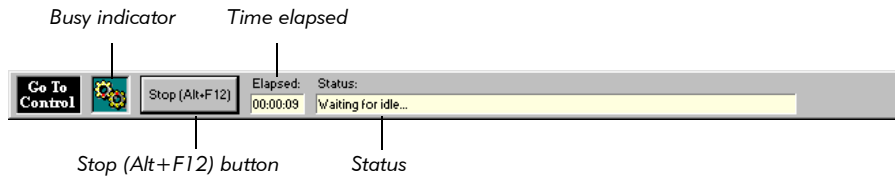
▶ A mask darkens the screen and displays the message *Go To Control*.

▶ TestFactory starts the AUT and drives to the selected control.

▶ After TestFactory drives to the control, the Go To Control progress bar closes and the TestFactory window is restored. The AUT stays open on the screen.

> **NOTE:** Be sure to wait until TestFactory restores its window before you exercise controls in the AUT.

Application Map Objects and Their Properties

The application map includes objects that represent all of the controls that TestFactory found in the AUT user interface, the actions that exposed additional user interface, and the transitional states encountered, such as a window closing. The following figure shows part of the application map for the "Classics" sample application.
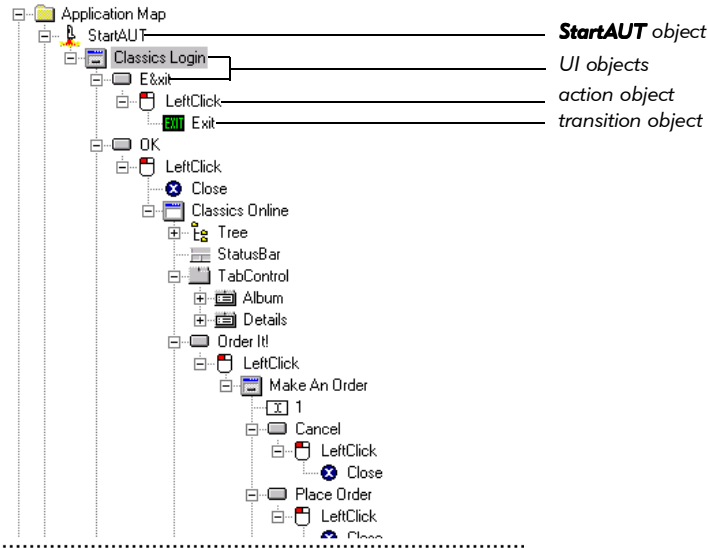
The application map contains **UI objects**, **action objects**, **shortcuts**, and **transition objects**. Except for shortcuts, each object in the application map is an **instance** of an object class found in the UI library. The basic object types are described below.

TestFactory displays the following five types of objects in the application map hierarchy under StartAUT:

▶   **Known UI objects** represent the various controls that the Application Mapper identified in the user interface. These include check boxes, list boxes, command buttons, combo boxes, grid controls, and so on. Each is an instance of a UI object class in the UI library.

> **NOTE:** TestFactory maps some control types, such as grid controls and calendars, that TestFactory does not currently test. TestFactory maps a grid control as a grid UI object, but Pilots cannot exercise the cells in the grid unless you first create region objects for them. For information about creating region objects, see *Creating and Mapping a Region Object for an Unmapped Control* on page 4-47.

▶ **Generic UI objects** represent controls that the Application Mapper encounters but cannot recognize. Although the Application Mapper queries the control for recognition properties such as class name, style, and text, it cannot match these properties with those of a known class in the UI library. Generic objects are often custom controls.

▶ **Action objects** represent the actions to which controls in the AUT respond during mapping. Action objects include LeftClick, RightClick, LeftDoubleClick, SelectTab, SetState, and ClearState.

An application map contains a single StartAUT action object that represents the action to start the AUT executable file. The **StartAUT** object is always the top-level action object in the application map hierarchy. The Application Mapper and Pilots use it to start the AUT.

▶ **Transition objects** indicate where a window closed, where the AUT terminated, or where the AUT crashed.

– Crash transition objects show where the AUT failed during mapping.

– Close transition objects show where an action caused a window to close.

– Exit transition objects show where an action caused the AUT to terminate normally.

▶ **Shortcut objects** show where the Application Mapper encountered a window that was already mapped on a different path through the user interface. The shortcut points to the window object originally mapped. To display the properties of the window a shortcut points to, click the shortcut. To jump to the mapped window object that a shortcut points to, double-click the shortcut.

Renaming UI Objects in the Application Map

As TestFactory maps input controls such as text boxes and combo boxes in the AUT, it assigns general names such as "TextBox1" and "TextBox2" to the UI objects it maps for them. To make an input UI object in the application map easier to identify, you can rename it.

To rename an object in the application map:

1. Click the object.

2. Press F2, and then type a new name in the active text box.

3. Press ENTER.

Properties of Objects in the Application Map

Objects in the application map have two kinds of properties—**UI object properties** and **user properties**. This section describes UI object properties and user properties and how to view them.

UI Object Properties

Every object in the application map has UI object properties. These are the behavioral and physical attributes of the UI object (and the UI object class to which it belongs).

To display the UI object properties for an object:

▶   Click the object in the application map.

The UI Object properties are displayed in the **Properties view** in the top right pane.

An object has four groups of UI object properties: **Application Mapper**, **Pilot**, **Shared**, and **Object** properties.

To expand a properties group and view the properties it contains:

▶   In the Properties view, click the properties group name.

Name	Value	Inherit
⊞ **Application Mapper**		
⊞ **Pilot**		
⊞ **Shared**		
⊞ **Object**		

Properties groups displayed in the Properties view

Click here to expand the Application Mapper properties group.
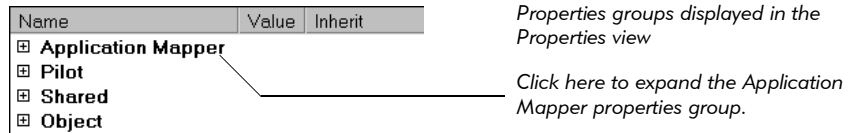
Application Mapper properties determine whether and how the Application Mapper interacts with an object. The following figure shows the Application Mapper properties of a window UI object.

Name	Value	Inherit
⊟ **Application Mapper**		
ExerciseDuringMapping	Yes	Yes
DoLeftClick	No	Yes
DoLeftDoubleClick	No	Yes
DoRightClick	Yes	Yes
WindowMatchThreshold	80	Yes
UseWaitForIdle	Yes	Yes

Application Mapper properties of a window object

Pilot properties determine whether and how TestFactory exercises an object during testing. Pilot properties also establish the format and content of input passed to the AUT during testing. The following figure shows some of the Pilot properties of a text object.

For controls that accept text as input, the Properties view includes the Style toolbar.

Pilot properties of a text object

Shared properties affect both mapping and testing. You can use them to specify a required string case to use as input for a control, to specify the order in which TestFactory exercises controls, and to make TestFactory wait a specified amount of time for the AUT to respond after exercising a control. The following figure shows the Shared properties of a text object.

Shared properties of a text object

Object properties are intrinsic to the UI object. They include such properties as **ObjectPath** (in the application map), **RecognitionMethod**, and **Style**. The following figure shows the **Object** properties of a window object.

Name	Value	Inherit
⊟ **Object**		
ObjectPath	Classics Online	Yes
WindowClassName	ThunderForm	Yes
WindowStyle	382664704	No
WindowID	3704	No
WindowRectangle	53,123,970,645	No
WindowText	Classics Online	No
Enabled	Yes	No
RecognitionMethod	"Name=frmMain"	No
Style	0x00000000	Yes
StyleMask	0x00000000	Yes
UILibrarySubclass	VBFormWindow	No
AccessKey		No
VolatileWindow	No	Yes

Object properties of a window object

Application Mapper, Shared, and Pilot properties are modifiable. Except for the RecognitionMethod and VolatileWindow properties, Object properties are purely descriptive and cannot be modified.

To learn the function of a UI object property, right-click its name in the Properties view, and then click **What's this?** on the shortcut menu.

Printing UI Object Properties

To print a bitmap image of a UI object and all of its UI object properties:

1. Click the UI object in the application map.

2. Click **File > Print** or click **Print** on the Standard toolbar.

In addition to the UI object image and UI object properties, TestFactory prints the project name, the date and time that you printed the image, and the path to the object in the application map.

User Properties

In addition to UI object properties, every object in the application map has **user properties**. User properties provide information about an object in the context of the project. They consist of the read-only properties **Created** and **Last modified**, modifiable properties such as **Name** and **Description**, and user-defined properties that you create for the object. You can specify user properties as search filters for objects that you want to include in a report, for scripts to include in a Test Suite, or for Pilots to add to a user scenario or mix-in.

Viewing User Properties

To open the User Properties dialog box and view the user properties for an object, do one of the following:

▶ Click the object, and then click **Edit > User Properties**.

▶ Right-click the object, and then click **User Properties** on the shortcut menu.
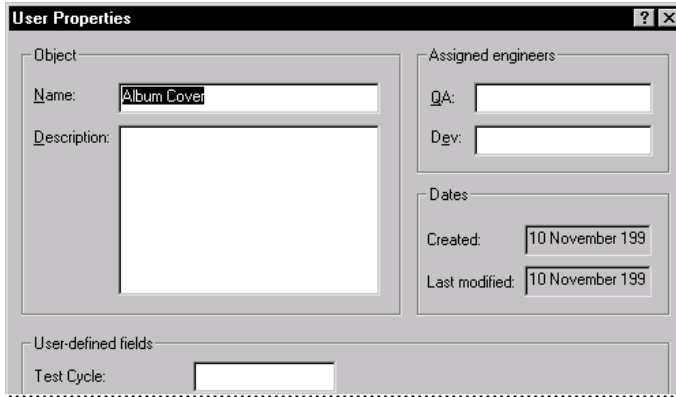
Defining a New User Property

To define a new user property for all objects in the application map:

1. Click **Tools > Options**, and then click the **User-Defined Properties** tab.
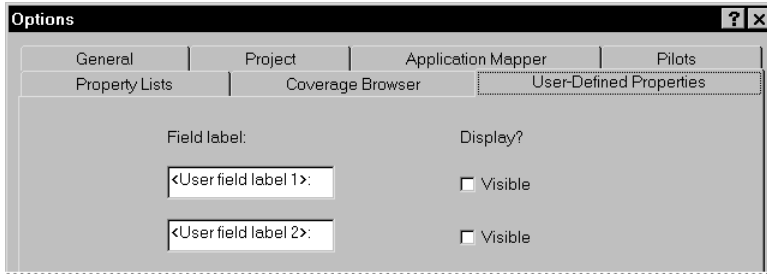
2. Type a name for a new user property in a **Field label** box.

3. If you want to display the property in the User Properties dialog box, select the **Visible** check box. If you prefer not to display the property in the User Properties dialog box, leave the **Visible** check box cleared.

4. Click **OK**.

5. To view a new user property (that you have made visible) for an object, click the object in the application map, and then click **Edit > User Properties**.

User-defined field
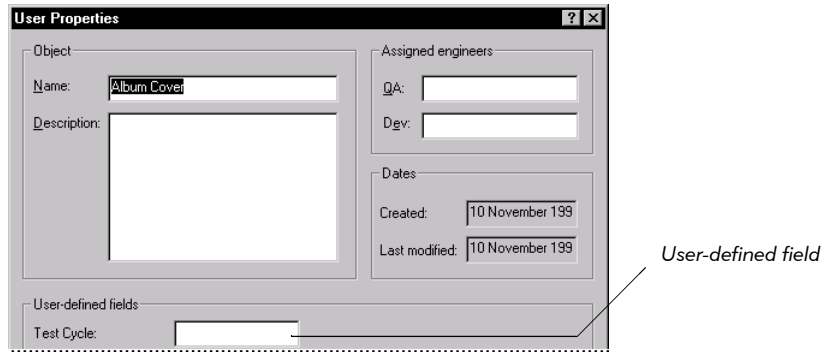
Finding Objects in the Application Map

If you need to find scripts, Pilots, or a particular type of UI object in the application map, or if you want to locate all UI objects that have a specific user property value, use the Find Objects window.

To locate objects based on UI object name, type, or user property values:

1. Click **Edit > Find Objects**, or click **Find Objects** on the Standard toolbar.

Find Objects window
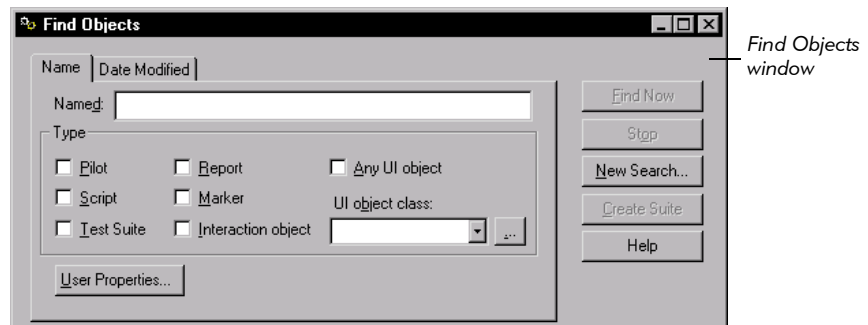
2. To specify an object or objects to find, do one of the following:

    – In the **Named** box, type part or all of an object name using the following syntax:

Type	To find
abc	Objects named abc
abc★	Every object with a name that starts with abc
★abc★	Every object with the string abc in its name
abc?	Every object with a 4-character name that starts with abc

    If you specify a name to search for an object, and the object name includes a mnemonic, be sure to type the ampersand character (&).

    – Select check boxes for one or more object types to find.

    Alternatively:

    a. Click **…** next to the **UI object class** list.
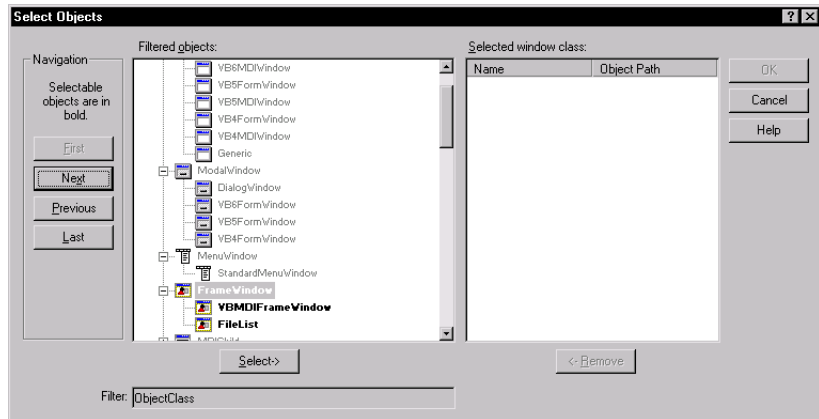
    a. Scroll through the **Filtered objects** list or click **First**, **Next**, **Previous**, and **Last** to jump to selectable objects in the list.

    b. To select a window class or subclass in the **Filtered objects** list, click the window class item in the **Filtered objects** list, and then click **Add**.

    c. To close the Select Objects dialog box, click **OK**.

3. To refine the search based on the user properties of UI objects:

    a. In the Find Objects window, click **User Properties**.

    b. Type information in one or more of the available boxes in the Filter User Properties dialog box, and then click **OK**.

4. To filter objects based on the date created or last modified, click the **Date Modified** tab, select the **Find all objects created or modified** check box, and then do one of the following:

    –   To specify a time interval:
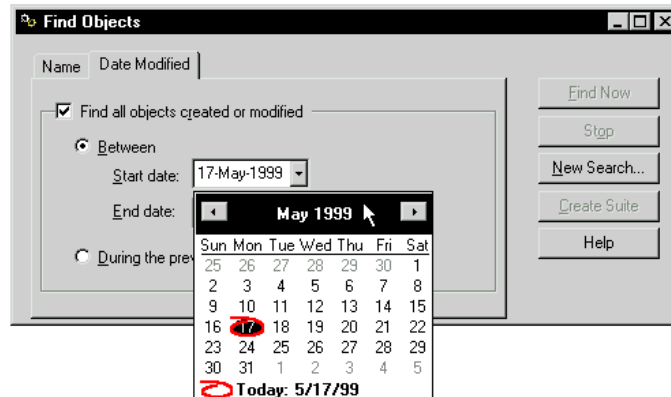
    a. Leave **Between** selected.

    b. To set a time interval start date, click the down arrow in the **Start date** box, and then click a calendar date.

    c. To set a time interval end date, click the down arrow in the **End date** box, and then click a calendar date.

    –   Alternatively, to specify a number of previous days up to and including the current day, click **During the previous**, and then type a number in the **day(s)** box.

5. Click **Find Now**.

After the Find Objects window displays a list of objects found, you can jump directly to a listed object in the application map.

To jump to one of the found objects listed:

1. Double-click the object name in the Find Objects window.

2. Close the Find Objects window.

The object is selected in the application map.

Excluding Specific Functions in the AUT from Mapping and Testing

The AUT can include functions that you do not necessarily want it to perform during mapping and testing. During mapping, a control such as a "Save as" menu command could create unwanted files. A "Clear database" or "Delete record" button could delete data that you want to keep for testing purposes. To prevent the AUT from performing functions such as these during mapping and testing, you can either modify UI object properties to exclude the controls involved from full mapping and testing, or use an alternative strategy outside of TestFactory. The following example shows how you can manage a print function during mapping and testing.

If you fully map or test an area of the AUT that includes the print function, you can generate unwanted printed material. To work around this problem, do one of the following:

▶ Map the first level of controls on the print dialog box, and then change the **ExerciseDuringMapping** and **ExerciseDuringTesting** properties for the UI object mapped for the print control.

▶ Set the Windows printer driver to hold the print job(s), and then delete the spooled jobs.

To change the ExerciseDuringMapping and ExerciseDuringTesting properties for the print control object:

1. Map the first level of controls on the print dialog box.

2. In the application map, click the UI object mapped for the print control.

3. In the Properties view in the top right pane, click **Application Mapper**.

4. Double-click the **Value** field for **ExerciseDuringMapping**, and then click **No**.

5. In the Properties view, click **Pilot**.

6. Double-click the **Value** field for **ExerciseDuringTesting**, and then click **Never**.

For information about how to set the Windows printer driver to hold the print job(s), and then delete the spooled jobs, see Help for your Windows operating system.

Improving the Application Map

The time and effort that you invest in analyzing and refining the application map is important. With a complete and accurate map, the Pilots that you run later can generate scripts that correctly test the AUT.

Some applications have features that TestFactory can fail to map initially. These features fall into the following categories:

▶ Multiple alternative paths in the same functional area of the AUT.

▶ A control that requires specific data as input.

▶ A control that must be exercised using a specific action.

▶ A control that requires a character string as input.

▶ Controls that must be exercised in a precise order.

▶ A control that the Application Mapper did not map.

▶ A control that the Application Mapper could not classify.

The following sections describe how to resolve these issues after a mapping session.

Using Interaction Objects to Guide the Application Mapper through the AUT

Although the Application Mapper is very good at automatically exploring and mapping an application, it cannot exercise the AUT the same way that an experienced user can. Even simple applications contain controls that are active or inactive based on certain conditions. Examples are a toolbar button such as **Copy** that is only available after the user selects a list box item, or a dialog box that opens only after the user performs a complex series of steps. To become active, a control can require a specific mouse action, string case, or text format. In addition, functional areas of an application can require that users exercise controls in a precise order.

An experienced user can perform all of the steps needed to take a particular path in an area of an application that has special requirements. The Application Mapper, however, can require your input to bring the AUT to the same state.

Interaction Objects and Interaction Object Components

You can provide the Application Mapper with the input it needs to navigate areas of the AUT by inserting and setting up **interaction objects** in the application map. Interaction objects guide TestFactory to specific areas of the AUT that it might not expose automatically. They do this by supplying specific input for, and defining an ordered sequence of interactions through, specific controls in the AUT.

You can use interaction objects to map and test multiple alternative paths in a functional area of the AUT. Without interaction objects, you can only map a single path in a given area of the AUT.

An interaction object is a container to which you add one or more UI object **components** that represent controls in the AUT. An interaction object must contain components for all of the controls that a user must exercise to take the path that you want TestFactory to take in the AUT.

When you add a UI object component to an interaction object, you essentially add a copy of the UI object mapped for a control. Once you add a component to an interaction object, that component is independent of the UI object on which it is based. It has its own properties that you can modify to control the way in which TestFactory exercises the corresponding control during mapping and testing.

Although TestFactory creates just one UI object in the application map for a control in the AUT, you can add UI object components for the same control to multiple interaction objects. Because you can modify the properties for each independent component, you can set up two interaction objects that contain components for the same controls to guide TestFactory through two different paths in the AUT.

This section describes how to set up interaction objects and provides examples of how you can use them to develop a complete and accurate application map.

Setting Up an Interaction Object

To set up an interaction object, you must:

▶   Insert an interaction object in the application map.

▶   Add the necessary components for the interaction.

▶   Set the properties for each component.

▶   Set the interaction order for the components.

This section provides instructions on how to insert and set up an interaction object to control the direction and activity of the Application Mapper as it navigates the AUT user interface.

Inserting an Interaction Object

To map an unmapped area of the AUT, you start by inserting an interaction object in the application map at the UI object mapped for the last control that must be exercised to expose the unmapped area. For example, on a purchase order form, a user must enter data in several boxes, and then click a command button to submit the order and receive a confirmation message. To map the path that includes the confirmation message box, you would insert an interaction object at the UI object mapped for the command button.

You must insert an interaction object at a UI object that corresponds to a control that responds to a user action. For example, you can insert an interaction object at a menu command, but not on a menu name.

To insert an interaction object, do one of the following:

1. In the application map, expand the top-level window that contains all of the controls involved in the navigational path that you want TestFactory to take.

2. Click the UI object mapped for the last control involved in the navigational path to expose more of the user interface.

3. Click **Insert > Interaction Object**.

    Alternatively,

1. In the application map, expand the top-level window that contains all of the controls involved in the navigational path that you want TestFactory to take.

2. On the Insert toolbar, click **Interaction Object**.

3. In the application map, click the UI object mapped for the last control involved in the navigational path to expose more of the user interface.

TestFactory inserts an interaction object under the selected (parent) UI object.

To name the interaction object, type a name in the active text box, and then press ENTER.

Type a name for the interaction object here.
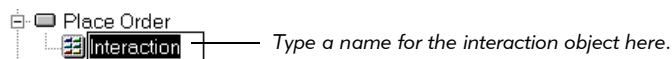
After you insert an interaction object in the application map, the **Interaction Object view** in the top right pane lists the parent component that you selected in the application map.

The Interaction Object view lists the parent of the interaction object as the first component.
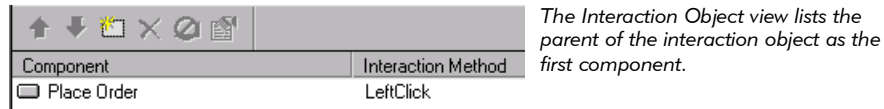
You cannot move the parent component that you select for the interaction object. The parent component is always the last component listed and the last component that TestFactory exercises as it maps and tests the interaction object.

Adding UI Object Components to an Interaction Object

The components that you add to an interaction object must meet the following two requirements:

▶ A UI object that you add as a component to an interaction object must be located in the same window as the parent UI object for the interaction object.

▶ UI object components within the same interaction object must not lead to different paths in the AUT. If you add two UI object components that represent controls that lead to different areas of the AUT, an error occurs after you start mapping the interaction object, and mapping stops.

The following figure illustrates part of the application map for the Classics sample application. The interaction object inserted at the Place Order button object can include components for any of the controls mapped under the Make an Order window object on this branch of the map.

Make An Order window object

The Place Order button object is the parent of the interaction object.

Possible interaction object components
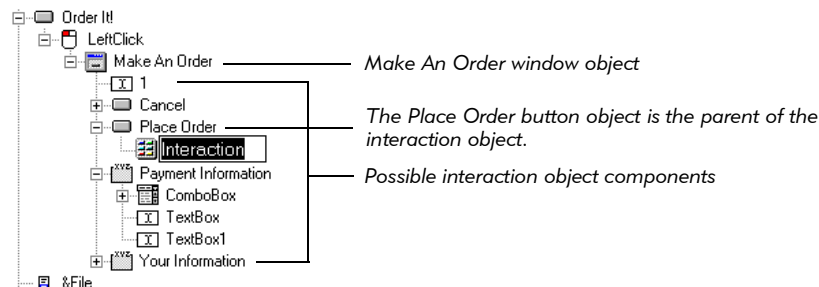
To add a UI object component to an interaction object, do one of the following:

1. To add a component from the Image view in the bottom right portion of the window, use the scroll bars in the Image view to display the image of the component that you want to insert.

2. In the Interaction Object view, click the component above which you want to add a new component.

3. Press INSERT, or click **Insert Item** on the Interaction Object toolbar.

Insert Item button   *Interaction Object toolbar*

Component	Interaction Method
	<Click the object in the Image view>
▢ Place Order	LeftClick
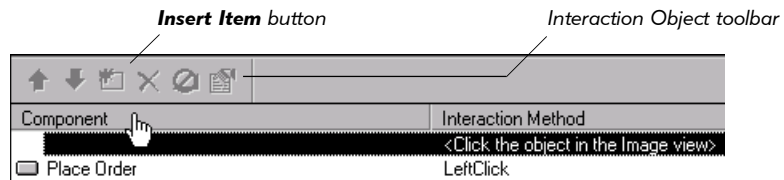
4. In the Image view, point to the image of the component that you want to insert, and then, after TestFactory outlines the image in red, click the image.

Payment Information

Card Number (include the spaces):

Card Type: Visa   Expiration Date:

TestFactory outlines the component you point to with a red box.
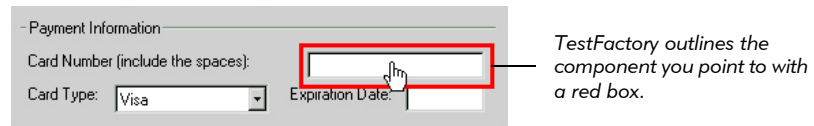
TestFactory lists the UI object component in the Interaction Object view.

Component	Interaction Method
Ⅰ Card Number	*Base*
▢ Place Order	LeftClick

Alternatively,

▶ To add a component from the application map, drag a UI object from the application map to the Interaction Object view.

   TestFactory places a UI object component above the parent component in the Interaction Object view.

   NOTE: Be sure that you drag, and do not click, the UI object. If you click the UI object in the application map, the Properties view for the UI object replaces the Interaction Object view in the top right pane.

To point to multiple controls in the Image view and insert them as components in the interaction object:

1. Press SHIFT and click **Insert Item** on the Interaction Object toolbar.

2. In the Image view, point to the image of a control for which you want to insert a component, and then click the image after TestFactory outlines the image in red.

3. Point to and click the image of the next control for which you want to insert a component. Continue to point and click until you have added all of the components necessary for the interaction.

4. To end the multiple selection process, click inside the Interaction Objects view, or press ESC as you move the pointer.

Adding Support Script Components to an Interaction Object

If the operations needed to bring the AUT to a given state cannot be accomplished using UI object components in an interaction object, you can add a support script component to the interaction object to help drive the AUT to the target state. For example, if a grid control (which TestFactory cannot currently exercise) leads to unexposed user interface, you can record the required actions in a Robot script, and then add that script to the interaction object.

To add a support script to an interaction object:

1. In the application map, click the interaction object.

2. In the left pane, expand the Robot Scripts folder.

3. Drag the support script object from the left pane to the Interaction Object view.

During mapping and testing, TestFactory exercises the script in the order that you list it in the Interaction Object view.

Copying and Pasting Interaction Objects

If you want to insert an interaction object that includes the same components as an existing interaction object, you can copy the existing interaction object, and then paste the copy in the map.

To insert a copy of an existing interaction object in the application map:

1. Right-click the existing interaction object, and then click **Copy** on the shortcut menu.

2. Right-click the UI object mapped for the last control involved in the navigational path that leads to the state you want to map, and then click **Paste** on the shortcut menu.

> **NOTE:** You can only paste a copy of an interaction object within the same top level window that contains the original interaction object. If you insert the copy at a different UI object than the original interaction object, then TestFactory replaces the last component in the interaction object with a component for the new parent UI object.

Add all of the components required for the interaction to the Interaction Object view. After you add all of the components, you can select interaction methods for those that require a specific action to become available, and select styles and specify required string cases for those that require specific text as input.

Copying and Pasting Interaction Object Components

If you want more than one instance of the same component in an interaction object, you can place a copy of the original component at another position in the interaction object.

To copy a component and paste it at another position in the interaction object:

1. Press CTRL, and then click the component that you want to copy.

2. Drag the component to another position in the interaction object.

TestFactory places a copy of the original component in the new position.

Setting the Properties for a UI Object Component

Like the UI object on which it is based, a UI object component has properties that determine how TestFactory exercises the control in the AUT during mapping and testing. These properties are parallel to the UI object properties displayed in the Properties view for a UI object.

Setting an Interaction Method for a Component

The interaction method that you specify for an interaction object component determines the action or style that TestFactory uses to exercise the corresponding control in the AUT.

To specify an action method for a component:

▶   In the Interaction Object view, double-click the **Interaction Method** field for the component, and then click the appropriate interaction method.
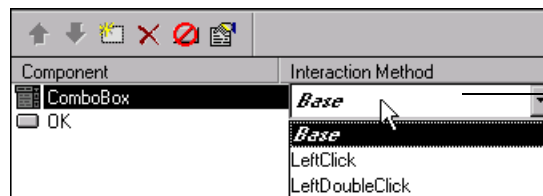
Interaction Method field for the ComboBox component

If you select an action method for an input-type of component to which you have assigned a style, the selected interaction method overrides the style.

Assigning a Style to an Input-Type Component

TestFactory provides many styles that you can assign to UI objects and components that represent input-type controls such as text boxes and combo boxes. A style consists of a combination of data entry settings that TestFactory uses to test controls during Pilot runs. Although none of the existing styles has a required string case, you can specify a required string case for any style. If you do, the Application Mapper uses the required string case for mapping.

We recommend that you assign a style to all of the input-type UI objects and components in the application map. If no existing style has the combination of data entry settings that you want TestFactory to apply to a control, you can override an existing style with modified settings, or create a new style. Assigning and modifying a data entry style is the most efficient way to determine how TestFactory exercises an input control.

After you add an input UI object component to an interaction object, TestFactory assigns the component the *Base* style. A UI object or component with the default Base style is tested using a random combination of integers, floating point values, and string values as entry data. You can assign the component a style that has a more appropriate mix of data entry settings for the control in the AUT.

To assign a style to an input component:

1. Click the input component in the interaction object view.

2. To open the Edit Data Entry Style dialog box, click **Style Properties** on the Interaction Object toolbar.

The Edit Data Entry style dialog box displays the data entry settings used to test a component (or UI object) with the default *Base* style.
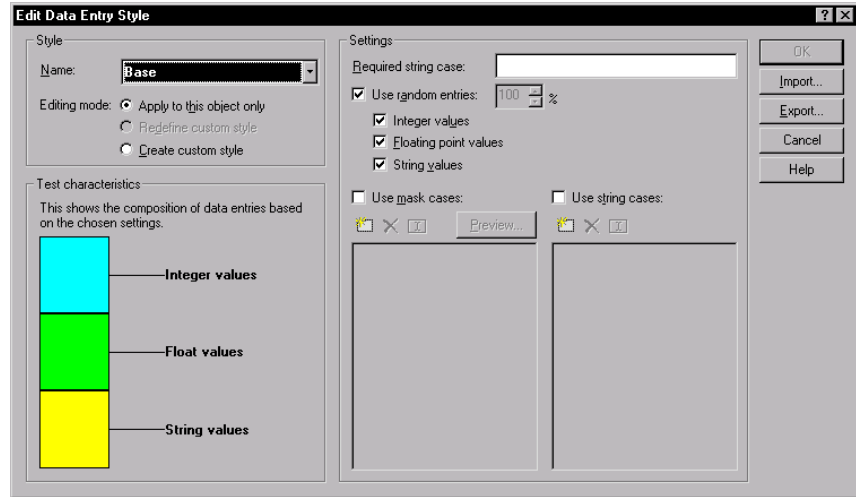
3. To view other available styles, under **Style**, click the **Name** box, and then click a style name.

List of existing data entry styles
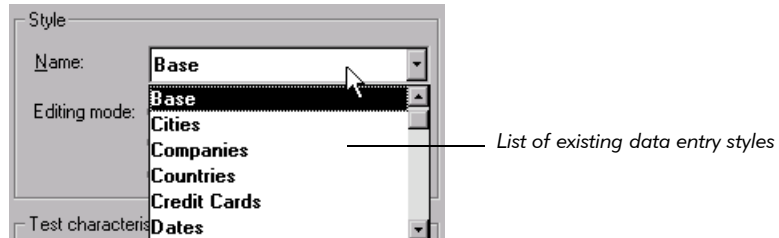
4. Under **Test characteristics** and **Settings**, examine the data entry composition for the selected style, and then do the same for other styles in the **Name** list.

5. In the **Name** list, select the style with the data entry composition that most closely matches the combination of data you want to use to test the control.

6. To specify a required string case for mapping, under **Settings**, type a string case in the **Required string case** box.

Type a required string case here.
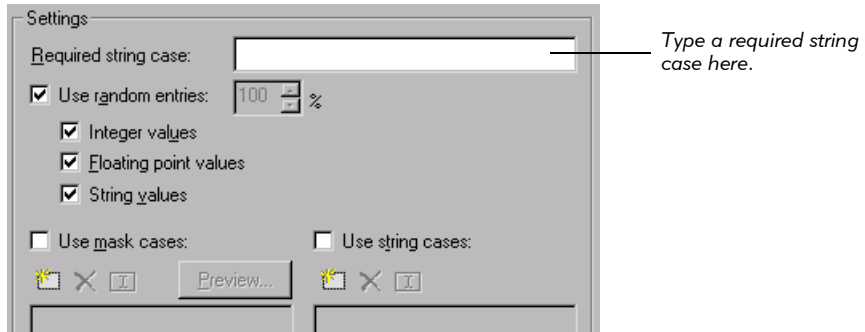
The Application Mapper uses only the **Required string case** setting to exercise the control. During testing, a Pilot uses the **Required string case** setting to reach the base state for testing, and then applies the other data entry settings.

7. Before you close the dialog box, you can edit the style for your Pilot runs. For information about changing a setting, click **Help**, or right-click the setting, and then click **What's This?** on the shortcut menu.

If you plan to use the modified settings only for the selected component, you can apply them as overrides of the selected style, or you can create a new style based on these settings.

8. To apply the modified settings as overrides of the selected style, click **OK**.

Alternatively,

To create a custom style based on the modified settings:

To create a new style, click here.

a. Under **Style**, click **Create custom style**.
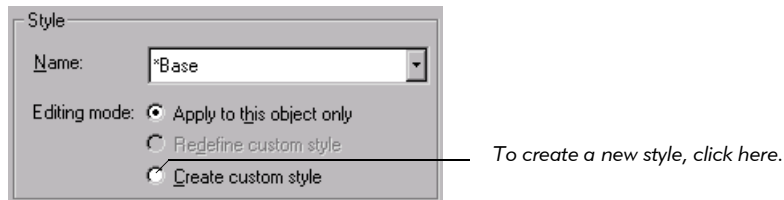
Type a style name here.
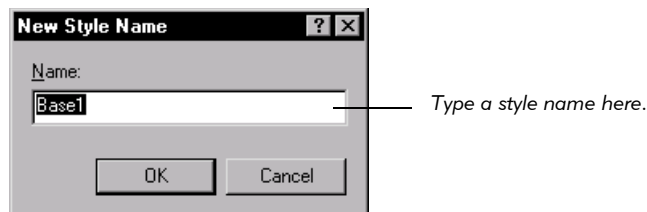
    b. Type a name for the style in the **Name** box.

    c. Click **OK**.

    d. To close the Edit Data Entry Style dialog box, click **OK**.

TestFactory assigns the new style to the selected component and adds the style name to the **Name** list and to the **Select style** list on the Style toolbar in the Properties view (for UI objects).

Specifying a Required String Case for a Component Using the Property List

An alternative way to specify a required string case for a component is to use the Property List dialog box. The properties that you can modify in the Property List for a component are the same as the UI object properties that you can modify for a UI object. The difference is that TestFactory applies the settings in the Property List only to the component as it maps and tests the interaction object that contains the component. Changing the property settings of a UI object component does not affect the UI object or its UI object properties.

> **NOTE:** We recommend that you use the Edit Data Entry Style dialog box to specify the required string case for a component. The Edit Data Entry Style dialog box lets you designate a style for the component, which TestFactory can then apply later when you run Pilots to test the mapped path.

To specify a required string case in the Property List dialog box:

1. In the Interaction Object view, double-click the component name.
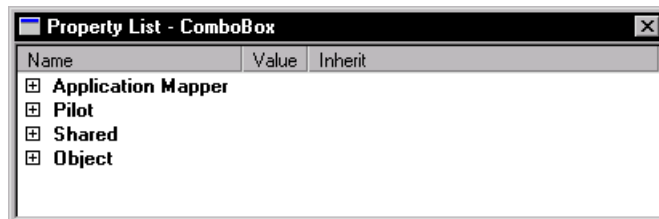
Property List dialog box for the ComboBox component

2. In the **Property List** dialog box, expand the **Shared** properties.
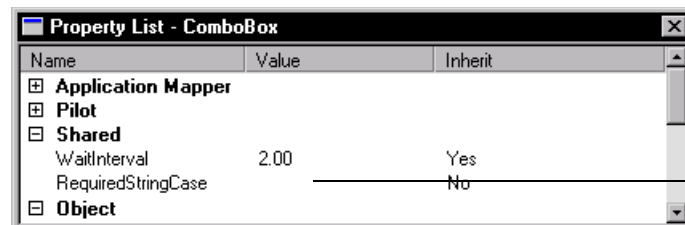
Click here.

3. Click the **Value** field for **RequiredStringCase**, and then type the required text in the active edit box.

4. Close the Property List dialog box.

> **NOTE:** Although many of the properties listed in the Property List dialog box for UI object components are the same as those listed in the Properties view for UI objects, the settings in this dialog box are applied only to the component. These settings do not change the UI object property settings of the UI object.

Setting the Interaction Order for Components

The Application Mapper exercises controls in the AUT based on the listed order of the corresponding components in the Interaction Object view, starting with the component at the top of the list.

To change the interaction order of a listed component:

▶ Click the component name, and then do one of the following:

– Use **Move Up** and **Move Down** on the Interaction Object toolbar.

– Press CTRL, and then press the up arrow and down arrow keys.

> **NOTE:** The parent component of the interaction object is always listed last. You cannot change its interaction order.

TestFactory observes the specified interaction order while mapping or testing the interaction object. For information about setting the interaction order for UI objects, see *Controlling the Interaction Order for UI Objects* on page 4-44.

Excluding a Component from Mapping

If an interaction object contains a component that you want to exclude from a mapping session, do the following:

▶ Click the component name in the Interaction Object view, and then click **Make Unavailable** on the Interaction Object toolbar.

To remove a component from the interaction object:

▶ Click the component name in the Interaction Object view, and then do one of the following:

– Click **Delete Item** on the Interaction Object toolbar.

– Press DELETE.

Mapping the Path Set by an Interaction Object

After you finish setting up the interaction object, you can map the path it sets out in the AUT.

To map the path for which you set up an interaction object, do one of the following:

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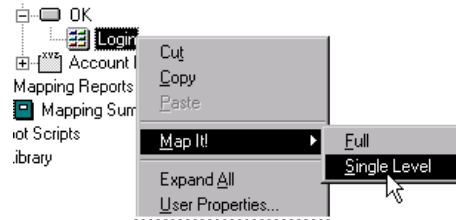
Alternatively,

▶ Use the Application Mapper Wizard, and specify the interaction object as the starting object for mapping.

As it maps an interaction object, the Application Mapper maps the last component to full depth, and maps all of the other components to single-level depth.

You can use any UI object that lies above an interaction object on the same branch of the application map as the starting object for mapping. If a branch of the application map contains two or more interaction objects at the same level in the map hierarchy, the Application Mapper maps them in the order they occur in the map, starting with the one inserted highest on the branch.

If an error occurs while the Application Mapper is exercising a component in an interaction object, TestFactory excludes the interaction object from further mapping and testing and displays the component name in the Interaction Object view in red text.

Excluding an Interaction Object from Mapping

Because interaction objects are TestFactory objects, and not UI objects, you cannot exclude one from mapping by modifying properties. If you want to exclude an interaction object from mapping, you must use its shortcut menu.

To exclude an interaction object from mapping:

▶ In the application map, right-click the interaction object, and then click **Exercise During Mapping** on the shortcut menu.
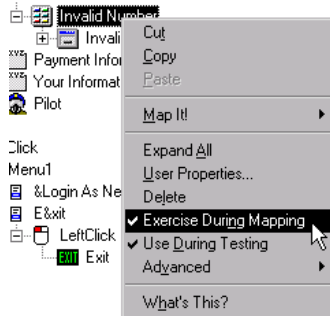
The **Exercise During Mapping** command toggles the mapping availability status of the interaction object.

If you want to include the interaction object in mapping sessions later, be sure to reset its active status.

Using Interaction Objects to Map Alternative Paths in the AUT

A functional area of an application can lead to multiple states, depending on how a user exercises the controls it contains. During application development, it is important that you map each available path in the AUT.

If you run a Pilot to test a region of the AUT that is not fully mapped, the Pilot can encounter controls it does not "expect" to see. If this happens, the Pilot generates UAW (unexpected active window) scripts and can fail to generate a code-coverage optimized best script. (For information about UAW scripts, see *Examining Pilot Run Results* on page 5-12 and *Viewing a UAW Script* on page 5-19.)

You can use interaction objects to map all of the available paths in an area of the AUT. Without interaction objects, you can only map one of the available paths in an area of the user interface.

Using an Interaction Object to Map Beyond a Logon Dialog Box

If the AUT contains a logon dialog box that requires input such as a password or user ID, then you must perform additional steps after the first mapping session so that the Application Mapper can access more of the user interface. The following figure shows the Classics Login dialog box and the resulting application map after mapping to single-level depth for the first time.

Classics Login dialog box

*To open the Classics window, a user must first enter a user name and a password, and then click **OK**.*

Single-level depth application map of Classics

UI object mapped for the Full Name combo box

UI object mapped for the Password text box
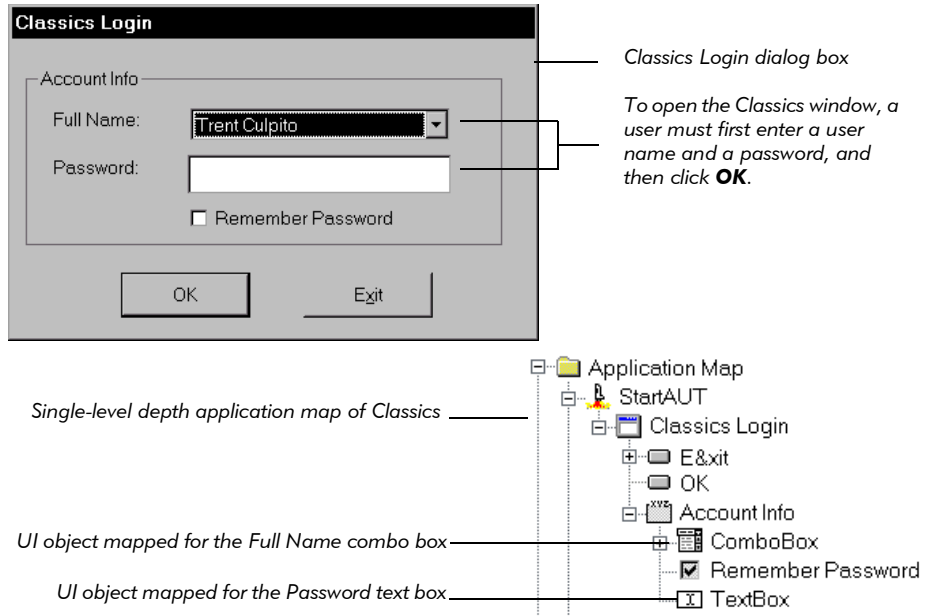
To get from the logon box to the Classics window, a user must enter a user name in the **Full Name** combo box, type a password in the **Password** text box, and then click **OK**. Because TestFactory did not have the required string cases to pass to the combo box and text box, the Classics window did not open after the Application Mapper exercised the OK button. To map beyond a login dialog box such as this, you must supply TestFactory with the necessary input to pass to the AUT, and specify the correct order for exercising the controls involved. The most efficient way to do this is to set up an interaction object.

To map beyond a login dialog box using an interaction object:

1. Map the AUT to single-level depth.

   After mapping is completed, the application map contains the StartAUT object, and the UI objects mapped for the logon dialog box and its child controls.

2. On the Insert toolbar, click **Interaction Object**.

3. In the application map, click the UI object mapped for the last control involved in the navigational path to expose more user interface. (In the Classics Login application map, the OK button object is the appropriate selection.)

OK button
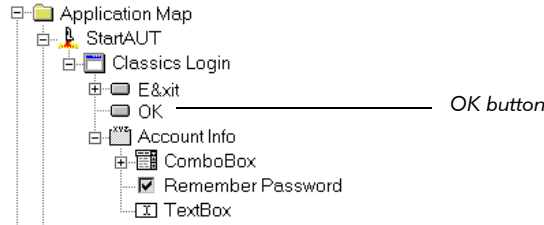
4. Type a name for the interaction object in the active text box.
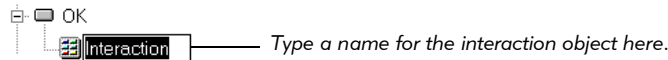
Type a name for the interaction object here.

5. Add all of the components necessary for the interaction to the interaction object.

6. Set the interaction method for each component that requires a specific action.

   Specify a required string case for the styles of each input-type component. (In the Classics example, you would specify required string cases for the *Base* style assigned to the ComboBox and TextBox components.)

7. To set the interaction order for a component, click the component name, and then use **Move Up** and **Move Down** on the Interaction Object toolbar to change its order in the list.

8. Remap the AUT from the <<Root>> object.

TestFactory passes the required string cases to the AUT. Supplied with the necessary logon information, the Application Mapper can access and map more of the user interface.

Although using an interaction object is the most convenient and flexible way to map beyond a logon dialog box, you can also do it by modifying the UI object properties for UI objects in the application map. For information about modifying UI object properties to direct mapping, see *Using UI Object Properties to Specify Input and Interaction Order for Mapping* on page 4-41.

Mapping Alternative Paths in an AUT

The following figure shows the Make An Order dialog box in the Classics sample application. To order a selected album, a user types a credit card number in the **Card Number** box, types the expiration date for the card in the **Expiration Date** box, and then clicks **Place Order**.

*To place an order for a selected item, a user must type information in these text boxes, and then click **Place Order**.*

After a user correctly submits an order, Classics displays the Order Confirmation message. If the user clicks **Place Order** without first completing the **Card Number** and **Expiration Date** boxes, Classics displays the Incomplete Order message. To test this area of Classics, you must first map both of these paths in the user interface.

To map alternative paths in an area of the AUT, start by mapping the default path that the Application Mapper takes if you do not specify input for controls in the AUT. Next, insert and set up an interaction object for each alternative path that a user can take, starting from a specific control (the last control involved in an interaction that exposes unmapped user interface).

To map the path to the Incomplete Order message box in Classics, the Place Order button object in the application map was mapped to full depth. The following figure shows the mapping results.

UI object mapped for the Incomplete Order message box
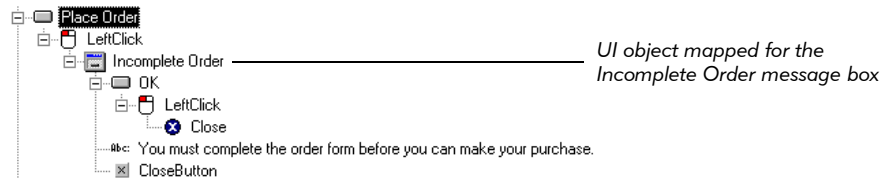
To map the path to the Order Confirmation message box in Classics, an interaction object was inserted in the application map at the Place Order button (the last control involved in the interaction). The following figure shows the Confirm Order interaction object and the corresponding Interaction Object view in the right pane.

Interaction Order view for the Confirm Order interaction object
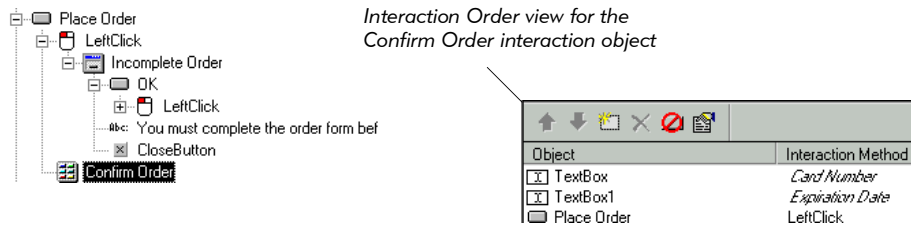
The Confirm Order interaction object contains the Place Order parent component, as well as the TextBox and TextBox 1 components (for the Card Number and Expiration Date text boxes, respectively).

TestFactory assigned the **Left Click** default interaction method to the Place Order component. Card Number and Expiration Date data entry styles were created for the TextBox and TextBox 1 components. A required string case was specified for both new styles.

The following figure shows the results of remapping the Place Order button to full depth.

The application map now contains the complete paths to the Incomplete Order message box and to the Order Confirmation message box.
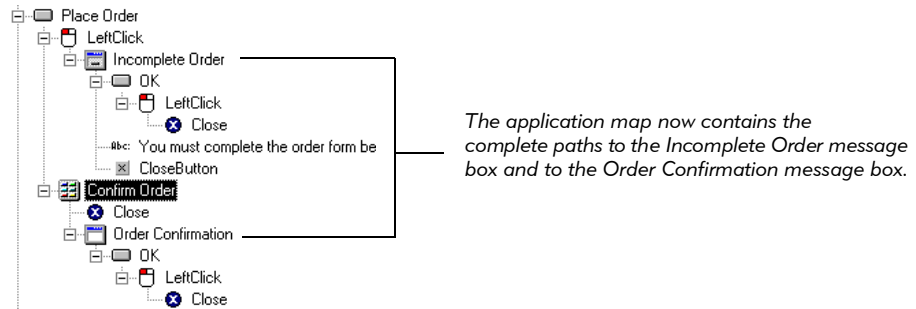
Using UI Object Properties to Specify Input and Interaction Order for Mapping

If you map an interaction object, TestFactory uses the interaction methods and required string cases that you specified for its components to exercise the corresponding controls in the AUT. If you map a UI object, TestFactory uses defaults to exercise the corresponding control, regardless of whether you added the UI object as a component to an interaction object and specified an interaction method or required string case for it.

If you want TestFactory to exercise a control in a specific way when the Application Mapper is not mapping an interaction object that contains the control as a component, you must first modify the UI object properties for the UI object that TestFactory mapped for it. This section describes how to specify mouse actions, required string cases, and interaction order to use for mapping UI objects.

We recommend that you specify input for controls in the AUT by representing the controls as components in interaction objects. You can use an unlimited number of data entry types to exercise a given control by inserting components for it in different interaction objects. This approach to mapping and testing lets you completely control input without having to modify the UI object mapped for the control.

Specifying Actions to Use for Mapping a UI Object

If the application map contains a UI object for a control that responds to mouse actions such as double-left-click or right-click, you can use the Application Mapper properties for the UI object to specify the mouse action(s) used during mapping.

To use the Application Mapper properties to set the correct action(s) for exercising a control:

1. In the application map, click the UI object mapped for the control.

2. In the Properties view in the top right pane, expand the **Application Mapper** properties group.

3. Double-click the **Value** field for the correct property (such as **DoLeftDoubleClick**), and then click **Yes**.

4. For actions that you do not want the Application Mapper to use, click the **Value** field for the corresponding property, and then click **No**.

The next time you map this UI object, TestFactory uses only the actions you specified to exercise it.

Specifying a Required String Case for Mapping

If a UI object is mapped for an input control that requires specific text as input, you can assign the UI object a data entry style, and then specify the required string case.

To assign a style to a UI object mapped for an input control:

1. In the application map, click the UI object mapped for the input control.

    The default data entry style setting for an input-type UI object is "<none>".

2. In the Properties view, click the **Select Style** box, and then click a style name.

Select Style *list*
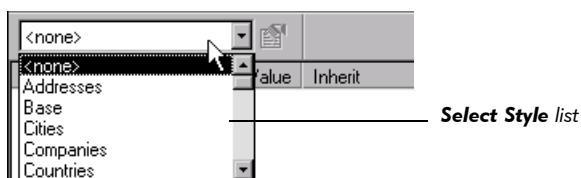
3. To open the Edit Data Entry Style dialog box, do one of the following:

–   Click **Style Properties**.

–   Right-click the UI object, and then click **Edit Style** on the shortcut menu.
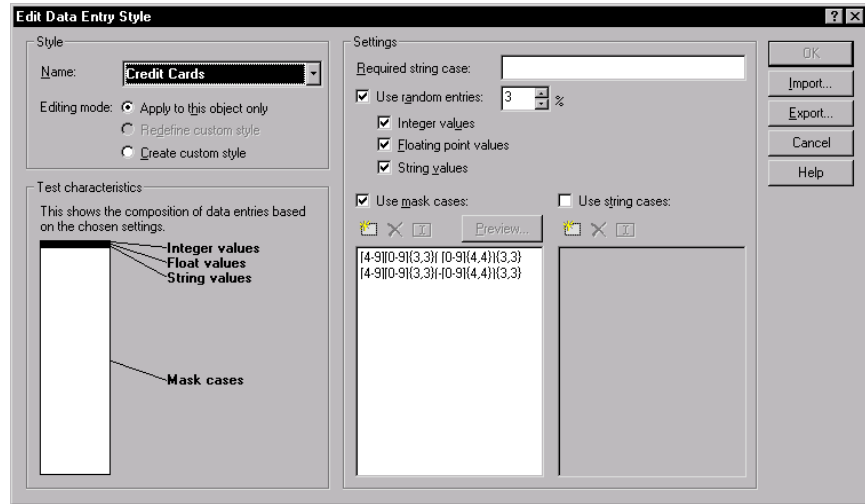
4. Under **Settings**, type the required string case in the **Required string case** box.

> **NOTE:** TestFactory applies the other data entry settings in the Edit Data Entry Style dialog box only during testing. The Application Mapper applies only the required string case you specify, and ignores the other settings.

The required string case you specify overrides the selected style. If you specify a data entry setting that overrides the current style, TestFactory places an asterisk character (⋆) next to the style name in the **Name** box.

5. Click **OK**.

The next time you map the control represented by this UI object, the Application Mapper uses only the specified required string case as input for the control.

You can specify a required string case for a UI object without assigning a style by modifying a Shared property.

To specify a required string case using the Shared properties:

1. In the Properties view, expand the **Shared** properties group.

Value field for the **RequiredStringCase** property

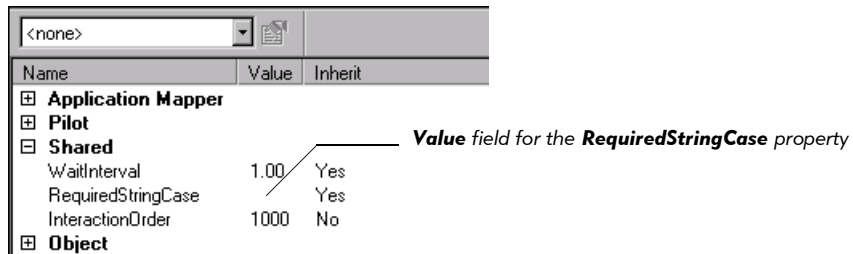**2.** Click the **Value** field for **RequiredStringCase**, and then type the required text in the active text box.

The next time you map this UI object, TestFactory uses only the required string case you specified as input.

> NOTE: If a required string case that you specify for a UI object will expose unmapped regions of the AUT, then be sure to remap the affected area before you run Pilots to test the area. Otherwise, a Pilot run in that region encounters unexpected active windows and generates UAW scripts. For information about UAW scripts, see *Examining Pilot Run Results* on page 5-12 and *Viewing a UAW Script* on page 5-19.

Controlling the Interaction Order for UI Objects

If a window in the AUT contains controls that a user must exercise in an exact order, you can set the interaction order used during mapping and testing by doing one of the following:

▶ Set up an interaction object that contains components for the controls, and arrange the components in the correct order for mapping and testing (recommended). For information about setting the interaction order for interaction object components, see *Setting the Interaction Order for Components* on page 4-34.

▶ Specify an **InteractionOrder** property value for the UI objects mapped for the controls involved in the sequence.

Setting the InteractionOrder Property for UI Objects

All UI objects in the application map have a shared InteractionOrder property with a default value of 1000. Because they have the same InteractionOrder value, TestFactory exercises controls in the same window in random order during mapping and testing.

If you change the InteractionOrder values of UI objects mapped for controls in the same window in the AUT, TestFactory exercises the controls based on the ascending order of their InteractionOrder values. For example, the Application Mapper exercises a UI object that has an InteractionOrder value of 900 before it exercises a UI object that has an InteractionOrder value of 1000.

If you set the value of the InteractionOrder property of a UI object to less than 1000, TestFactory maps the corresponding control to single-level depth only. During mapping, if a UI object that has an InteractionOrder value of less than 1000 leads to an unmapped area of the AUT, or closes the window that contains it, mapping stops.

If you set the InteractionOrder of a UI object to a value *greater* than 1000, TestFactory maps the corresponding control to the depth you specify (single-level or full depth). If the last control in an interaction sequence exposes an unmapped area of the AUT, and you want to map that area, be sure to set the InteractionOrder of the corresponding UI object to a value equal to or greater than 1000.

To set the interaction order for controls using the InteractionOrder property:

1. In the application map, click the UI object mapped for the control that you want the Application Mapper to exercise first.

2. In the Properties view, expand the **Shared** properties group.

Value field for the InteractionOrder property
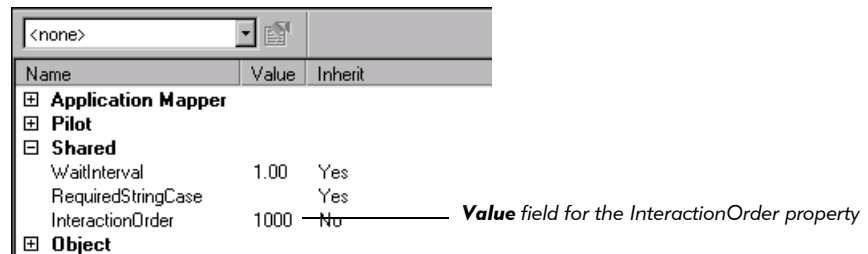
3. Replace the default value (1000) for **InteractionOrder** with a value that is greater than 0 and less than 1000.

4. In the application map, click the next UI object in the interaction sequence.

5. In the Properties view, type a value for **InteractionOrder** that is greater than the value you set for the previous UI object in the interaction sequence and less than 1000.

6. Repeat steps 3 through 5 for the remaining UI objects in the interaction sequence.

After you start mapping, the Application Mapper exercises the controls in ascending interaction order (starting with the UI object that has the lowest InteractionOrder value).

Restoring the Default Values for UI Object Properties

If you have modified the UI object properties of an object in the application map, and you want to restore the default property values for that object, you can do so at any time.

To restore the default property values for an object:

▶ In the application map, right-click the object, and then click **Restore UI Properties** on the shortcut menu.
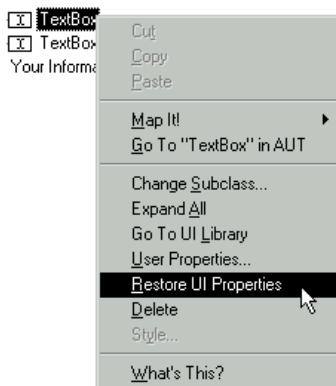
TestFactory restores all of the original default values of the UI object properties.

Creating and Mapping a Region Object for an Unmapped Control

In some instances, the Application Mapper fails to detect a control in the user interface of the AUT. The following figure shows the TestFactory window with the project for the Classics sample application open. In the application map, the Album tab object is selected, and the Image view displays a bitmap of the Album tab control. The Album tab contains an image control that a user can double-click to order the album selected. The Application Mapper did not detect or map the image control.

Unmapped image control
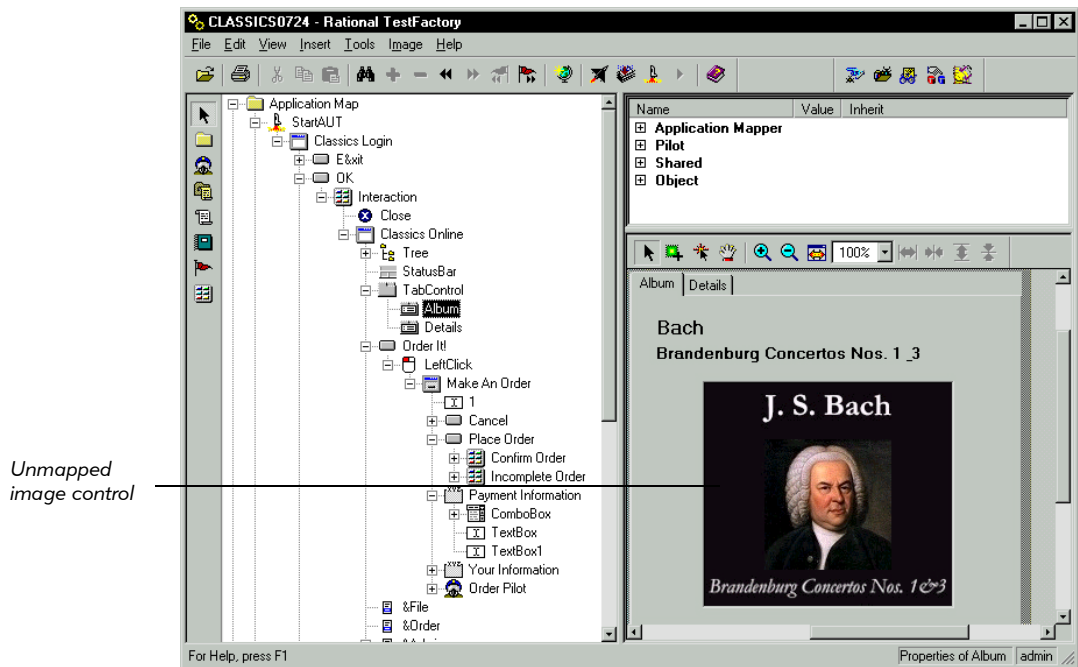
Creating a Region Object

To force the Application Mapper to detect and map an unmapped control, you must create a **region object** for it in the application map. After you create a region object that leads to unmapped areas of the user interface, remap the area of the AUT that contains the region object.

To insert a region object in the application map, use one of the following three methods:

Method 1 (recommended)

1. In the application map, click the UI object mapped for the parent of the unmapped control.

In the Classics application, the Album control is the parent of an image control that TestFactory did not map.

2. Click inside the Image view in the lower right pane.

3. Click **Draw Region** on the Image toolbar, or click **Image → Draw Region**, and then drag the pointer diagonally across the unmapped region of the parent image.

Region object tracker drawn on the album image in the Image view
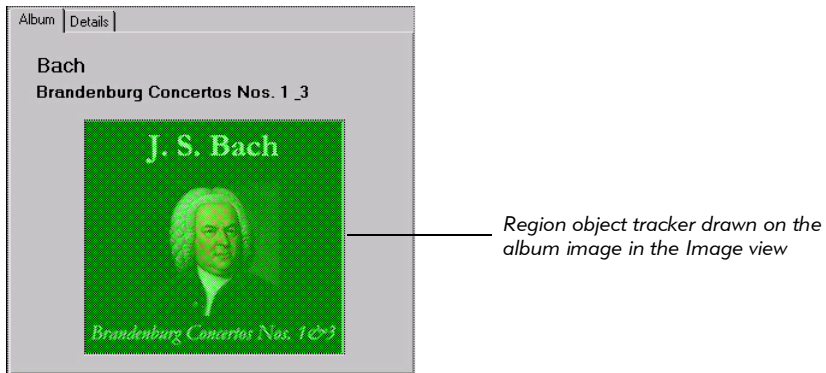
Method 2

1. In the application map, click the UI object mapped for the parent of the unmapped control.

2. Click inside the Image view.

3. Click **Image > Insert Region**.

Method 3

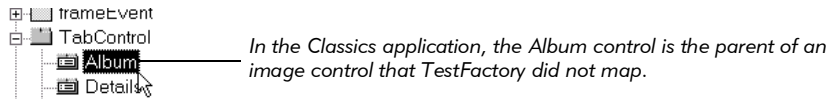1. In the application map, click the UI object mapped for the parent of the unmapped control.

2. In the Image view, right-click the parent control image, and then click **Insert Region** on the shortcut menu.

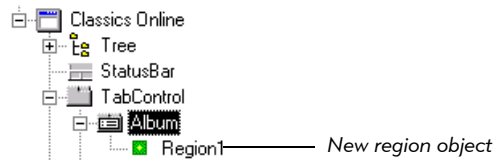TestFactory places a region object tracker on the parent image and inserts a region object under the parent UI object in the application map. The new region object is given a default name "Region*N*," where *N* represents the number of region objects created so far.

New region object

To rename the region object in the application map, click it, press F2, type a name, and then press RETURN.

Changing the Size or Position of a Region Object

You can use the region tracker that TestFactory places in the Image view to change the size and location of the region object.

Region object tracker
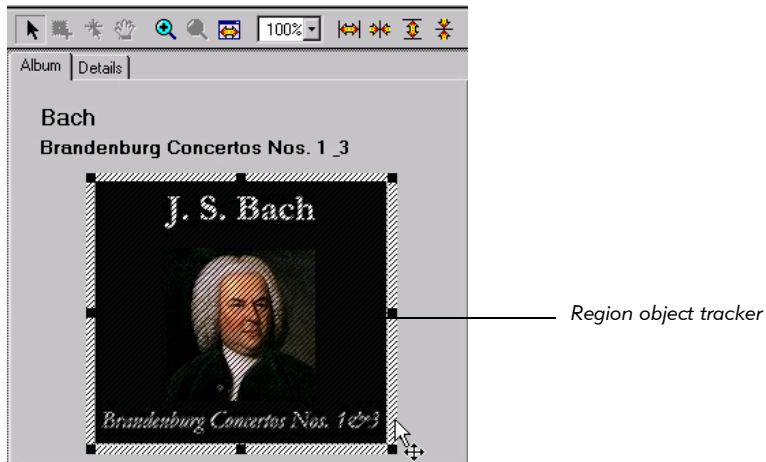
To make the tracker modifiable, click it.

To change the size of the region object tracker, do one of the following:

▶ Drag the tracker handles.

▶ Use the expand and shrink buttons on the Image toolbar.

▶ Press CTRL and then click the arrow keys on your keyboard.

To move the tracker, do one of the following:

▶ Drag the tracker to a different location in the Image view.

▶ Use the arrow keys on your keyboard.

For more information about changing the size and location of the region object, see the topic *Defining undetected controls* in TestFactory Help.

Setting the Action or Input for a Region Object

After you create a region object, you must make sure that TestFactory correctly exercises the control it represents. If the control requires a specific action or actions, you can specify these by modifying the UI object properties for the region object.If the control accepts text or requires a specific text string as input, you can select a data entry style for the region object.

To specify the correct way to exercise a region object using the UI object properties:

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

2. If you created the region object for a control that requires a specific action:

   a. In the Properties view, expand the **Application Mapper** group.

   b. Double-click the **Value** field for the correct action (such as **DoLeftDoubleClick**), and then click **Yes**.

   You can use region objects to supply input for controls that TestFactory does not fully map. For example, if you create region objects for the cells in a grid control and specify input for the region objects, you can run Pilots that test the control.

3. If you created the region object for an input control, you can assign it a style and, if necessary, specify a required string case to pass to the control. If you assign a style, select one that has an entry data composition that best matches the mix of entry data you want to apply to the control during Pilot runs. For information about how to assign a style, see *Specifying a Required String Case for Mapping* on page 4-42.

4. If you created the region object for a control that leads to an unmapped part of the AUT, then remap the area of the AUT that contains the control.

Adjusting the Hot Spot for a Region (or Other UI) Object

Every UI object has a **hot spot**, defined by x,y coordinates, that TestFactory clicks to exercise the control. By default, the hot spot lies at the center of a UI object. If necessary, you can reposition the hot spot on a UI object in the application map.

To display and move the hot spot for a UI object:

1. Click the UI object in the application map.

2. Click **Image > Move Hot Spot** or click **Move Hot Spot** on the Image toolbar.
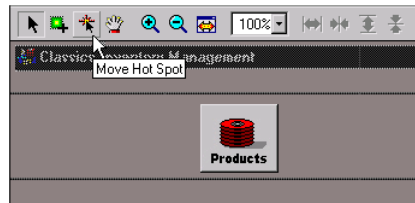
3. Drag the hot spot to a different location on the bitmap image.

4. To make the hot spot invisible, do one of the following:

   – Press ESC.

   – Click **Image > Move Hot Spot**.

   – Click **Move Hot Spot** on the Image toolbar.

5. Remap the part of the AUT that contains the UI object.

Deleting a Region Object

To delete a region object:

1. In the application map, click the region object, and then press DELETE.

2. Click **Yes** to confirm that you want to delete the selected item.

3. If you have added the region object to one or more interaction objects, then you must manually remove the region object components and remap the affected areas of the AUT.

Mapping Similar Windows

As it maps the AUT, the Application Mapper compares each new window it finds with the windows it has already mapped to determine whether the window is a unique new object or a duplicate of a window that is already mapped. The Application Mapper bases its comparison decision on the extent of the differences it finds between the window object instances it encounters. If the difference between two windows is small, the Application Mapper "decides" that the new window is a duplicate of a window already mapped, and maps it as a shortcut that points to the original instance of the window. If the difference is large, the Application Mapper maps the window as a unique window object.

The application map fragment in the following figure illustrates the correct mapping of the "Make An Order" window and the "To Make An Order" shortcut that references the first occurrence of the window in the application map. The Application Mapper first encountered the window after exercising the "Order It!" command button. The Application Mapper encountered the window again after exercising the "Place New Order" command on the Order menu.

Make An Order window object

Shortcut to the Make An Order window object
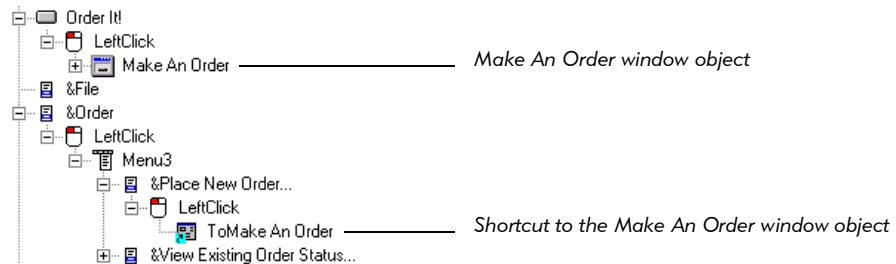
The Application Mapper can make incorrect window comparison decisions. If two unique windows are very similar, the Application Mapper can map the windows as a single window object. If a window assumes states that look very different, the Application Mapper can map two states of the same window as two separate window objects. To correct such errors, you can either directly override the Application Mapper decision, or you can adjust the Application Mapper sensitivity threshold to differences between windows.

The examples in the next three sections show how to do the following:

▶ **Merge** two window objects (after the Application Mapper incorrectly maps two states of the same window as two unique window objects).

▶ **Split** two unique windows (after the Application Mapper incorrectly maps two unique windows as a single window and a shortcut).

▶ **Redirect** a shortcut that references the wrong window.

Merging Window Objects

To override the mapping of a single window as two unique window objects:

1. Right-click the duplicate window object that should be mapped as a shortcut, and then click **Merge Windows** on the shortcut menu.
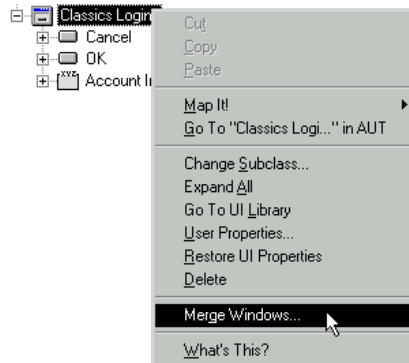

To replace a window object with a shortcut, TestFactory first deletes the window object and all of its children. Test Factory prompts you to confirm that you want it to do this.
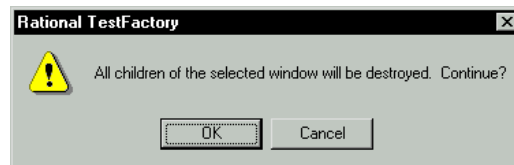
NOTE: If the branch of the application map you are about to change contains scripts, reports, or other TestFactory objects that you want to keep, move them to another location before you merge the windows. To move an object to a different location in the application map:

1. Right-click the object, and then select **Cut** or **Copy** on the shortcut menu.

2. Right-click an object in a different branch of the application map that you want as the new parent for the object you are moving, and then click **Paste** on the shortcut menu.

3. To delete the window object and all of its children, click **OK**.
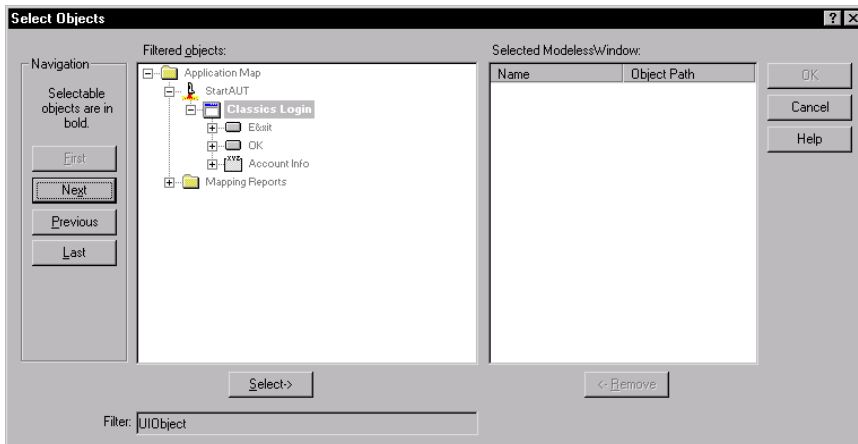
4. In the **Filtered objects** list, locate and click the window object that was mapped first, and then click **Select**.

5. Click **OK**.
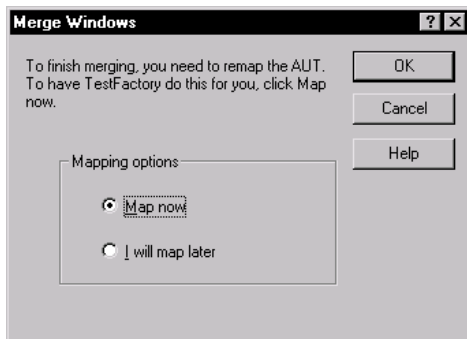
To finish merging the windows, you must remap the regions of the AUT that are affected by the change. After you do, TestFactory replaces the duplicate window object with a shortcut that points to the original window object.

6. To have TestFactory remap the affected portions of the AUT immediately, click **Map now**. If you prefer to make additional changes to the application map before remapping, click **I will map later**.

> **NOTE:** If you choose to map later, be sure to map the affected area to full depth. The correct starting object you use for mapping later depends on whether you make additional changes that also require remapping, such as splitting windows or redirecting a shortcut. For important information about selecting the correct starting object for mapping, see the topic *Merge two states of a dynamic window* in TestFactory Help.

7. Click **OK**.

Repeat these steps wherever multiple states of a single window are mismapped as unique windows.

Splitting Window Objects

The figure below shows two sections of the application map for the Classics sample application. The Suppliers and Purchase Order windows in Classics both contain a Report button, each of which leads to a dialog box used to create a report. Although the dialog boxes are very similar, they are different windows.
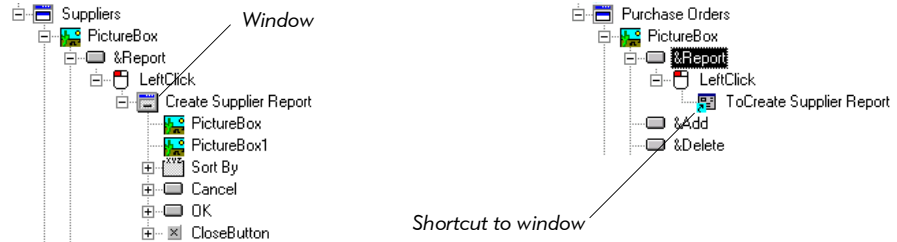
Because the dialog boxes are so similar, the Application Mapper merges them. It maps the first Report dialog it encounters as the Create Supplier Report window object, and it maps the second Report dialog box as a shortcut that points to the Create Supplier Report window object. To properly test the Create Purchase Orders Report dialog box, you must split the windows and then remap the area that contained the shortcut.

To override the incorrect mapping of two unique windows as a window object and a shortcut:

1. Right-click the shortcut, and then click **Split Window** on the shortcut menu.

2. To complete this procedure, you must remap the AUT. To have TestFactory remap immediately, click **Map now**. If you prefer to make additional changes to the application map before you remap, click **I will map later**.
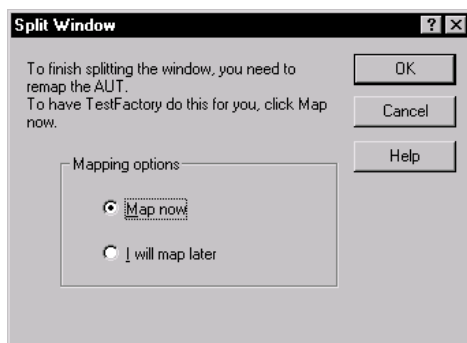
NOTE: If you choose to map later, be sure to map the affected area to full depth. The correct starting object you use for mapping later depends on whether you make additional changes that also require remapping, such as splitting windows or redirecting a shortcut. For important information about selecting the correct starting object for mapping, see the topic *Split two merged windows* in TestFactory Help.
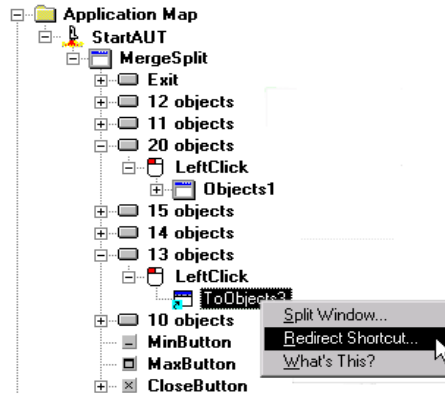
3. Click **OK**.

Redirecting a Shortcut

In the application map fragment shown below, the "ToObjects 2" shortcut references the wrong window object. If a shortcut object references the wrong window, you must redirect the shortcut before you test the corresponding area of the AUT.

To change the referenced window for a shortcut:

1. Right-click the shortcut object, and then click **Redirect Shortcut**.

2. In the **Filtered objects** list, click the correct window object.
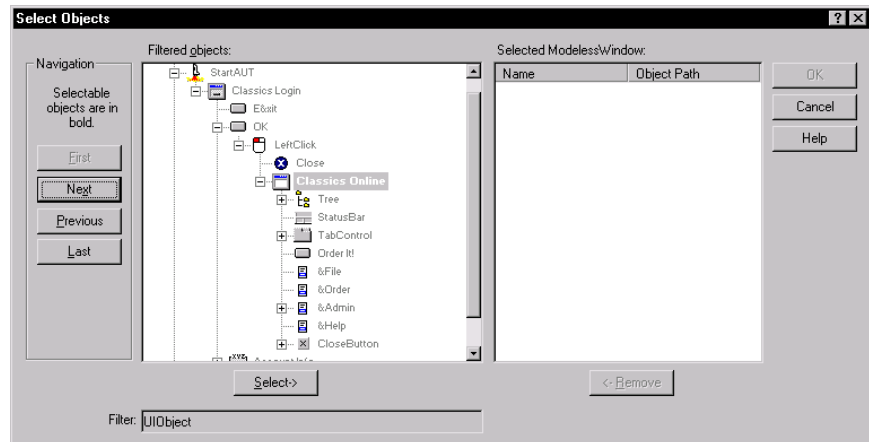
3. Click **Select**, and then click **OK**.

4. To complete the change, remap the AUT. To have TestFactory remap immediately, click **Map now**. If you prefer to make additional changes to the application map before you remap, click **I will map later**.

> **NOTE:** If you choose to map later, be sure to map the affected area to full depth. The correct starting object you use for mapping later depends on whether you make additional changes that also require remapping, such as splitting windows or redirecting a shortcut. For important information about selecting the correct starting object for mapping, see the topic *Redirect a shortcut* in TestFactory Help.

Adjusting Sensitivity to Differences in Window Objects

If you find the Application Mapper is too sensitive or is not sensitive enough to differences between windows in the AUT, you can change the **WindowMatchThreshold** property value that the Application Mapper uses to make its window comparison decision. If you change the WindowMatchThreshold value for a window class in the UI library, the threshold setting affects all subsequent comparisons of windows belonging to that class. If you change the value for a specific window object in the application map, the threshold setting affects only that window object.

The WindowMatchThreshold property has a default percent value of 80. This setting affects mapping in the following way:

▶ If the Application Mapper compares two windows and finds that at least 80 percent of their characteristics match, then it merges them and maps the windows as a single instance of the MainWindow UI object class. The first instance encountered is mapped as a window UI object. A subsequent instance is mapped as a shortcut that points to the original window object.

 Window object

 Shortcut object

▶ If less than 80 percent of the characteristics in two windows match, then the Application Mapper places two separate window UI objects in the application map.

To adjust Application Mapper sensitivity globally for all subsequent mappings of window controls belonging to a window UI object class:

1. Expand the UI Library folder.

2. Click the window UI object class for which you want to adjust the comparison threshold.

3. In the Properties view, click **Application Mapper**.

Application Mapper properties for a window object class

Name	Value	Inherit
⊟ **Application Mapper**		
ExerciseDuringMapping	Yes	Yes
DoLeftClick	No	Yes
DoLeftDoubleClick	No	Yes
DoRightClick	Yes	Yes
WindowMatchThreshold	80	Yes
UseWaitForIdle	Yes	Yes
⊞ **Pilot**		

WindowMatchThreshold property

4. In the **Value** field for **WindowMatchThreshold**, type a new percentage value.

   A value greater than 80 makes the Application Mapper more sensitive to differences between the window objects it compares and increases the chance that two states of the same window are mapped as two unique window objects.

   A value of less than 80 makes the Application Mapper less sensitive to the differences in the windows and increases the chance that two unique windows are mapped as a single window object.

To adjust Application Mapper sensitivity locally for the next mapping of a specific window:

1. Click the window UI object in the application map that the Application Mapper incorrectly merged or split.

2. In the Properties view, click **Application Mapper**.

3. In the **Value** field for **WindowMatchThreshold**, type a new percentage value.

4. Remap the affected portion of the AUT. Use the window object for which you changed the WindowMatchThreshold property as the starting object for mapping.

Timing Events During Mapping

An AUT can contain controls that require time to respond after they are exercised. If the Application Mapper does not allow enough time for a control to respond, it fails to map beyond that control. To solve this timing problem, you can impose a wait interval that forces the Application Mapper to wait for the AUT to respond.

You can use several different methods to adjust the timing TestFactory uses to exercise controls in the AUT. You can adjust timing for all controls in the application, for an entire class or subclass of controls, or for just a single control.

Specifying a Maximum Wait-For-Idle Time for All Controls

During mapping, TestFactory uses an idle detection routine to determine the active status of the AUT. After TestFactory exercises a control, it waits up to 20 seconds (the default maximum wait-for-idle time) for the AUT to become idle before it exercises the next control. If the AUT requires more than 20 seconds to perform most operations, you can reset the maximum wait for idle value for mapping.

To specify a new maximum wait-for-idle time:

1. Click **Tools > Options**, and then click the **Application Mapper** tab.
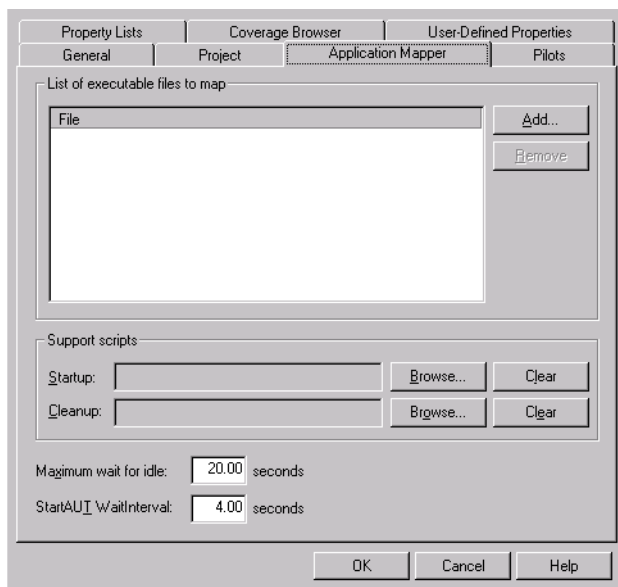
2. In the **Maximum wait for idle** box, type a new value, in seconds.

> **NOTE:** If the AUT is a Java application or applet, the Application Mapper tab displays the **Extended WaitInterval** box, which contains a 0.50-second default value. The **StartAUT WaitInterval** box displays a 20.00-second default value. TestFactory changes these values so that mapping does not time out waiting for the JVM to become idle.

Timing Events for a Class or Subclass of Controls During Mapping

To control the timing of events for an entire class or subclass of controls during mapping, you can set the **WaitInterval property** for the object class or subclass.

To specify a value for the WaitInterval property for an object class or subclass:

1. In the application map, right-click a UI object that belongs to the object class or subclass for which you want to adjust timing, and then click **Go To UI Library** on the shortcut menu.

2. In the Properties view, click **Shared**.

WaitInterval property

3. Click the **Value** field for **WaitInterval**, and then type in the number of seconds for TestFactory to wait after it exercises a control that belongs to this object class.

> **NOTE:** Because WaitInterval is a Shared property, TestFactory also imposes the WaitInterval value that you specify during Pilot runs.

During mapping, TestFactory waits the specified number of seconds after exercising a control before it continues to map.

4. To use the WaitInterval value as the absolute wait time, instead of the minimum wait time, click **Application Mapper** in the Properties view, and then change the **UseWaitForIdle** property value to **No**.

UseWaitForIdle property

Timing Events for a Single Control During Mapping

To control the timing of events for a single control in the AUT during mapping, you can set the WaitInterval property for the corresponding UI object in the application map.

To set the WaitInterval property for a single control in the AUT:

1. In the application map, click the UI object mapped for the control in the AUT.

2. In the Properties view, click **Shared**.

3. Click the **Value** field for **WaitInterval**, and then type in the number of seconds for TestFactory to wait after it exercises the control.

4. To use the WaitInterval value as the absolute wait time, instead of the minimum wait time, click **Application Mapper** in the Properties view, and then change the **UseWaitForIdle** property value to **No**.

You can also change the timing of events for a single control by setting a WaitInterval value for the action used to exercise the control. For example, after a user left-clicks a command button in the AUT, a dialog box opens, but only after a 15-second delay. To adjust the timing for this delay during mapping, you could either set a WaitInterval value for the control button UI object, or you could set a WaitInterval value for the LeftClick action object mapped just above the dialog box object.

The following table shows how TestFactory imposes the wait interval based on whether you set the value for a UI object or action object in the application map, or for an object class or action class in the UI library.

WaitInterval property value set for	Affects
A UI object in the application map	Only the specific UI object
An action object in the application map	The specific action object and its top-level child UI object in the application map
A UI object class in the UI library	All UI objects in the application map that belong to the object class
An action object class in the UI library	All top-level child UI objects in the application map that respond to the action

Reclassifying a Generic Object

Although TestFactory has an extensive library of object classes, it cannot recognize every control that it encounters. Software developers create control types that are not represented in the UI library. If the Application Mapper encounters a control that it does not recognize, it classifies the control as a **generic object** and creates a unique generic object subclass for it.

The following figure shows a portion of the application map for the Classics sample application. The Create Supplier Report window in Classics contains two owner-created check box controls that TestFactory cannot find among the existing object classes in the UI library. These are mapped as generic objects.
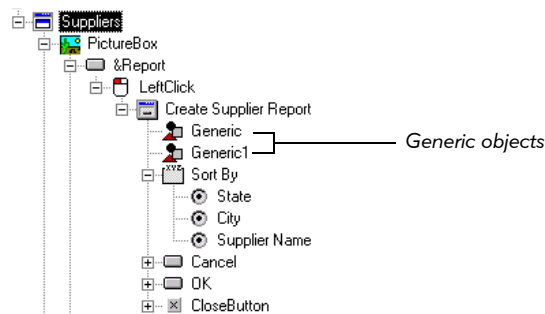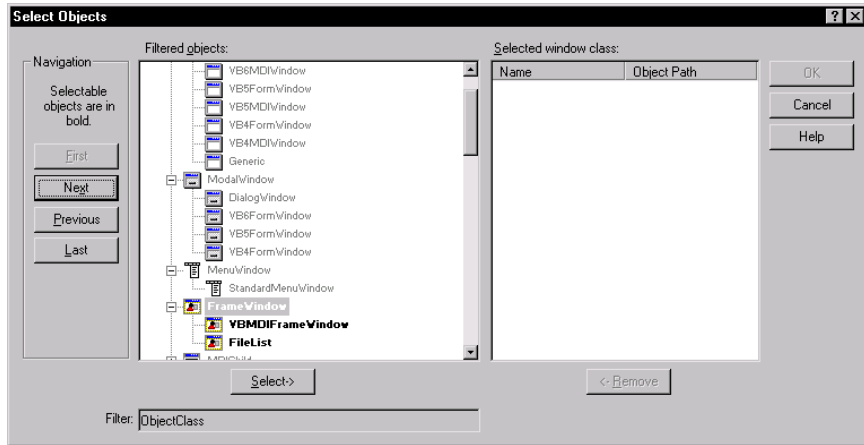

Generic objects

If you run a Pilot against an area of an application map that contains a generic object, the Pilot fails to recognize the generic object and excludes it, and all of its child objects, from the scripts it creates. To solve this problem, you must reclassify the generic object before you test it.
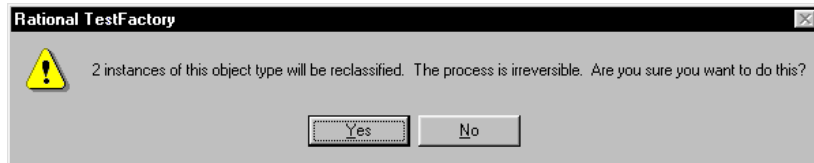
To reclassify a generic object:

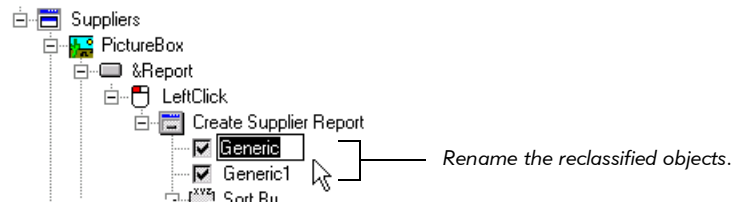1. In the application map, right-click the generic object, and then click **Reclassify** on the shortcut menu.
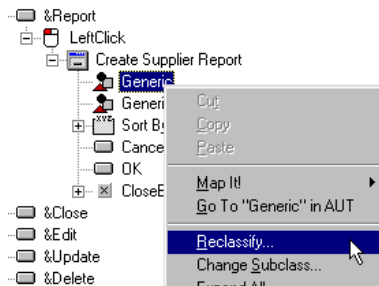
2. In the **Filtered objects** list, locate and click the appropriate UI object class for the unrecognized control.

3. Click **Select**, and then click **OK**.

4. Click **Yes** to confirm that you want to reclassify the object(s).

Rename the reclassified objects.

5. In the application map, rename the now reclassified UI objects.

After you reclassify the generic objects, TestFactory recognizes them as known UI objects during all subsequent mapping and testing sessions and can better test the objects they represent.

Handling Error Messages and Crash Transition Objects in the Application Map

If the Application Mapper encounters an error message during mapping, TestFactory places a window object that represents the error message dialog box in the application map. If the AUT crashes during mapping, TestFactory places a crash transition object that shows where the AUT crashed in the application map. In either case, you can make adjustments so that the Application Mapper maps all of the available paths in the AUT.

If the application map contains a window that represents an error message dialog box, then do the following:

1. Determine what led to the error message. The error message could have been activated if TestFactory did not pass required input to controls in the AUT, or if it did not exercise controls in a specific order.

2. In the application map, leave the path mapped to the error message.

3. Set up an interaction object that guides the Application Mapper through the unmapped path in the user interface.

4. To map the path that does not lead to the error message, map the area for which you set up the interaction object.

5. Run a Pilot to test the new mapped path.

If the application map contains a crash transition object:

1. Leave the crash transition object in the application map until the defect that caused the crash is fixed.

   NOTE: If you run a Pilot in an area of the application map that contains a crash transition object, the Pilot run automatically excludes the controls that caused the AUT to crash.

2. After the defect that caused the crash is fixed, right-click the crash transition object in the application map, and then click **Delete** on the shortcut menu.

3. After TestFactory prompts you to confirm that you want to delete the object, click **Yes**.

4. Remap the affected portion of the AUT.

5. Run a Pilot to test the affected area of the AUT.

Mapping New Builds

We recommend that you map each new build of the AUT if the user interface of the AUT has changed. This ensures that the application map reflects changes made since the previous build and improves the quality of the scripts that Pilots generate.

> **NOTE:** Be sure to instrument and rebuild the executable file for each new build of the AUT before you map it. If you added the ActiveX Test Control to the previous build of the AUT, be sure to add it to all subsequent builds before you instrument.

If several regions of the AUT have changed since you last mapped it, remap it using StartAUT as the starting object. If only a small region of the AUT has changed, you can remap just the changed region using the Application Mapper Wizard or the **Map It!** shortcut.

Mapping a Changed Region of the AUT Using the Map It! Shortcut

To map a changed region of the AUT using the **Map It!** shortcut:

1. In the application map, right-click the top UI object in the branch that corresponds to the changed region of the AUT. For example, if a group box control contains options that have changed, right-click the group box object in the application map. This is the starting object for mapping.

2. To start mapping, point to **Map It!** on the shortcut menu, and then click **Full** or **Single Level** to indicate depth of mapping.
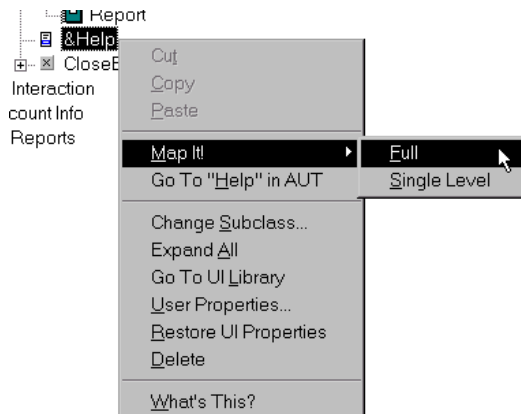
Mapping a Changed Region of the AUT Using the Application Mapper Wizard

To map a changed region of the AUT using the Application Mapper Wizard:

1. Click **Tools > Application Mapper**.

2. In steps 1 and 2 of the wizard, click **Next**.

3. To select the starting object for mapping, click **Browse** in step 3.
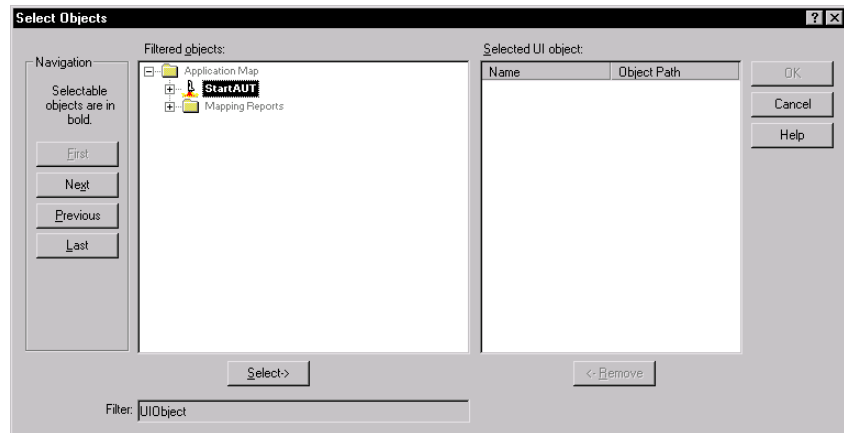
4. Scroll through the **Filtered objects** list, or click **First**, **Next**, **Previous**, and **Last** to jump to selectable objects in the list.

5. Click an object in the **Filtered objects** list, and then click **Select**.

6. Click **OK**.

7. To start mapping, click **Finish**.

Deleting UI Objects Mapped for Controls that Have Been Removed from the AUT

If a mapped control is removed from the user interface of the AUT, the UI object for that control persists in the application map until you remove it. The Summary Report lists controls that were mapped in previous builds, but were not encountered in the last mapping session. After you map a new build of the AUT, check the Summary Report for missing objects.

After you determine that a missing control was intentionally removed from the user interface, you must delete the UI object mapped for it in the application map. If you run a Pilot to test the affected area of the AUT, the Pilot "expects to see" the mapped control. If the application map still contains a UI object for the missing control, the

Pilot creates a UAW script to indicate that it encountered an unexpected active window. (For information about UAW scripts, see *Viewing a UAW Script* on page 5-19.)

Deleting UI Object Components Associated with a Deleted UI Object

If you try to delete a UI object that has been added as a component to one or more interaction objects, you must also delete the interaction object components.

To delete a UI object that has been added to one or more interaction objects:

1. In the application map, click the UI object to delete, and then press DELETE.
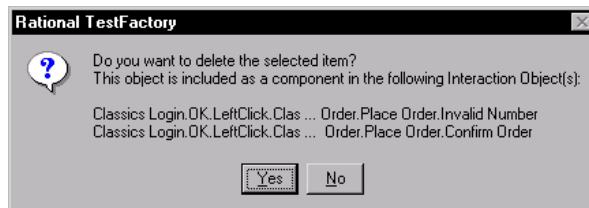
If you have left the **Prompt on object deletion** check box selected on the **General** tab of the Options dialog box, TestFactory displays a message listing up to ten interaction objects in the application map that contain the component.

2. To delete the IU object and all of the components derived from it, click **Yes**.

In each interaction object that contained the component, the name of the deleted component is displayed in gray text and Strikethrough font style. The **Interaction Method** value is set to **Deleted**. The location of a deleted component is marked in this way so that you can easily find it and, if necessary, replace it with a different component. Leaving the name of a deleted component in an interaction object does not affect mapping or testing.

If deleting a component changes the user interface exposed by an interaction object, be sure to remap the interaction object after the deletion and before you test the affected area of the AUT.

Running the Application Mapper from the Command Line

You can run the Application Mapper from the command line. TestFactory accepts Application Mapper command-line arguments that let you specify the path to the AUT, the starting object for mapping, single-level or full-depth mapping, arguments to pass to the AUT, and the working directory for the AUT. For information about how to run the Application Mapper from the command line, see the Appendix, *Using TestFactory Command-Line Arguments* or see the topic *Mapping command line arguments* in TestFactory Help.

Mapping Secondary Applications

An AUT can consist of a main application and one or more 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. If the AUT calls secondary executable files, you can map these in addition to the main executable file.

If you specified a secondary Visual Basic application to instrument before mapping, TestFactory automatically lists its executable file on the **Application Mapper** tab. This gives the Application Mapper access to all of the controls in the secondary executable file.

To prevent the Application Mapper from mapping an instrumented secondary application to full depth during subsequent mapping sessions:

1. Click **Tools > Options**, and then click the **Application Mapper** tab.
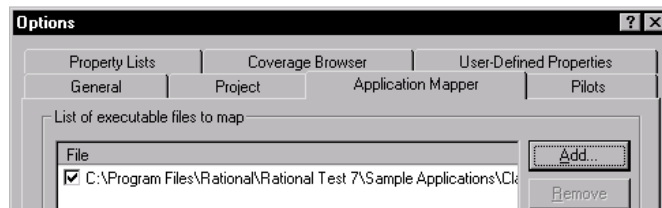
2. To exclude a listed executable file from mapping, do one of the following:

    –   In the **List of executable files to map** box, clear the check box to the left of the listed executable file.

    –   Click the file path, and then click **Remove**.

3. Click **OK**.

After you next map the AUT, the Mapping Summary report displays the secondary executable files that were excluded from full-depth mapping.

If you did not instrument a secondary application (Visual Basic only), the Application Mapper maps just the main application and the first level of controls in the secondary application. If you plan to fully map and test a secondary application written in Visual Basic, we recommend that you instrument it before mapping. For information about instrumenting a secondary Visual Basic application, see *Instrumenting Visual Basic Source Code* on page 3-7.

To fully map an uninstrumented secondary application that you do not plan to test:

1. Click **Tools > Options**, and then click the **Application Mapper** tab.
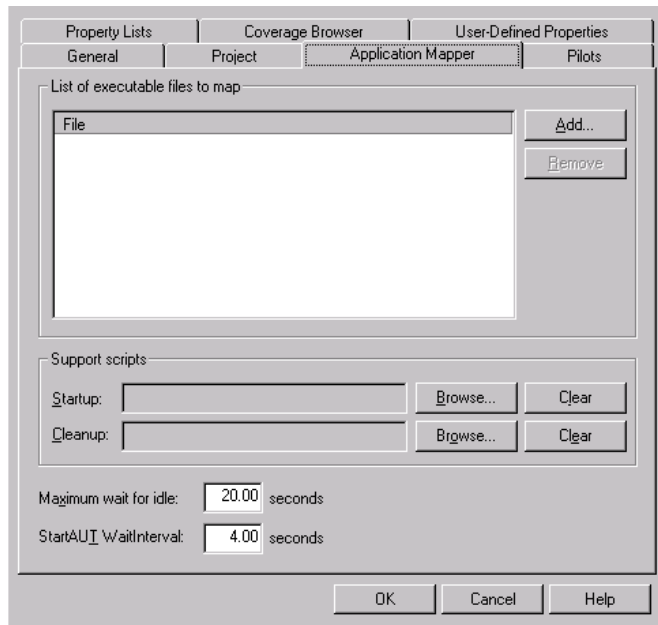
2. Click **Add**, and then browse to find and select the executable file for the secondary application.

3. Leave the check box next to the file name selected.

4. Click **OK**.

Inserting TestFactory Objects in the Application Map

The Insert toolbar along the left side of the TestFactory window provides buttons that you can use to insert several types of **TestFactory objects** in the application map. TestFactory objects include folders, Pilots, Test Suites, scripts, reports, markers, and interaction objects. For a description of each insertable TestFactory object, see *The Insert Toolbar* on page 2-11. The correct use of each TestFactory object type is addressed separately in different sections of this manual.

To insert a TestFactory object in the application map, do one of the following:

▶ Click a button for a TestFactory object on the Insert toolbar, and then click a destination for the object in the application map.

▶ Drag a TestFactory object from the Insert toolbar to a destination in the application map.

▶ Click a destination in the application map, and then click **Insert > Folder** (or another TestFactory object).

To insert multiple instances of a TestFactory object in the application map, press SHIFT, and then click a button on the Insert toolbar. The icon for the object attaches to the pointer. You can continue to click and insert the object at multiple locations in the application map. To restore the pointer to the select mode, click **Select** on the Insert toolbar or press ESC.
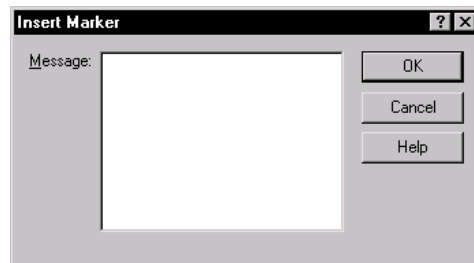
Creating a Marker in the Application Map

You can insert a marker in the application map to use as a bookmark or to enter notes. The markers are for your use only. They do not affect application mapping or testing.

To create a marker:

1. Insert a marker at a destination in the application map.

2. Type a note to append to the marker in the **Message** box, and then click **OK**.

Date/Time Modified	Entered by	Message
🛈 04/17/98 14:46	ADMIN	Cancel button added 4/16/98

Marker view in the right pane

The Marker view in the right pane displays the date and time the marker was inserted or last modified, the user who created it, and the message text.

3. To rename a marker object, click it in the application map, press F2, and then type a name.

4. To edit the marker message, double-click the **Message** field in the Marker view, and then type a new message.

5. To move through marker objects in the application map in sequence, use **Previous Marker** and **Next Marker** on the Standard toolbar.

Creating and Working with TestFactory Reports

You can insert a TestFactory report object to create 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.

To create a TestFactory report:

▶ Insert a report object in the application map.

NOTE: The location in the application map does not affect the report contents. You can create a "Reports" folder to hold all of your reports.
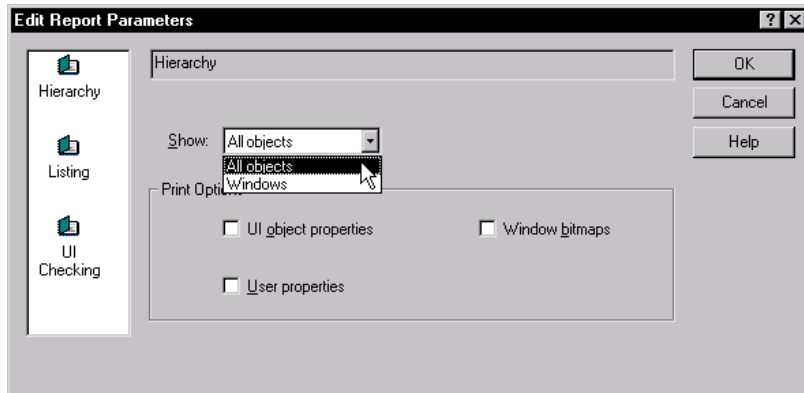
The box on the left side of the Edit Report Parameters dialog box displays icons for **Hierarchy**, **Listing**, and **UI Checking** reports. A Hierarchy report includes data on all UI objects in the application map. A Listing report can include data on UI objects, Pilots, Test Suites, and scripts. A UI Checking report provides information on mnemonics conflicts, improper alignment of controls, and other possible problems detected in the graphical user interface of the AUT.

Configuring a Hierarchy Report

To configure a Hierarchy report:

1. To include all objects in the application map in the Hierarchy report, leave **All objects** displayed in the **Show** list. To include all window UI objects and their immediate child objects in the report, click **Windows** in the **Show** list.

2. To include UI object properties in the printed version of the report, under **Print Options**, select the **UI object properties** check box.

3. To include user properties in the printed version of the report, under **Print Options**, select the **User properties** check box.

4. To include bitmap images of the window objects in the printed version of the report, under **Print Options**, select the **Window bitmaps** check box.

5. To run the report, click **OK**.

Configuring a Listing Report

To configure a Listing report:

1. Click the **Listing** report icon.
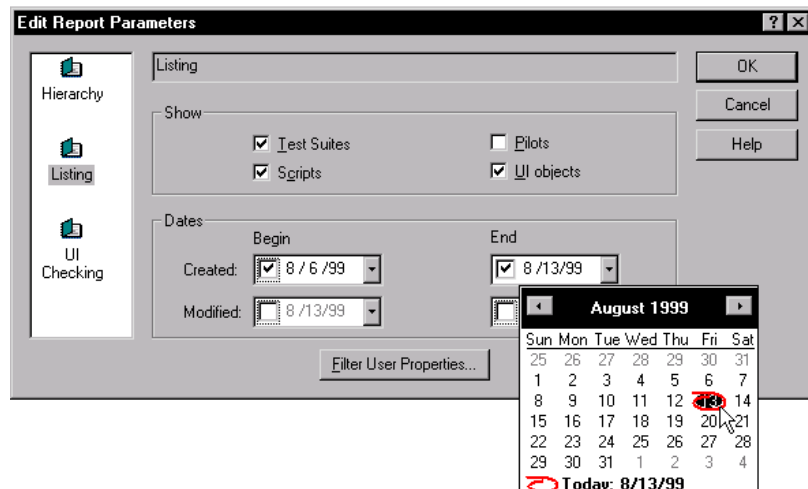
2. Under **Show**, select the check boxes for the types of objects that you want to include in the report.

3. To filter objects created during a specific time interval, under **Dates**, click the arrow in the **Begin** box for **Created**, and then click a begin date for the interval. Click the **End** box arrow, and then click an end date for the interval.

4. To filter objects modified during a specific time interval, under **Dates**, click the **Begin** box arrow for **Modified**, and then click a begin date for the interval. Click the **End** box arrow, and then click an end date for the interval.

5. To filter objects based on user property values, click **Filter User Properties**, and then specify values in the Filter User Properties dialog box.

6. To run the report, click **OK**.

Configuring a UI Checking Report

To configure a report about possible problems TestFactory detects in the graphical user interface of the AUT:

1. Click the **UI Checking** report icon.
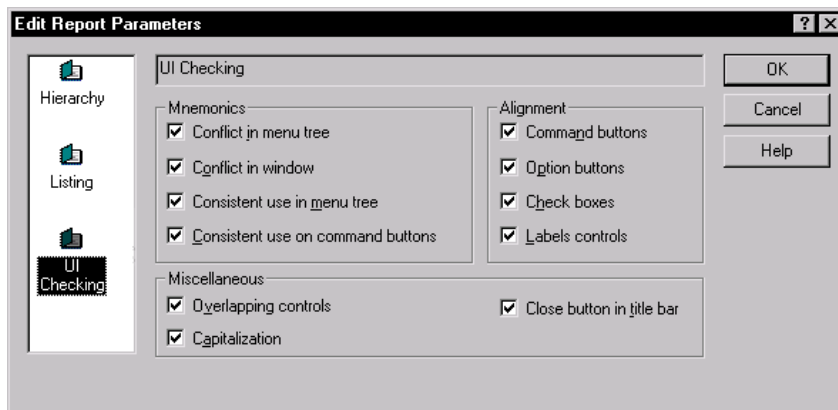
2. Clear the check boxes for features that you do not want to use to generate the UI Checking report.

3. To run the report, click **OK**.

The following figure shows the contents of a UI checking report that was run for the Classics sample application.

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.

The **Name** column shows a bitmap of the UI object mapped for the control and a description of the problem that was detected. The **Object Path** column displays the final path of the UI object mapped for the control in the application map. For example, the value **Make An Order.Place Order** indicates that a problem was detected for the Place Order button in the Make An Order window.

To jump from an object listed in a report to the corresponding object in the application map:

▶ Double-click the object listed in the report.

To return to the report contents:

▶ Click **Previous Object** on the Standard toolbar.

Modifying a Report

To modify a report:

1. Click the report object in the application map, and then click **Report → Edit**.

   Alternatively, click the report object in the application map, and then click **Edit Parameters** on the Report toolbar.

2. Change the report parameters, and then click **OK** to rerun the report.

Rerunning a Report After Changing the Application Map

To rerun a report after you change the application map, do one of the following:

▶ Click the report object in the application map, and then click **Report > Run**.

▶ Click the report object in the application map, and then click **Run Report** on the Report toolbar.

Exporting a Report as a Text File

To export a report as a text file, do one of the following:

1. Click the report object in the application map, and then click **Report > Export**.

   Alternatively, click a report object in the application map, and then click **Export** on the Report toolbar.

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

Printing a Report

To print a report:

1. Click the report object in the application map.

2. Click **File → Print**, or click **Print** on the Standard toolbar.

▸ ▸ ▸   C H A P T E R   5

Automatically Generating Scripts

This chapter describes how to use TestFactory to automatically generate scripts that test the AUT. It provides instructions on how to insert, set up, and run a Pilot, as well as how to analyze the results of a Pilot run. The final sections describe how to create custom TestFactory scripts and how to record new user actions in Pilot-generated scripts. This chapter includes the following topics:

▶   About Pilots

▶   Setting up and running Pilots

▶   Examining Pilot run results

▶   Changing default settings for new Pilots

▶   Opening and editing a best script in Robot

▶   Creating a custom TestFactory script

▶   Checking for memory defects in Visual Basic and C++ applications (Windows NT)

▶   Improving Pilot-generated scripts

▶   Testing controls in the AUT during Pilot runs

About Pilots

The Pilot is the workhorse of TestFactory. Its automatic scripting capability mechanizes the writing of script code necessary for regression testing. You can drop a Pilot at any location in the application map. From there, the Pilot automatically generates scripts that focus on specific functional areas of the AUT.

During a Pilot run, a Pilot goes progressively deeper into the code of the AUT and uses the script segments it generates to build a script that gives maximum coverage of the AUT with a minimum number of script segments. The result is an optimized best script that provides extensive code coverage and contains a minimum of redundant script code.

Your objectives for creating test scripts depend on what phase of development the AUT is in. Early in the development process, and during phases of rapid change in the AUT, it is more important to flush out severe defects than to create scripts that provide high coverage and have longevity.

Traditionally, engineers did not bother to automate testing at the beginning stages of product development because of the high cost of maintaining scripts. With TestFactory, the cost of script maintenance is so low that you can incorporate automation in your development cycle as soon as you have a user interface. Early in product development, you can run Pilots as a smoke test to flush out defects quickly.

The Pilot results include defect scripts that uncover severe bugs such as AUT crashes, Visual Basic run-time errors, and assertions. You can use Pilot runs early in the development cycle and against each new build of the AUT to spot areas of code instability. You can run Pilots after fixing severe defects to uncover new defects and to ensure that the AUT has not regressed.

The Pilot run results can also include UAW (unexpected active window) scripts. If an unexpected active window opens during a Pilot run, TestFactory pulls the running script segment and saves it as a UAW script. You can examine a UAW script to determine what steps a Pilot performed to activate an unexpected window, and use the information to improve the application map.

Although you continue to look for defects as the AUT matures and stabilizes, at some point you become more concerned with how completely your scripts test the product and how well the AUT meets functional requirements. As functional areas of the AUT stabilize, you can begin running Pilot scenarios to simulate action sequences that users are likely to perform. You can also compose Pilot mix-ins to test the interaction of several different functions or to occasionally introduce a new functional element to an otherwise ordered Pilot scenario.

Late in the development cycle, once the AUT is free of severe defects and has been proven to meet its functional requirements, you can set up and run Pilots that check for memory-related defects in the AUT. Pilots can generate defect scripts that uncover memory and resource leaks, invalid memory access errors, memory overwrites, uninitialized memory reads, and memory access beyond the bounds of an array.

Setting Up and Running Pilots

In addition to providing information about setting up and running a Pilot, this section describes several issues to consider as you insert Pilots in the application map.

Effective Pilot Placement

Pilot placement in the application map determines how useful the test results will be. Keep the following factors in mind as you insert Pilots in the application map:

▶ Pilots are most effective if you insert them at UI objects that lead to major functional areas of the AUT.

▶ A Pilot uses the application map, and not the AUT, to build scripts. Place Pilots in regions of the application map that you have mapped to full depth and that include all of the paths available in the part of the AUT you want to test.

▶ Inserting a Pilot too low on a branch of the application map (so that it includes too few UI objects) can result in small code coverage values.

▶ As you combine multiple Pilots in scenarios and mix-ins, be sure to place each Pilot so that it has exclusive access to the controls in its region of the application map. In other words, combine Pilots with access to functional areas that do not overlap.

▶ If your goal is to obtain an optimized best script to test a region of the AUT, insert Pilots at map locations that correspond to stable areas of the AUT. If you just want to run Pilots to find defects, stability is not as important a factor.

▶ If you want to run a Pilot to test an area of the AUT that includes a control that you do not want to test, do one of the following:

    – Set the ExerciseDuringTesting property for the UI object mapped for the control to **Never** (see *Managing Data Entry Styles* on page 5-46).

    – Exclude the control from testing by adding the UI object mapped for the control to the Pilot **Exclude** tab.

    – If the control is represented as a component of an interaction object, make the component unavailable in the Interaction Object view.

Inserting a Pilot

1. To insert a Pilot, do one of the following:

   – Click a destination in the application map, and then click **Insert > Pilot**.

   – Click **Pilot** on the Insert toolbar, and then click a destination in the application map.

   – Drag a Pilot from the Insert toolbar to a destination in the application map.

2. Name the Pilot, and then press ENTER.

Setting Up and Starting a Pilot Run

The tabs on the Pilot properties page in the right pane contain settings that you can modify to set the parameters of a Pilot run. The following procedures describe how to modify the settings on each tab for a simple Pilot run on your local machine, how to restore default Pilot settings, and how to start the Pilot run.

Modifying Settings on the Setup Tab

The settings on the **Setup** tab let you set general parameters for the Pilot run.

To modify **Setup** tab settings:

1. After you insert a Pilot, click the **Setup** tab in the right pane.
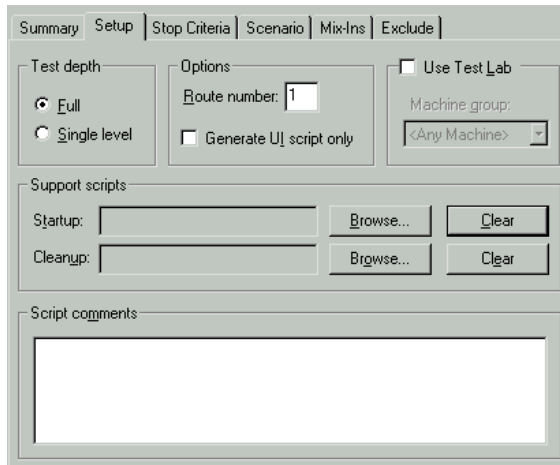
Just as you can map the AUT in increments, you can also test the AUT in increments. In full-depth testing, the Pilot drives to the base state (the UI object at which you inserted the Pilot) and from there, exercises every control it encounters at all levels of the AUT. In single-level depth testing, the Pilot drives to the base state and exercises the control it represents, but does not exercise other controls that are exposed.

2. To let the Pilot explore its area of the application map to full depth, under **Test depth**, leave **Full** selected. To limit Pilot exploration to the top level of UI objects that it encounters in the application map, click **Single level**.

   TestFactory passes the value specified in the **Route number** box to the random number generator to determine the starting path that the Pilot takes to generate scripts.

3. If you have run this Pilot before, and you want it to take a different starting path on the next run, under **Options**, type a new number between 1 and 99 in the **Route number** box. The route number you specify to change the run path is not important, as long as it is different than the previous one.

4. To generate just a UI script, and prevent the Pilot from generating other kinds of scripts:

   a. Under **Options**, select the **Generate UI script only** check box.

   After you select the check box, the **UI Script** button replaces the **Start** button at the bottom of the Pilot properties page.

   b. To start the Pilot run and quickly generate a UI script, click **UI Script**.

5. If the **Use Test Lab** check box is selected, clear it.

   To make external resources available to the AUT during testing, use the boxes under **Support scripts**. For example, to initialize a database of customer names and give the AUT access to it, you can create a Robot script that initializes the database, and then specify that script as a startup script for the Pilot run. You can also create a Robot support script that sets persistent defaults in the AUT. To restore the system to its previous state after testing, create a cleanup script in Robot, and then specify it as a cleanup script in the **Setup** tab.

6. To specify support scripts for a Pilot run:

   a. Under **Support scripts**, click **Browse** next to the **Startup** box, and then locate and select a Robot script that brings the system to the state appropriate for the Pilot run.

   b. Under **Support scripts**, click **Browse** next to the **Cleanup** box, and then
   locate and select a Robot cleanup script.

7. Type comments about the current Pilot run in the **Script comments** box. The
Pilot inserts the text as a comment at the top of the scripts that it generates.

Modifying Settings on the Stop Criteria Tab

The settings on the **Stop Criteria** tab let you specify the criteria that must be met to
complete the Pilot run. Testing stops after the Pilot run reaches any one of the criteria
that you set on this tab.

> **NOTE:** If you run a Pilot to generate just a UI script, TestFactory ignores the
> **Stop Criteria** tab settings.

To modify the stop criteria:

1. Click the **Stop Criteria** tab.
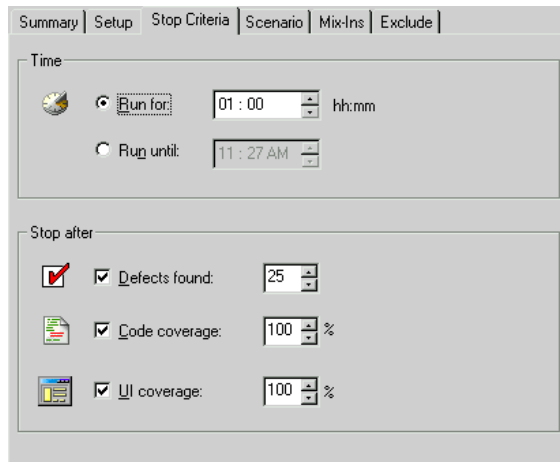
2. You must specify a time-related stop criterion for a Pilot run. You can either
specify a run duration, or specify a clock time for the run to stop.

   – To specify the duration of the Pilot run in hours and minutes, under **Time**,
   leave **Run for** selected, and then enter a time limit in the **hh:mm** box. The
   minimum run duration you can set is five minutes (00:05).

   – To specify a clock time for the run to stop, under **Time**, click **Run until**, and
   then, in the adjacent box, enter the clock time for the run to stop. The latest
   clock time you can set is 23 hours and 59 minutes from the current time.

3. To stop a Pilot run after a specified number of defects are detected, leave the **Defects found** check box selected and enter a target number in the adjacent box.

4. To change the code coverage target value, leave the **Code Coverage** check box selected and enter a new value in the adjacent box.

5. To change the UI coverage target value, leave the **UI Coverage** check box selected and enter a new value in the adjacent box.

   If the goal of the Pilot run is to achieve high code coverage, consider excluding UI coverage as a stop criterion. If the Pilot has access to only a small region of the AUT, its scripts can achieve 100% UI coverage long before they achieve the maximum code coverage possible.

> **NOTE**: To specify default stop criteria settings for all new Pilots, use the **Pilots** tab in the Options dialog box. For instructions, see *Changing Default Settings for Pilots* on page 5-29.

Modifying Settings on the Exclude Tab

If you run the Pilot in an area of the AUT that contains a control that you do not want to test, you can use the **Exclude** tab to prevent TestFactory from exercising it during the Pilot run.

To exclude a control in the AUT from testing:

1. Click the **Exclude** tab.
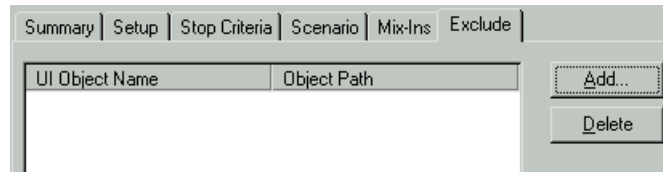
2. Click **Add**.
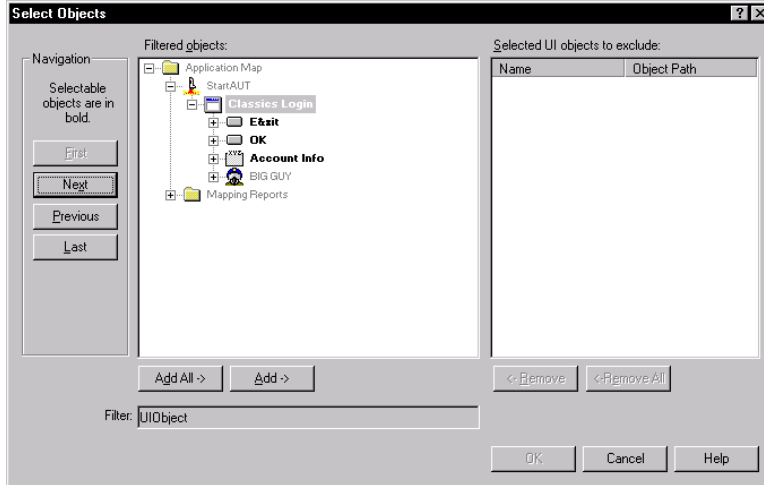
3. In the **Filtered objects** list, select the UI object mapped for the control that you do not want to test.

4. Click **Add**.

5. Click **OK**.

> **NOTE:** To exclude a UI object from *all* testing, change the value of its ExerciseDuringTesting property to Never. A UI object listed on the **Exclude** tab is only excluded from runs of the selected Pilot.
>
> To exclude an interaction object from all testing, right-click the interaction object in the application map, and then click **Use During Testing** on the shortcut menu.

Restoring Default Pilot Settings

If you want to restore the default settings for a Pilot, you can do so at any time.

To restore the default settings for a Pilot:

▶   Click **Reset** at the bottom of the Pilot properties page.

TestFactory restores the Pilot settings to their default values, some of which are specified on the **Pilots** tab on the Options dialog box. For information about default settings on the **Pilots** tab, see *Changing Default Settings for Pilots* on page 5-29.

Making Best Script Verification Unavailable

After a Pilot generates a best script, TestFactory runs the script to verify that it contains no script segments that uncover defects or unexpected active windows, and to calculate more accurate code coverage results for the best script. Script verification can significantly increase the duration of a Pilot run. You can shorten Pilot runs by making the best script verification feature unavailable.

To make the best script verification feature unavailable:

1. Click **Tools > Options**, and then click the **Pilots** tab.
2. Under **Options**, clear the **Verify best scripts** check box.
3. Click **OK**.

After you make best script verification unavailable, your Pilots will take less time to run. However, without verification, you run the risk of generating best scripts that uncover defects or unexpected active windows when you run them later.

Starting the Pilot Run

Once you have modified the settings for the Pilot run, you can start it on your local machine.

To start the Pilot run:

▶ Click **Start** at the bottom of the Pilot properties page.

On-Screen Events During a Pilot Run

After you start a Pilot run, the TestFactory window closes, a mask dims the screen and displays the *Running Pilot* message. The following figure shows the Pilot progress bar that opens at the bottom of the screen.
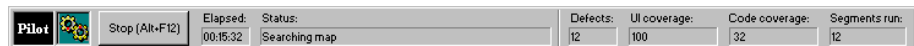
NOTE: After the run starts, do not try to use the keyboard or mouse unless you want to stop the Pilot run.

During the Pilot run, the Pilot progress bar displays the following information:

- ▶ A busy indicator in motion shows that the Pilot is running.

- ▶ The **Elapsed** box displays the duration of the Pilot run so far.

- ▶ The **Status** box displays the Pilot activity underway.

- ▶ The **Defects** box displays the number of defects detected in the AUT so far.

- ▶ The **UI coverage** box displays the percentage of unique UI objects available to the Pilot that the best script has touched.

- ▶ The **Code coverage** box displays the code coverage value for the best script.

- ▶ The **Segments run** box displays the number of script segments created and run so far.

During a Pilot run, TestFactory builds a navigational map for the Pilot, plots a path through the AUT, and creates and runs script segments. TestFactory then generates the best script, calculates code coverage, and organizes the UI script, defect scripts (if any), and UAW script (if an unexpected active window was activated) that it generated.

Stopping a Pilot Run

To stop the Pilot run at any time, click **Stop** on the Pilot progress bar or press ALT+ F12. TestFactory prompts you to indicate whether you want to save an optimized best script. To save a best script, click **Yes**. Bear in mind that if you stopped the Pilot run immediately after you started it, TestFactory will probably not have generated enough data to produce an optimized best script.

> **NOTE:** To specify a shortcut key combination other than the default ALT+ F12 to stop a Pilot run, click **Tools > Options**, and then, on the **General** tab, select a different key combination from the **Stop shortcut key** list.

After a completed Pilot run, the **Summary** tab displays the following information in the restored TestFactory window:

UI coverage Percentage of unique UI objects available to the Pilot that the best script touched.

Code coverage Percentage of all source code in the AUT that the best script touched.

Defects found Total number of defects detected.

Elapsed time Duration of the Pilot run.

Stopped due to The stop criterion (or the user) that ended the run.
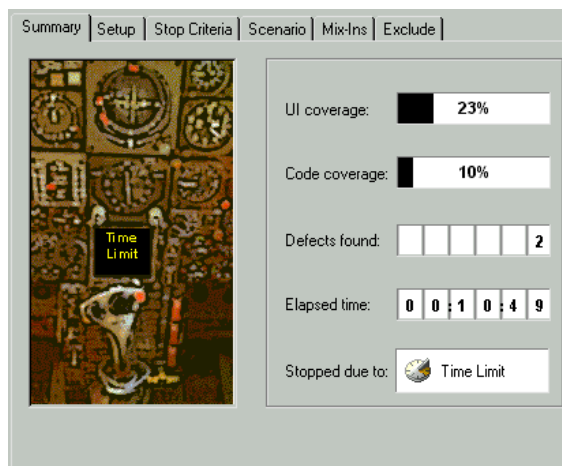
Summary tab

Examining Pilot Run Results

The most important Pilot run results are the scripts that your Pilot generates. This section describes the kinds of scripts a Pilot can generate and how to view all of the information that they provide.

Pilot Run Folder Contents

After a Pilot run is completed, TestFactory inserts a *<Pilot name>*-Run-*<date time>* folder under the Pilot object in the application map.
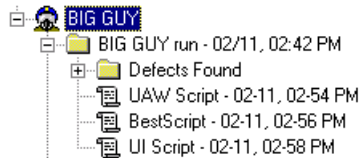
The *<Pilot name>*-Run-*<date time>* folder can include the following items:

▶ A single best script.

▶ If you specified support scripts for the Pilot run, TestFactory places the support scripts and the best script in a Test Suite object.

▶ If the Pilot generated script segments that uncovered severe defects, the results include a Defects Found folder that contains one or more single-segment defect scripts.

▶ If an unexpected window opens during the Pilot run, the results also include a UAW (unexpected active window) script.

▶ A single UI script.

> **NOTE:** To keep detailed information about Pilots and scripts, you can take advantage of the User Properties dialog box. To open the User Properties dialog box, right-click a Pilot or script object, and then click **User Properties** on the shortcut menu.

Renaming Generated Scripts

If you prefer your own script-naming scheme to the one TestFactory uses, you can rename the scripts that your Pilots generate.

To rename a generated script in the application map.

1. Click the script object.

2. Press F2.

3. Type a name in the active text box, and then press ENTER.

Viewing the Script Outline

The script **Outline** tab displays the steps that a script performed in an easy-to-read format. To view the **Outline** tab in the right pane, click the script object in the application map.
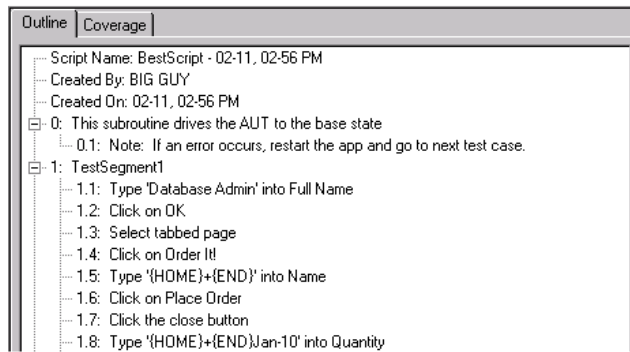
To print the outline, click **File > Print**.

NOTE: You can copy the outline text for a defect script directly to a defect report in ClearQuest.

Viewing Coverage Results for a Script

The **Coverage** tab displays the **UI Coverage** and **Code Coverage** values for the script. TestFactory calculates the UI coverage percent value (for all generated scripts) based on the number of unique UI objects that the Pilot has access to from its location in the application map. TestFactory calculates the code coverage percent value (for the best script and the UI script) based on the source code that the script touched relative to *all* source code in the AUT.

To view coverage results for a script:

1. Click the script object in the application map, and then click the **Coverage** tab in the right pane.
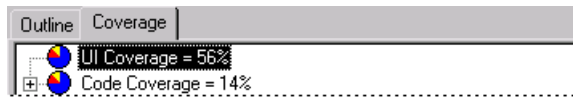
2. To view the coverage values for every source file in the AUT, expand the **Code Coverage** item.
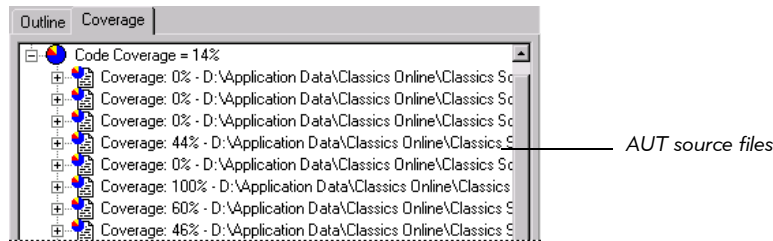
AUT source files

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

Source file procedures
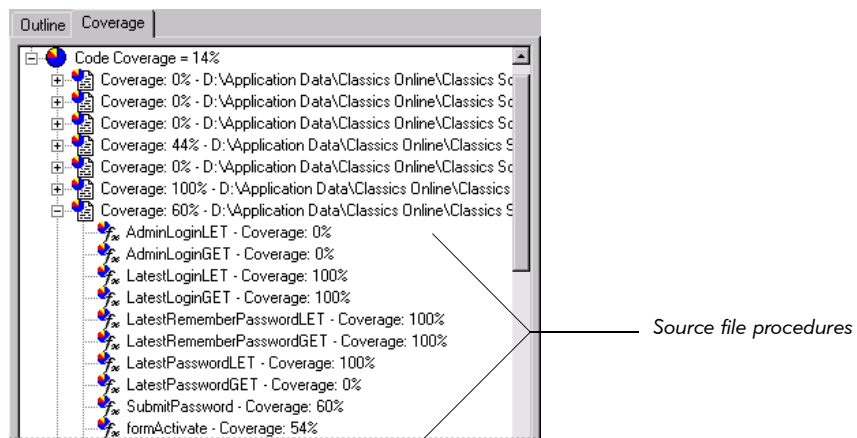
Viewing Code Coverage Information for Scripts Generated for Java Applications and Applets

The code coverage information calculated for scripts generated for Java applications and applets differs from code coverage information for scripts generated for C++ and Visual Basic applications. If the AUT is written in Java, TestFactory builds the coverage dictionary during the testing process rather than during instrumentation. Java .class files reveal themselves to TestFactory only after a script touches the code contained in the .class file.

If the AUT is a C++ or Visual Basic application, the code coverage value for a best script represents the percentage of *all* source code in the AUT that the script touched. TestFactory "knows" the total amount of source code in the AUT before you run a Pilot. If the AUT is a Java application or applet, TestFactory finds out about the total amount of source code incrementally, through testing. As a result, the code coverage values for best scripts run against an AUT written inJava are calculated relative to the .class files exposed so far. This means that code coverage values can be artificially high, especially for the first few Pilots you run. As you run more Pilots to test different functional areas of a Java AUT, the coverage dictionary becomes more complete, and code coverage values for scripts become more realistic.

Instrumenting a new build of a Java AUT deletes the current coverage dictionary. To rebuild the coverage dictionary, you must rerun the scripts that expose the Java .class files.
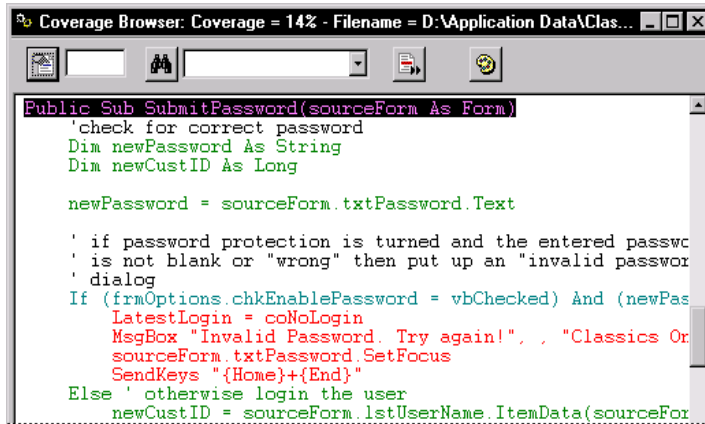
The Coverage Browser (C++ and Visual Basic Applications)

If the AUT is written in Visual Basic or C++ and you instrumented it before you ran a Pilot (or you have a coverage dictionary and access to the instrumented Visual Basic source files), then you can open the **Coverage Browser** and view the source code that a script exercised.

NOTE: TestFactory does not open Java .class files in the Coverage Browser.

To view source code coverage details:

1. To see the source code for a procedure in the Coverage Browser, double-click the procedure on the **Coverage** tab. The following figure illustrates instrumented source code displayed in the Coverage Browser.

2. To jump to a specified line of code in the Coverage Browser, type the line number in the first text box, and then click the **Go To Line** button.

3. To jump to the first instance of a text string, type the text string in the second text box, and then click the **Find Text** button.

   NOTE: TestFactory does not support partial word matches. Be sure to type an entire word or text string in the text box.

4. To jump to the next line of source code that was *not* covered, click the **Next Not Covered** button.

   You can also scroll through the source code file.

5. The Coverage Browser displays color-coded coverage results. To view the current color settings for displayed text, click the **Coverage Text Color** button.

Changing the Appearance of Text In the Coverage Browser

To change the appearance of text in the Coverage Browser:

1. Click **Tools > Options**, and then click the **Coverage Browser** tab.
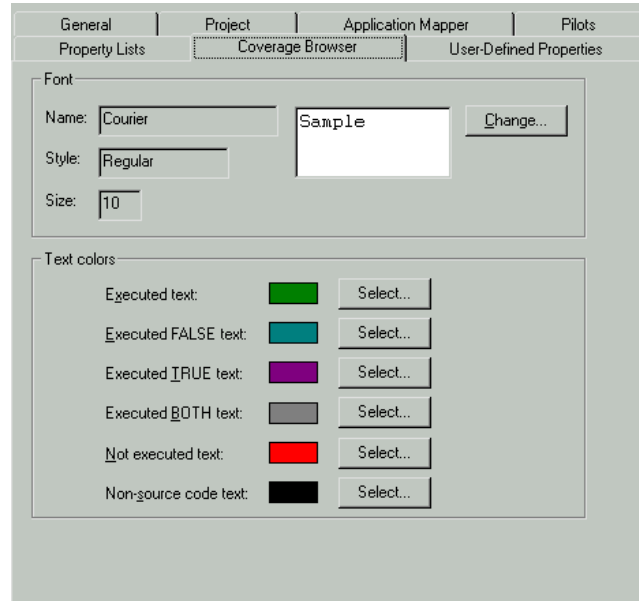
2. To change the font used to display text, under **Font**, click **Change**, and then use the Font dialog box to change the text font.

3. To change the color used to display a source code segment, under **Text color**, click **Select** adjacent to the source code segment color, and then use the color palette to change the setting.

4. Click **OK**.

Viewing the Log for a Defect Script

After a Pilot run, you can view the log for a defect script in the Test Log window of Rational TestManager.

To view the log for a defect script:

▶ Click the defect script object in the application map, and then do one of the following:

   – Click **Script > View Log**.

   – Click **Tools > Rational Test > Rational TestManager**, and then click **File > Open Test Log**.

Alternatively, right-click the script object, and then click **View Log** on the shortcut menu.
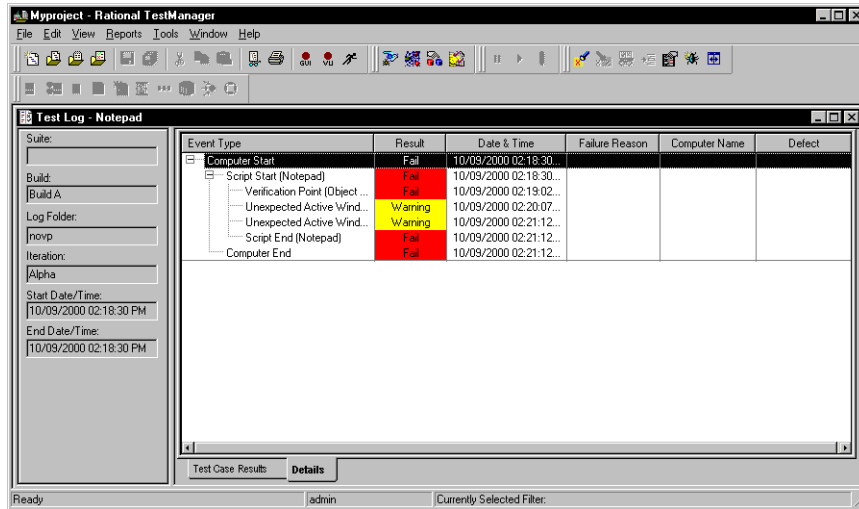
Reporting a Defect

TestFactory uses ClearQuest as its defect tracking system. Before you can use ClearQuest from TestFactory, you must first set up the database for the AUT in ClearQuest. For information about specifying the AUT database, see the *Getting Started with Rational Robot* manual.

From TestFactory, you can link to ClearQuest to report new defects. To start ClearQuest and report a new defect, do one of the following:

▶ Click **Tools > Rational ClearQuest**.

▶ On the Tools toolbar, click **Start ClearQuest**.

For detailed information about reporting a defect in ClearQuest, see ClearQuest Help. For a complete description of methods for tracking script defects, see the *Using Rational TestManager* manual.

> **NOTE:** You can click a defect script and copy the steps on the **Outline** tab to the description section of your defect form in ClearQuest.

Viewing a UAW Script

During a Pilot run, if a window that TestFactory expects to see does not open, or if an unmapped window such as an error message box opens instead, then TestFactory retains the running script segment as a UAW (unexpected active window) script. You can use a UAW script to trace the steps a Pilot took in the AUT before losing its way, and then map the unmapped path.

If you are satisfied with the Pilot results, there is no need to review the UAW script. If you want to analyze a UAW script, click the script object, and then examine the steps displayed in the **Outline** tab in the right pane.

If the script steps do not provide the information you need, you can analyze the UAW script further in Robot. To examine a UAW script in Robot, you must open it in Robot, set the Robot GUI playback options, play back the script, and then view the log for the script in the Test Log window of TestManager .

To open a UAW script in Robot:

▶   In the application map, right-click the UAW script object, and then click **Open** on the shortcut menu.

To set the GUI playback options and play back the script in Robot:

1.   Click **Playback Script** on the Robot toolbar.
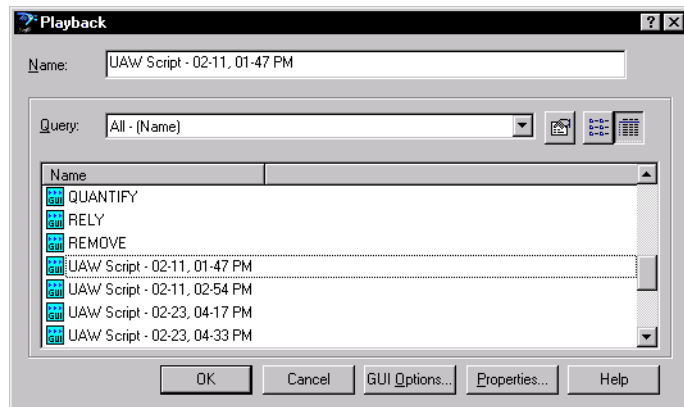
2.   Click **GUI Options**.

3. Click the **Log** tab.
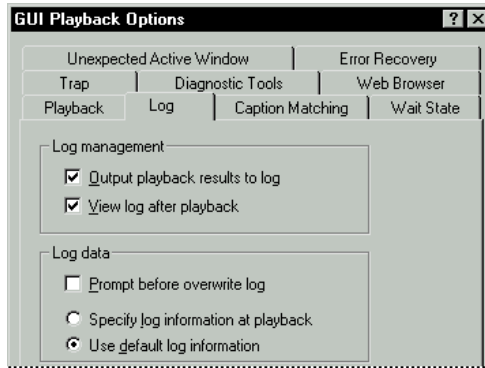
4. Under **Log management**, select the **Output playback results to log** and the **View log after playback** check boxes.

5. Under **Log data**, click **Use default log information**.

6. Click the **Unexpected Active Window** tab.
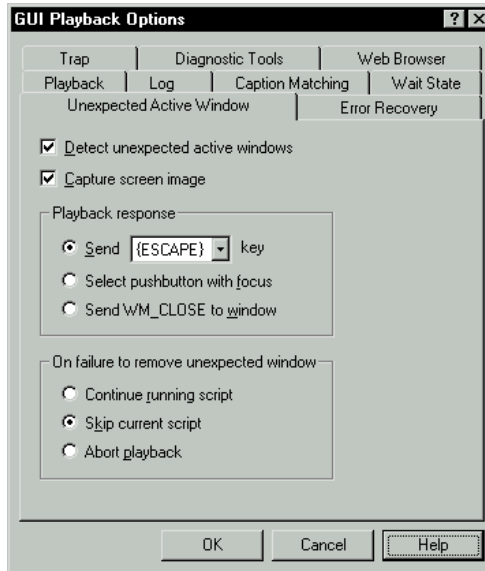
7. Select the **Detect unexpected active windows** and the **Capture screen image** check boxes.

8. Under **On failure to remove unexpected window**, click **Skip current script**.

> **NOTE:** After you finish examining the UAW script, be sure to select the **Continue running script** option. Leaving **Skip current script** selected interferes with Pilot and script runs in TestFactory.

9. To close the GUI Playback Options dialog box, click **OK**.

10. To play back the UAW script, click **OK**.

Robot plays back the script until the UAW opens. After the script playback ends, the Test Log window of TestManager displays the log for the UAW script.
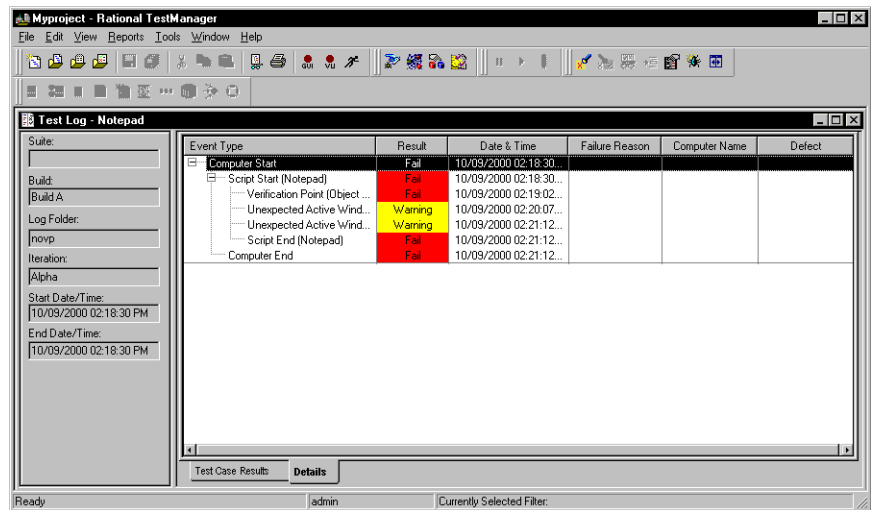
To open the Image Comparator and view a screen capture that includes the UAW:

▶   Double-click the Unexpected Active Window item in the **Event Type** column.

To view the properties of the unexpected active window:

1. Click **Tools > Rational Test > Rational TestManager**, and then click **File > Open Test Log**.

2. Click the **Details** tab.

3. Right-click the **Unexpected Active Window** item in the **Event Type** column, and then click **Properties** on the shortcut menu.

4. To view the script line number associated with the UAW, click the **General** tab.

To view additional properties of the unexpected active window, click the **Configuration** tab.
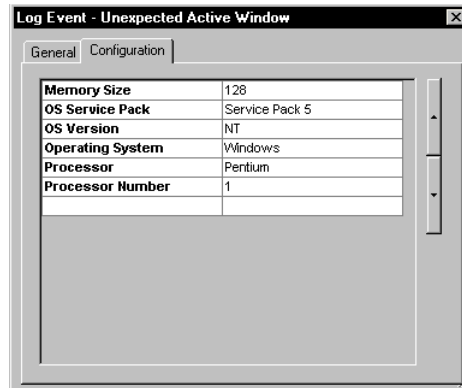
5. Close the Log Event dialog box, and then quit TestManager.

If you have finished viewing UAW scripts for now, do the following before you start to work in TestFactory again:

1. In Robot, click **Tools > GUI Playback Options**, and then click the **Unexpected Active Windows** tab.

2. Under **On failure to remove unexpected window**, click **Continue running script**.

3. Click **OK**.

If your Pilots frequently generate UAW scripts, consider doing the following:

▶ Set up interaction objects to map unmapped paths in the area of the AUT you are testing.

▶ Check to see if the UAW script is generated as a result of a timing problem. If it is, set a delay interval for the appropriate action object in the application map. For information about setting delay intervals for Pilot runs, see *Specifying a Delay Interval to Include in Generated Scripts* on page 5-49.

Using Pilot Scenarios to Simulate User Action Sequences

After the AUT stabilizes, you can use Pilots to test interactions involving multiple functional areas of the AUT. Among other things, you can test how the AUT responds when functional areas are exercised in a sequence that a user is likely to follow. A Pilot **scenario** lets you simulate and test a user action sequence in the AUT.

To create a scenario, you combine Pilots inserted at different functional regions of the application map and arrange them in an ordered sequence. Each Pilot in the sequence contributes a few steps to each script segment that the lead scenario Pilot generates. The lead Pilot then builds the best script from the pool of script segments, each of which contains steps in the sequence that you specify.

Creating a Pilot Scenario

To create a Pilot scenario:

1. Determine the first functional area of the AUT that you want the scenario to test, and insert a Pilot at the corresponding location in the application map. This is the lead Pilot in the scenario.

   NOTE: If a Pilot already exists at a map location where you want to begin a scenario, insert a new lead Pilot anyway. Segregating Pilots helps to limit confusion regarding the function of each.

2. Name the lead Pilot object in the application map, and then change settings on the **Setup**, **Stop Criteria**, and **Exclude** tabs on the Pilot properties page.

3. Insert additional Pilots at application map locations corresponding to the other functional areas that you want to add to the test sequence. To ensure that TestFactory calculates accurate code coverage values for generated scripts, do not combine Pilots that have access to any of the same controls.

   NOTE: To avoid making the scenario too complicated, we recommend that you include non-lead Pilots that are fairly simple. Avoid adding Pilots that contain their own scenarios.

4. Adjust the settings on the **Setup** and **Exclude** tabs for each non-lead Pilot.

> **NOTE:** In a scenario run, TestFactory applies the stop criteria specified for the lead Pilot and ignores the stop criteria specified for non-lead Pilots.

5. Click the lead Pilot, and then click the **Scenario** tab.
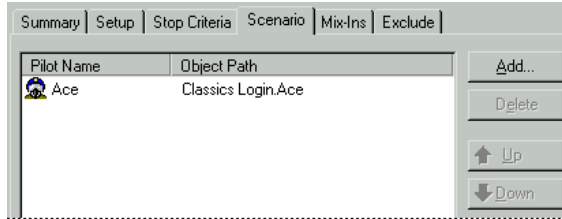
6. Click **Add**.
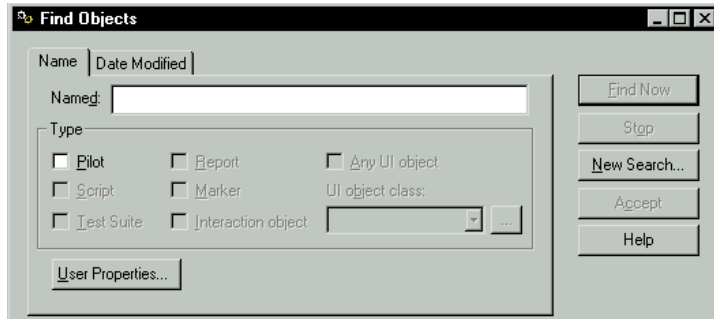
7. To locate Pilots to add to the scenario, select the **Pilot** check box, and then click **Find Now**.
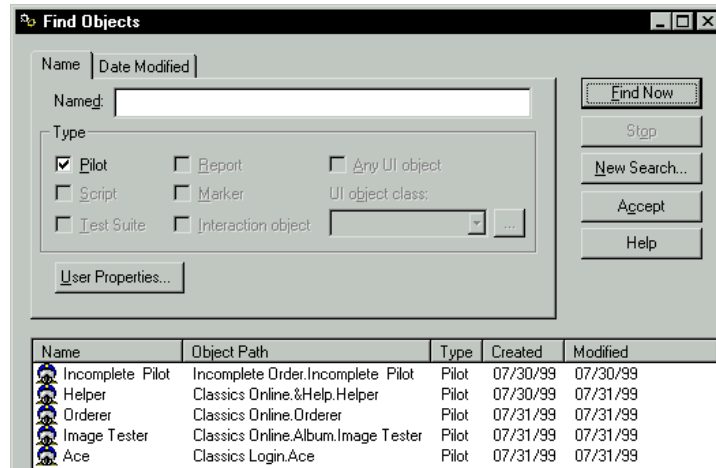
8. To add all of the Pilots found, click **Accept**. To add a subset of the Pilots found, press CTRL or SHIFT, select the Pilots to add, and then click **Accept**.

NOTE: To achieve results that are easy to track and analyze, we recommend that you include no more than five to eight Pilots in a scenario.

9. To specify the order of steps in the script segments that the lead Pilot generates, select each Pilot listed, and then use the **Up** and **Down** buttons.

10. To run the Pilot scenario on your local machine, click the lead Pilot in the application map, and then click **Start** at the bottom of the Pilot properties page.

Using Pilot Mix-Ins to Test Random Interactions

A Pilot mix-in is useful for testing how functional areas of the AUT interact when you combine them randomly. For example, if the AUT is a word processing application, and you want to test how the AUT responds when the spell checker is used occasionally during an editing session, you can insert Pilots that access the functional areas that you want to test, and combine them in a mix-in.

You can also add a mix-in Pilot to a scenario. As part of a scenario, a mix-in is a useful way to include a step that a user might occasionally perform in the course of an otherwise predictable action sequence. For instance, if you created a scenario Pilot for the customer tracking AUT, you could mix in a Pilot for a Delete Customer Record dialog box that a user would only open infrequently.

The **Mix-Ins** tab is similar to the **Scenario** tab in that you can use it to combine Pilots from different functional regions in the application map. Each mix-in Pilot contributes a few steps to a specified percentage of the script segments generated.

A mix-in includes the **Setup** tab and **Exclude** tab settings of the Pilots that you add to it. For example, if you specify a control to exclude from a mix-in Pilot, that exclusion remains in effect for the steps that the Pilot contributes during the run.

NOTE: Although you can add any number of mix-in Pilots to a scenario, we recommend that you include just one or two.

Creating a Pilot Mix-In

To create a mix-in Pilot to run with a Pilot scenario:

1. Insert a Pilot at the branch of the application map that corresponds to the functional area that you want to randomly mix in to the scenario.

2. Name the mix-in Pilot.

3. Specify settings for the mix-in Pilot on the **Setup** and **Exclude** tabs.

4. Click the lead Pilot for the existing scenario, and then click the **Mix-Ins** tab.
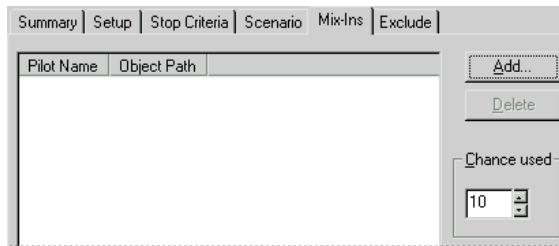
5. Click **Add**.
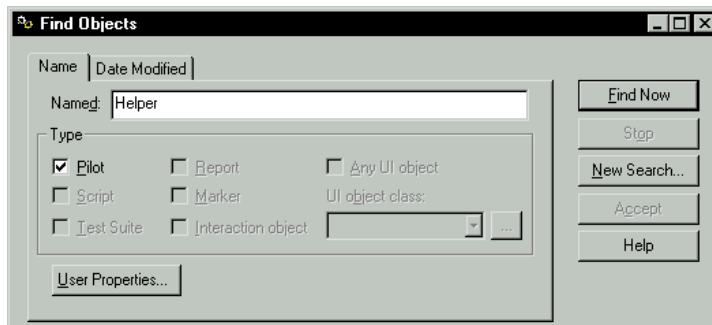
6. Type the name of the mix-in Pilot in the **Named** box.

7. Select the **Pilot** check box.

8. Click **Find Now**.
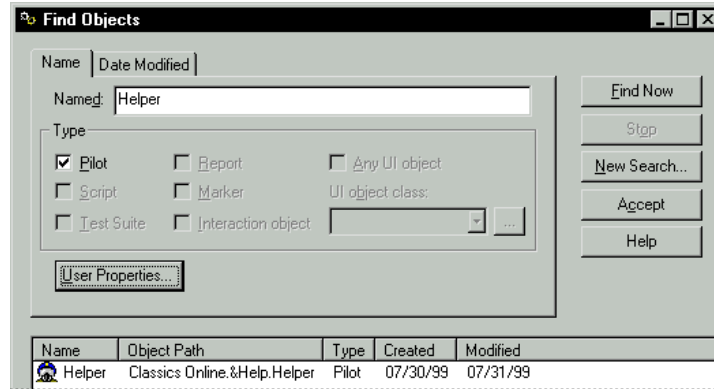
9. Click **Accept**.

10. To specify the probability with which TestFactory mixes in steps from the mix-in Pilot, enter a percent value in the **Chance used** box. For example, to have TestFactory exercise the functional area for the mix-in Pilot approximately once in every 20 action sequences, change the value in the **Chance used** box to 5.

11. To run the scenario Pilot that now includes a mix-in Pilot on your local machine, click the lead Pilot, and then click **Start** on the Pilot properties page.

Additional Adjustments for Pilot Runs

You can improve a Pilot run by doing one or more of the following:

▶ Specify values for **StringCases** and **MaskCases** properties of UI objects and interaction object components that represent input controls (such as text boxes and combo boxes) that require user input of a particular type or in a particular format.

▶ Before you rerun a Pilot, change the route number on the **Setup** tab to refresh the Pilot. Changing the route number ensures that the Pilot takes a new path through the AUT and increases the chances that the next run uncovers new defects.

▶ TestFactory does not support the automatic testing of some types of controls. For example, TestFactory does not exercise individual cells contained in grid controls and calendar controls during mapping and testing. TestFactory can, however, test these cells if you create region objects for them in the application map.

Changing Default Settings for Pilots

A Pilot gets some of its default settings from the **Pilots** tab in the **Options** dialog box. You can change these settings globally from within the **Pilots** tab.

To change the default settings for new Pilots:

1. Click **Tools > Options**.

2. Click the **Pilots** tab.
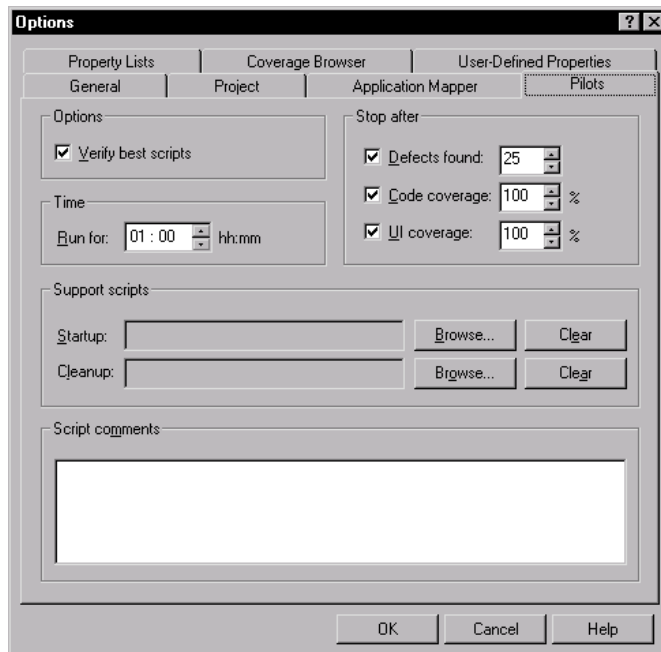
3. Except for the **Verify best scripts** check box, the options on the **Pilots** tab correspond to options on the Pilot **Setup** and **Stop Criteria** tabs. Change these options as you would change them on the other tabs.

4. To make these settings your defaults for new Pilots, click **OK**.

All new Pilots that you insert and run use the new default settings. If you want to run an existing Pilot using the new default settings, click the Pilot object in the application map, and then click **Reset** at the bottom of the Pilot properties page. The next time you run the Pilot, TestFactory applies the new default settings.

Opening and Editing a Best Script in Robot

If you want to insert a verification point in a best script, or if you want to record new actions in it, you can open the script and edit it in Robot.

Opening a Best Script in Robot from TestFactory

To open a best script in Robot, do one of the following:

▶ Right-click the script object in the application map, and then click **Open** on the shortcut menu.

▶ Click the script object in the application map, and then click **Script > Open**.

▶ Click the script object in the application map, and then click **Tools**, point to **Rational Test**, and then click **Rational Robot**.

▶ Click the script object, and then click **Start Robot** on the Tools toolbar.

For information about inserting verification points and recording user actions, see Robot Help or the *Using Rational Robot* manual.

Obtaining Code Coverage for Robot Scripts

If a project contains Robot scripts, you can run these scripts from TestFactory to get code coverage values for them. To obtain code coverage values for a Robot script, the script must satisfy the following requirements:

▶ The Robot script must start the AUT.

▶ The script code used to start the AUT must be recorded using the Robot Insert menu or the GUI Insert toolbar, and not using the Start → Programs menu or a shortcut on the desktop. For information about starting the AUT using the GUI Insert toolbar or the Insert menu, see the topic *Starting an Application in a GUI Script* in Robot Help.

▶ The Robot script must quit the AUT. The script can use any method that the AUT provides to quit the application (for example, the Close button on the title bar, or the **Exit** command on the **File** menu).

If you used the object code method to instrument the AUT, and you want to run a Robot script from TestFactory to get code coverage information for the script, you must first replace the StartApplication statement in the script with the SQAShellExecute statement.

To replace the StartApplication statement in a Robot script with the
SQAShellExecute statement:

1. Start Robot and open the script that you want to run from TestFactory.

```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 8/12/99  3:06:41 PM
    'Script Name: ClassicsC Order
    StartApplication """D:\Sample Applications\Classics Online\ClassicsC.exe"""

    Window SetContext, "Name=frmExistingLogin", ""
    InputKeys "D"
    PushButton Click, "Name=cmdOK"
```

*StartApplication statement
in a recorded Robot script*

2. Find and delete the "StartApplication" statement near the beginning of the
    script.

3. On the line that contained the StartApplication statement, type the following
    SQAShellExecute statement:

```
SQAShellExecute "LaunchAUT.exe", _
"<path to folder where TestFactory is installed>", _
"<project name>|:ExeName:<path to AUT>"
```

```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 8/12/99  3:06:41 PM
    'Script Name: ClassicsC Order
    SQAShellExecute "LaunchAUT.exe", _
    "C:\Program Files\Rational\Rational Test 7", _
    "CLASSICS|:ExeName:D:\Sample Applications\Classics Online\ClassicsC.exe"

    Window SetContext, "Name=frmExistingLogin", ""
```

SQAShellExecute statement

> **NOTE:** The underscore character at the end of the first two lines is a line
> continuation character that makes the statement more readable. If you want
> to keep the statement all on one line, delete the underscore characters.

To get code coverage information for a Robot script, run it from TestFactory.

> **NOTE:** The SQAShellExecute statement starts the AUT when you play back the
> script again from Robot. It is not necessary to restore the StartApplication
> statement after you run the script from TestFactory.

Creating a Custom TestFactory Script

The TestFactory script object provides another way to create a script. You can use the script object to record a TestFactory script in Robot and keep it with the scripts that you created automatically in TestFactory. This customizable script is a template, written in SQABasic™, that contains essential header files, the basic steps that TestFactory displays on the **Outline** tab for the script, and comments that tell you where to insert script code.

If you instrumented the AUT, you can run the custom script from TestFactory to get a code coverage value for it. TestFactory displays the code coverage value on the **Coverage** tab after the script is run. You can view and run all of your Pilot-generated scripts in Robot.

Creating and Opening a Custom TestFactory Script

To create a custom TestFactory script and open it in Robot:

1. Click a folder or other object in the application map as a destination for the script object, and then click **Insert > Script**.

   Alternatively, drag a script object from the Insert toolbar to a destination in the application map.

2. Rename the script object, and then press ENTER.

3. To open the script in Robot, do one of the following:

   – Click **Script > Open**.

   – Click **Tools**, point to **Rational Test**, and then click **Rational Robot**.

   – Click **Start Robot** on the Tools toolbar.

   – Right-click the script object, and then click **Open** on the shortcut menu.

Recording Actions in a Custom TestFactory Script

Recording a custom TestFactory script in Robot is a two-part process. In the first part, you record the actions TestFactory performs to drive the AUT to the base state for testing. In the second part, you perform and record the actions that you want to include in the test case.

To record as TestFactory drives the AUT to the base state for the script:

1. The custom TestFactory script contains code that starts and quits the AUT, and comments that tell you where to place your code. In the shell script, click the cursor after the comment `'Step: <Add code here to drive the AUT to base state>`.

2. Click **Record > Insert At Cursor**.

3. After Robot minimizes its window, click **Pause Recording** on the GUI Record toolbar.

GUI Record toolbar
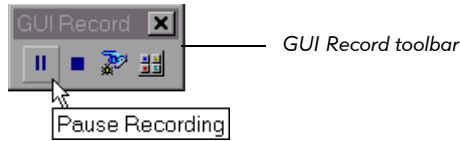
4. To bring the AUT to the appropriate state for recording new actions, right-click the UI object in the application map that you want to represent the base state for the script, and then click **Go To "*Control*" in AUT** on the shortcut menu.

5. Immediately after the TestFactory window closes, click **Pause Recording** to begin recording as TestFactory drives toward the selected control in the AUT.

6. Wait until the TestFactory window is restored, and then click **Pause Recording** on the GUI Record toolbar.

7. Minimize the TestFactory window, and then click **Stop Recording** on the GUI Record toolbar.

To record a test case:

1. In the shell script, click the cursor after the comment `'Step: <Add your test case code here>`.

2. Click **Record > Insert At Cursor**.

3. Perform the actions in the AUT that you want to record as the test case.

4. Quit the AUT.

5. On the GUI Record toolbar, click **Stop Recording**.

> To display additional steps for the custom script on the **Outline** tab, insert them in the script manually as comments, using the format in the following example:
>
> ```
> 'Step: 1: This step starts the AUT.
> 'Step: 2: Type 'quick brown log' into Sample text.
> 'Step: 3: Turn on italic.
> ```

6. Save the script in Robot.

You can play back the script immediately in Robot, or you can quit Robot. To obtain code coverage for the script, right-click the script object in the application map, and then click **Play Back** on the shortcut menu.

NOTE: If you prefer not to use the **Go To** *"Control"* feature to record the actions that drive the AUT to the base state for testing, you can record all of the actions as you perform them manually in Robot.

Checking for Memory Errors in Visual Basic and C++ Applications (Windows NT)

Late in the development cycle, you can run Pilots and scripts to check for memory-related defects in an AUT written in C++, Visual Basic 5, or Visual Basic 6. To test for memory errors, you must be running TestFactory in Windows NT.

NOTE: If the AUT is written in Visual Basic 4, you cannot run a Pilot to check for memory errors.

TestFactory can check for the following kinds of memory errors:

▶ Memory and resource leaks

▶ Invalid memory access

▶ Memory overwrites

▶ Uninitialized memory reads

▶ Memory access beyond the bounds of an array

TestFactory detects these memory-related defects in addition to the other types of defects it detects during a normal Pilot run.

Preparing to Test for Memory Errors in the AUT

To prepare to test for memory errors in the AUT:
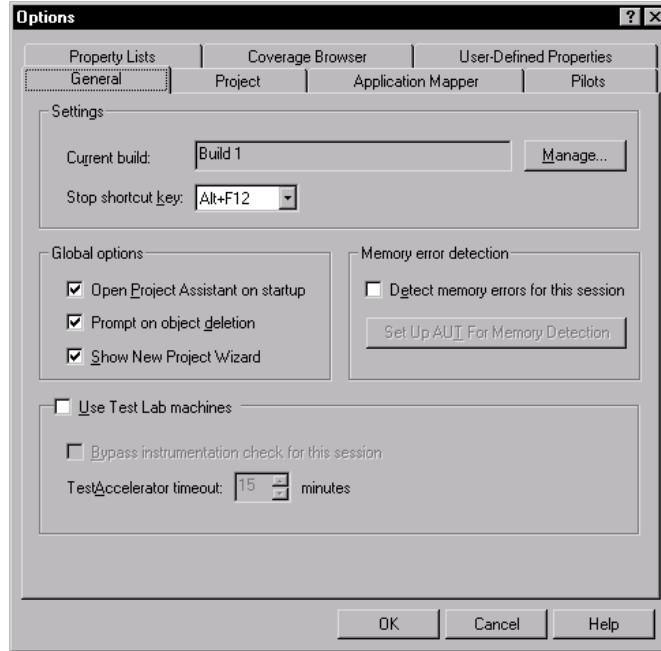
1. Click **Tools > Options**.

2. Under **Memory error detection** on the **General** tab, select the **Detect memory errors for this session** check box.

   Before you can test for memory errors, you must set up the AUT for memory error detection. Because this can take several minutes, we recommend that you set up the AUT before you start a Pilot or script run. If you do not, TestFactory sets up the AUT after the run starts. This can potentially cause timing problems.

Click here to set up the AUT for memory testing.
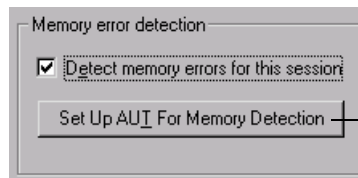
3. To set up the AUT now, click **Set Up AUT For Memory Detection**.

4. After the AUT is set up, click **OK**.

After you set up the AUT, memory checking remains in effect for all Pilots and scripts until you clear the **Detect memory errors for this session** check box, change projects and then reopen the original project, or quit TestFactory.

To make memory checking unavailable:

1. Click **Tools > Options**.

2. Under **Memory error detection** on the **General** tab, clear the **Detect memory errors for this session** check box.

3. Click **OK**.

Running a Pilot to Check for Memory Errors

You can run an existing Pilot, or insert a new one to check for memory errors. The steps for setting up a memory-checking Pilot run are the same as those for other Pilot runs. A memory-checking Pilot differs from other Pilot runs in that it can only be run on your local machine, and TestFactory does not calculate code coverage values for the scripts that are generated. (TestFactory ignores the value set for **Code coverage** on the **Stop Criteria** tab.)

A Pilot that you run to check memory errors runs much more slowly and uses significantly more memory than it would otherwise. If you start a Pilot run and you see a system error message indicating that the machine has insufficient resources to run the Pilot, consider increasing your system resources before you test for memory errors.

Checking the Timing of a Pilot Run

If you start a Pilot run without first setting up the AUT for memory checking, do the following:

▶ After you start the run, monitor the Pilot progress bar until you see **running scripts** displayed in the **Status** box. If the AUT starts within five minutes, you can stop monitoring the Pilot run. If the AUT has *not* started after five minutes, do the following:

   a. To stop the Pilot run, click **Stop** on the Pilot progress bar.

   b. After the TestFactory window is restored, click **Tools > Options**.

   c. Under **Memory error detection** on the **General** tab, click **Set Up AUT For Memory Detection**.

   d. After the AUT is set up, click **OK**. (Setup can take several minutes.)

      e.   In the application map, click the Pilot.

      f.   Restart the Pilot run.

Setting up the AUT before you restart the Pilot run should solve timing problems during the run.

Viewing the Results of a Pilot Run to Detect Memory Errors

The Best Script

If you run a Pilot to check for memory errors, TestFactory does not calculate code coverage for the best script. Instead, it optimizes the best script based on UI coverage. To get a code coverage value for a best script that was generated during memory checking, clear the **Detect memory errors for this session** check box on the **General** tab, and then run the script from TestFactory.

Defect Scripts

The Defect Found folder for a memory-checking Pilot can include defect scripts that uncover memory-related defects, as well as defect scripts that uncover other kinds of defects (AUT crashes, Visual Basic run-time errors, and assertions). To determine whether a defect script is associated with a memory error or other kind of defect, you must view its log in the Test Log window of Rational TestManager. For information about viewing a script log, see *Viewing the Log for a Defect Script* on page 5-17. For information about the kinds of memory errors uncovered by defect scripts, see the topic *Using Purify Messages: Message Types* in Rational Purify Help.

If you run a Pilot in an area of the AUT that contains an error unrelated to memory, the Pilot generates a single defect script for that defect, regardless of how many times the defect turns up during the run. However, if you run a Pilot in an area of the AUT that contains a memory error, the Pilot generates a new defect script each time the error shows up during the run. If this happens, the Pilot results include a Defects Found folder that contains multiple defect scripts for a single memory error. Although there is no way to completely avoid generating redundant memory defect scripts, you can reduce the number of these that a Pilot generates.

If a Pilot generates many defect scripts for a single memory error, do the following:

1.   To modify the Pilot stop criteria so that a Pilot run generates fewer defect scripts (for any kind of defect):

      a.   Click the Pilot in the application map.

      b.   Click the **Stop Criteria** tab, and then do one of the following:

      –   Enter a low target value (between 5 and 10) for the **Defects found** criterion.

      –   Enter a short run time (such as 00:10) in the **hh:mm** box for the **Run for** criterion.

2. Rerun the Pilot.

3. If the Pilot run generated defect scripts, view their logs until you find one or two that are associated with memory errors.

4. Fix the memory errors in the AUT, and then rerun the Pilot.

If the Pilot encountered the same memory error repeatedly during the previous run, then tracking down and fixing the first occurrence of a memory error should reduce the number of redundant defect scripts generated.

If your memory-checking Pilots are generating large numbers of non-duplicate defect scripts for relatively small functional areas of the AUT, it could indicate that you started checking for memory errors too early in the development cycle. Consider waiting until the AUT is further developed before you check for memory errors.

Running Scripts to Check for Memory Errors

In addition to running Pilots that check for memory errors in the AUT, you can run existing scripts from TestFactory to check for memory errors.

To run scripts to test for memory errors in the AUT:

1. Click **Tools > Options**.

2. Under **Memory error detection** on the **General** tab, select the **Detect memory errors for this session** check box.

3. To set up the AUT now, click **Set Up AUT For Memory Detection**.

4. After the AUT is set up, click **OK**.

5. In the application map, right-click the script that you want to run for memory-checking, and then click **Play Back** on the shortcut menu.

You can also run multiple scripts in a Test Suite to check for memory errors. For information about running Test Suites, see Chapter 6, *Developing and Running a Test Suite*.

Testing Controls in the AUT During Pilot Runs

Before you run a Pilot to test a functional area of the AUT, you can modify the testing environment to ensure that your Pilots generate robust, stable scripts. This section provides information about how to improve test results by selecting and modifying the data entry styles used to test input controls, and by modifying the properties of UI objects and interaction object components.

TestFactory exercises controls in the AUT differently during mapping than it does during Pilot runs. The two major ways in which TestFactory behaves different during mapping and testing are as follows:

▶ The Application Mapper exercises controls in a systematic fashion, while a Pilot exercises controls in a more unpredictable fashion. For example, after mapping starts, the Application Mapper exercises any interaction objects it finds before it exercises UI objects located at the same level of the application map hierarchy. After a Pilot run starts, TestFactory randomly exercises the objects it encounters. It does not necessarily exercise an interaction object before it exercises UI objects located at the same level of the application map hierarchy.

▶ TestFactory uses a wider variety of data entry types to test an input control in the AUT than it uses to map it. For example, to exercise an input control during mapping, the Application Mapper can use a required string case that you have specified, or mouse actions such as double-left-click or right-click. To test the same control, a Pilot can use several different types of entry data, including required string cases, mask cases, as well as randomly-generated string cases, integer values, and floating point values.

Without some direction from you, a Pilot can spend much of its run time testing negative cases. To introduce more focus to your Pilots runs, you can assign data entry styles to and modify the properties of UI objects and UI object components in the application map. This section describes how to provide some direction to TestFactory behavior during testing so that your Pilots generate rich scripts that touch as much of the AUT source code and uncover as many defects as possible.

Selecting a Style and Modifying Data Entry Settings for UI Objects and UI Object Components

If you run a Pilot to test an area of the AUT that contains an input control, the Pilot uses a set of default entry data to exercise the control. The default entry data consists of randomly-generated integers, float values, and string values. You can view the default test data by clicking an input UI object in the application map and expanding the **Pilot** properties in the Properties view.

You can provide an input control in the AUT with entry data that is more valid than the default entry data by assigning a style to the UI object and components that represent the control in the application map. If you assigned a style to a UI object or interaction object component before mapping, you can always adjust the data entry settings for testing. If you did not assign a style to an input UI object or interaction object component before mapping, we recommend that you do so before testing.

> **NOTE:** If you choose not to assign a style to an input-type UI object or component, you can specify entry data by modifying the UI object properties of UI objects in the Properties view and modifying the component properties in the Property List dialog box. For information about modifying properties, see *Modifying Properties to Control TestFactory Actions During Pilot Runs* on page 5-46.

This section describes how to:

▶ Assign a style to a UI object or to a UI object component that represents an input-type control in the AUT.

▶ Specify a required string case for the style.

▶ Modify the data entry settings for a style.

▶ Create a custom style.

To assign a data entry style to an input-type UI object and view its data entry settings:

1. In the application map, click the UI object.

2. On the Style toolbar at the top of the Properties view, click the **Select Style** arrow, and then click a style name.

Style toolbar

Select Style arrow

Existing data entry styles.
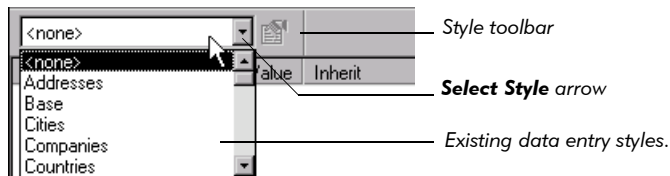
3. To open the Edit Data Entry Style dialog box and view the settings for the selected style, do one of the following:

– Click **Style Properties** on the Style toolbar.

– Right-click the UI object, and then click **Edit Style** on the shortcut menu.

After you add an input UI object component to an interaction object, TestFactory assigns the component the *Base* style.

To select a different style for a UI object component and view its data entry settings:

1. In the Interaction Object view, click the component name.

2. To open the Edit Data Entry Style dialog box, click **Style Properties** on the Interaction Object toolbar.

3. Under **Style**, click the **Name** box, and then click a different style name.

The following figure shows the Edit Data Entry Style dialog box. The histogram under **Test characteristics** shows the data entry composition that TestFactory uses during testing.
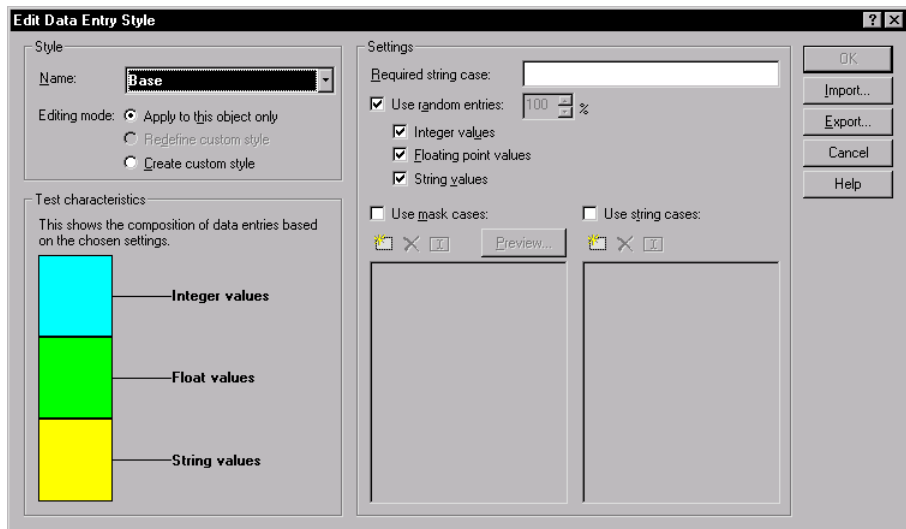
Specifying a Required String Case for a Style

Although none of the existing styles in the **Name** list has a required string case, you can add one to a style by typing it in the **Required string case** box under **Settings**.

If you specified a required string case for a style you assigned to a UI object or component before mapping, the Application Mapper used only that string case to exercise the corresponding control. If you run a Pilot to test this control, the Pilot includes the required string case in the mix of entry data it uses to test the control. To test the control using only the required string case, you must modify the data entry settings for the style.

To limit the entry data to the specified string case during testing:

1. Under **Settings**, clear the **Use random entries**, **Use mask cases**, and **Use string cases** check boxes.

2. To save your style changes and close the Edit Data Entry Style dialog box, click **OK**.

Specifying a Mix of Random Entries for a Style

All of the data entry styles that TestFactory provides specify random data entry for testing controls. The random data used to test controls can include integer values, float values, and string values.

In the Base style that TestFactory assigns to input-type components in interaction objects, **Use random entries** is the only data entry setting selected. If you keep this style for a component, 100% of the data entry used to test the input control is randomly generated, with integer values, float values, and string values equally represented in the mix.

If you use a style other than the *Base* style, or you create a new style for a UI object or component, we recommend that you set the percent for **Use random entries** to a low value such as 3% as you start running Pilots to test the control. This approach allows TestFactory to generate a small number of negative test cases during a Pilot run, but limits the number of defects that script segments are likely to encounter. Later, you can increase the percent value to uncover more defects.

To modify the mix of random entries for the style, do one or more of the following:

▶ To set the percentage of data entry that is represented by randomly-generated integer values, float values, and string cases:

– Under **Settings**, enter a percent value in the **%** box next to **Use random entries**.

▶ To exclude a specific type of randomly-generated value from the mix of random entries, clear the corresponding **Integer values**, **Floating point values**, or **String values** check box.

▶ To remove random entries from the data entry style entirely, clear the **Use random entries** check box.

Specifying or Modifying Mask Cases for a Style

Styles such as *Credit Cards*, *Dates*, and *Dollars* include a list of **mask cases** that you can use to test controls that require input in a specific format. The following figure shows the mask cases specified for the *Credit Card* style.

Preview button

Mask cases for the Credit Card style
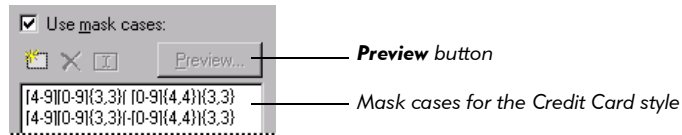
To view the sample data that TestFactory generates for a listed mask case:

▶ Under **Settings**, click a listed mask case, and then click **Preview**.

Sample data generated for the selected mask case
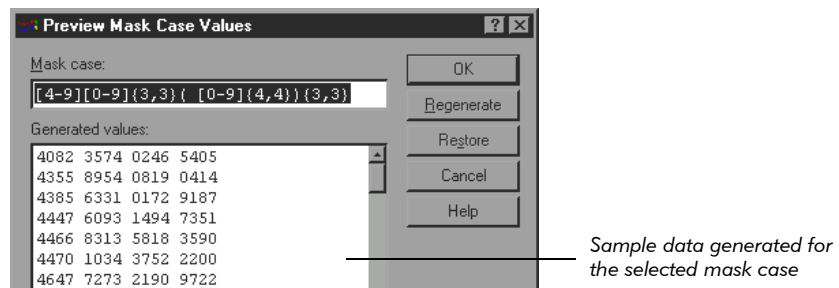
From the Preview Mask Case Values dialog box, you can edit a mask case and view the resulting sample data that TestFactory generates. For information about how to edit a mask case and view the sample data that TestFactory generates for it, click **Help**.

To save changes that you make to a mask case and close the Preview Mask Case Values dialog box, click **OK**.

To add a new mask case to the list:

▶ Click **Insert mask case**, and then type a mask case in the active box.

To edit a mask case in the list:

▶ Click the mask case you want to edit, click **Edit mask case**, and then make changes in the active box.

To remove a mask case from the selected style:

▶ Click the mask case, and then click **Delete**.

To remove all mask cases from the selected style:

▶ Press SHIFT, select all of the mask cases, and then click **Delete**.

To make mask cases unavailable for testing:

▶ Clear the **Use mask cases** check box.

Specifying or Modifying String Cases for a Style

Styles such as *Cities*, *Countries*, and *Web Sites* include a list of **string cases** for testing input controls. The following figure shows the string cases specified for the *Cities* style.

String cases for the Cities style
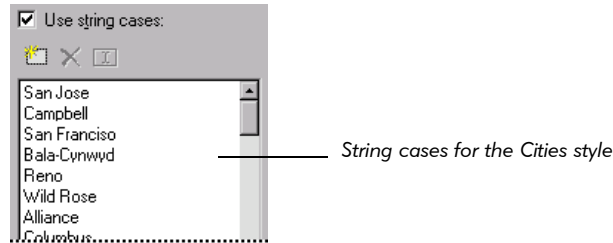
To add a new string case to the list:

▶ Click **Insert string case**, and then type a string case in the active text box.

To edit a string case in the list:

▶ Click the string case you want to edit, click **Edit string case**, and then make changes in the active box.

To remove a string case from the list:

▶ Click the string case that you want to remove, and then click **Delete**.

To remove all string cases from the selected style:

▶ Press SHIFT, select all of the string cases, and then click **Delete**.

To make string cases unavailable for testing:

▶ Clear the **Use string cases** check box.

Creating a Custom Data Entry Style

After you change the style settings, you can save the changes as overrides to the selected style, or you can create a custom style that has the modified settings.

To save the style settings as overrides of the selected style and close the Edit Data Entry Style dialog box, click **OK**.

To create a custom data entry style after you finish making changes to the data entry settings:

1. Under **Style**, click **Create custom style**.
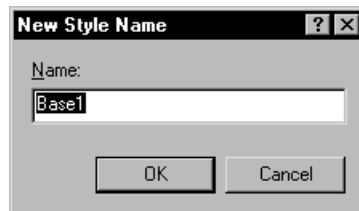
2. Type a name for the custom style in the **Name** box.

3. Click **OK**.

TestFactory adds the custom style name to the **Name** list in the Edit Data Entry Style dialog box, and to the **Select Style** list on the Style toolbar.

Managing Data Entry Styles

TestFactory provides a Manage Data Entry Styles dialog box that you can use to manage the data entry styles that you apply to input-type UI objects and interaction object components. You can use this dialog box to create new styles, edit or delete existing styles, and to import styles from or export styles to another project.

To open the Manage Data Entry Styles dialog box, do one of the following:

▶   Click **Tools > Manage Styles**.

▶   In the application map, right-click an input-type of UI object, and then click **Style** on the shortcut menu.
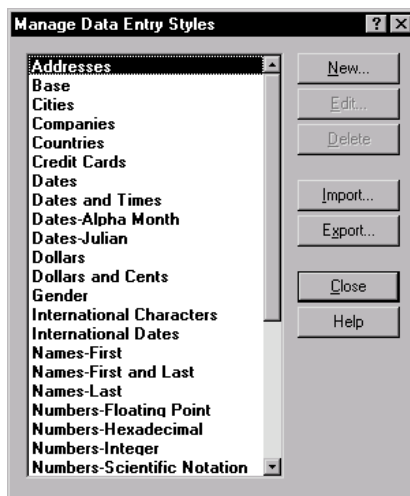
The Manage Data Entry Styles dialog box

For information about working with a listed style or creating a new style, click **Help**.

Modifying Properties to Control TestFactory Actions During Pilot Runs

You can control some aspects of TestFactory behavior during a Pilot run by adjusting the properties of action objects and the UI objects and components you are testing. This section describes how to change property values to improve Pilot run results.

To see the UI object properties of a UI object or action object, click the object in the application map. The Properties view displays the UI object properties in the top right pane. To open the Property List dialog box and see the properties of a component, double-click the component in the Interaction Object view.

Excluding a Control from All Testing

To exclude a control from all testing, you can modify the ExerciseDuringTesting property value of the UI object and components that represent the control.

To exclude a control from all testing:

1. In the application map, click the UI object mapped for the control that you do not want to test.

2. In the Properties view, click the **Value** field for **ExerciseDuringTesting**, and then click **Never**.

3. If the UI object mapped for the control that you want to exclude from testing has been added as a component to one or more interaction objects, do the following for every added component:

   a. Double-click the component.

   b. In the Property List dialog box, expand the **Pilot** properties group.

   c. Click the **Value** field for **ExerciseDuringTesting**, and then click **Never**.

If a project contains a UI script or best script that touches a control that you exclude from testing, and you run the script while the exclusion is in effect, the run will fail. We recommend that you do not run scripts such as these as long as the control is excluded from testing. If you plan to permanently exclude a control from testing, then delete or edit the scripts that exercise the control.

Setting the Interaction Order for Controls During Testing

NOTE: The following information about setting the interaction order for controls pertains to UI objects only. TestFactory exercises interaction object components based on their listed order in the Interaction Object view during both mapping and testing.

After you map the AUT the first time, you can specify the order in which TestFactory exercises UI objects during a Pilot run by changing the values of their InteractionOrder and ExerciseDuringTesting properties. During the mapping process, the Application Mapper exercises all of the controls in the AUT and follows the interaction order values that you have specified for objects in the application map. A Pilot, on the other hand, randomly exercises the controls to which you give it access. For example, if you insert the Pilot at a dialog box, TestFactory uses the mapped path to navigate directly through the user interface to the dialog box, which is the base state for the Pilot run. After reaching the base state, the Pilot exercises the controls in the dialog box randomly, touching some controls and ignoring others.

If you have set an interaction order for UI objects that a Pilot can access, and you want the Pilot to follow that specified order, then you must change the value of the ExerciseDuringTesting property for those objects to Always. This guarantees that those objects are tested in the correct sequence during the Pilot run.

To control the order in which TestFactory exercises the child objects of a UI object:

1. Click the UI object that you want TestFactory to exercise first.

> **NOTE:** If you specify the interaction order for objects for mapping the AUT, then you do not need to do it again before you test the objects. However, you must set the ExerciseDuringTesting property for the objects.

2. In the Properties view, click **Pilot**.

Name	Value	Inherit
⊞ **Application Mapper**		
⊟ **Pilot**		
ExerciseDuringTesting	Sometimes	Yes
DoAcceleratorSelection	No	Yes

3. Double-click the **Value** field for **ExerciseDuringTesting**, and then click **Always**.

4. In the Properties view, click **Shared**.

Name	Value	Inherit
⊞ **Application Mapper**		
⊞ **Pilot**		
⊟ **Shared**		
WaitInterval	1.00	Yes
RequiredStringCase		Yes
InteractionOrder	1000	No

5. Click the **Value** field for **InteractionOrder** and replace the default value of 1000 with a value greater than 0.

6. Click the next UI object for the interaction sequence.

7. In the Properties view, click the **Value** field for **ExerciseDuringTesting**, and then click **Always**.

8. Double-click the **Value** field for **InteractionOrder** and replace the default value of 1000 with a value that is greater than the value that you set for the first UI object.

9. Repeat steps 6 through 8 for the remaining UI objects in the interaction sequence.

During the Pilot run, TestFactory exercises UI objects in ascending order of the InteractionOrder property values.

Specifying a Delay Interval to Include in Generated Scripts

If a functional area of the AUT requires some lag time to respond to an action such as a left click, you can specify a delay interval to apply each time the action is used during testing. The delay interval value that you set forces Robot to wait the specified number of seconds before performing the next action during Pilot runs. TestFactory automatically adds code for the delay interval to the Pilot-generated scripts that include the action object.

To specify a delay interval for an action:

1. In the application map, click the action object for which you want to set a delay interval.

2. In the Properties view, expand the **Pilot** properties group.

Name	Value	Inherit
⊞ **Application Mapper**		
⊟ **Pilot**		
UseDuringTesting	Sometimes	Yes
DelayInterval	0.00	Yes
⊞ **Shared**		
⊞ **Object**		

DelayInterval property

3. Click the **Value** field for **DelayInterval**, and then type a value, in seconds, to set the duration of the delay.

When Robot encounters a delay value in a script, it waits for the specified amount of time before continuing playback. A delay value is useful if the action takes more time than that specified in the **Delay between commands** box in Robot. (For information about the **Delay between commands** box, see the topic *Setting GUI Playback Options* in Robot Help.

Because the delay interval is encoded in Pilot-generated scripts, and is independent of Pilot property settings in TestFactory, the delay is imposed whether you open and play back the script in TestFactory or in Robot.

Specifying a Wait Interval to Apply to an Object During Testing

If a functional area of the AUT requires some lag time to respond after a control is exercised, you can use the **WaitInterval** property to force TestFactory to wait a specified number of seconds after exercising the control before it continues testing.

To force a wait interval after a Pilot exercises a control in the AUT:

1. In the Properties view or Property list dialog box, expand the **Shared** properties group.

Name	Value	Inherit
⊞ **Application Mapper**		
⊞ **Pilot**		
⊟ **Shared**		
WaitInterval	1.00	Yes
RequiredStringCase		Yes
InteractionOrder	1000	No

Value field for the WaitInterval property

2. Click the **Value** field for **WaitInterval**, and then type a value, in seconds, for Pilots to wait after exercising the control before continuing to test.

After a Pilot exercises a control, TestFactory waits until the number of seconds specified for the **WaitInterval** property elapses, or until the **Timeout after** value set in Robot elapses—whichever is greater—before it continues. If you specify a WaitInterval value that is higher than the 30-second default **Timeout after** value in Robot, then TestFactory sets the Robot **Timeout after** value to the higher value when a script that exercises the object is played back. Otherwise, the default **Timeout after** value is used.

NOTE: Unlike the DelayInterval value, the WaitInterval value is not encoded in Pilot-generated scripts. If you start Robot and open and play back a script that includes UI objects for which you have specified a WaitInterval, Robot cannot apply the wait interval. For information about specifying a wait interval value for an object in the application map, see TestFactory Help.

Specifying a Required String Case

If you did not assign a data entry style to an input UI object or component, but you want to specify a required string case to pass to the corresponding input control during Pilot runs, you can do so by entering a value for the RequiredStringCase property.

To enter a value for the RequiredStringCase property:

1. In the Properties view or Property List dialog box, expand the **Shared** properties group.

Value field for the RequiredStringCase property

2. Click the **Value** field for **RequiredStringCase**, and then type a required string case in the active text box.

A Pilot that has access to the UI object or component includes the specified required string case in the mix of entry data it uses to exercise the control in the AUT. If you want Pilots to use only the required string case as entry data, you must modify the Pilot properties of the UI object or component.

To limit the entry data to the specified required string case:

1. In the Properties view or Property List dialog box, expand the **Pilot** properties group.

2. Change the values for the following Pilot properties to **No**:

    –   **UseStringCases**

    –   **UseMaskCases**

    –   **UseIntegerValues**

    –   **UseFloatValues**

    –   **UseStringValues**

Specifying the Entry Data Used to Test Input Controls

In addition to the RequiredStringCase property, UI objects and components that represent input controls have several Pilot properties that you can modify to specify the entry data used to test an input control. These Pilot properties parallel the data entry settings in the Edit Data Entry Style dialog box.

If you change the Pilot properties values for a UI object to which you have assigned a data entry style, the changes are applied as overrides of the selected style. If you open the Edit Data Entry Style dialog box after you change the Pilot properties in the Property List dialog box, you can see that the data entry settings reflect the changes you made to the Pilot properties.

If you change the values of Pilot properties for a UI object that has *no* data entry style, and you later assign a style to that UI object, the data entry settings of the assigned style override the changes that you made to the Pilot properties.

If you change data entry settings for a component by changing the values of Pilot properties in the Property List dialog box, the changes are applied as overrides of the selected style for the component. If you open the Edit Data Entry Style dialog box after you change the Pilot properties in the Property List dialog box, you can see that the data entry settings reflect the changes to the Pilot properties.

Specifying String Cases to Use During Testing

To specify string cases used to test an input control, you can use the StringCases property. During a Pilot run, TestFactory randomly enters one of the string cases in the input control.

To specify string cases for a UI object or component:

1. In the Properties view or Property List dialog box, expand the **Pilot** properties group.

2. If the value for **UseStringCases** is set to **No**, change the value to **Yes**.

3. To open the StringCases dialog box, double-click the **Value** field for **StringCases**.

The following figure shows the StringCases dialog box for an input UI object that has no data entry style assigned to it.
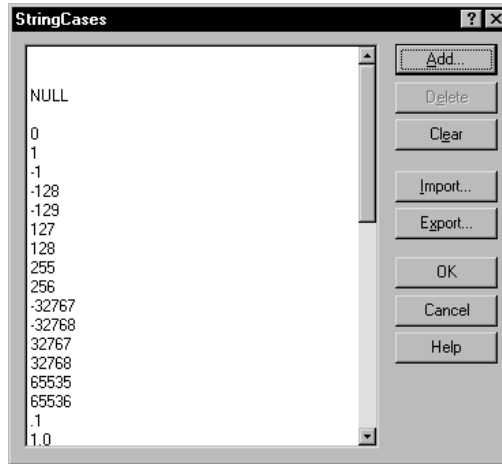
StringCases dialog box for an input UI object that has no data entry style assigned to it

4. To change the StringCases list, do one or more of the following:

   – To add a value to the list, click **Add**, and then type a value in the active box.

   – To delete a listed value, click the value, and then click **Delete**.

   – To delete all listed values, click **Clear**.

   – To edit a value, click the value, and then modify or replace it.

5. To import a text file that you created in another program, and use the text strings it contains as string cases for testing, click **Import**, and then use the Import From dialog box to locate and load the file.

> **NOTE:** TestFactory uses line breaks to distinguish between separate string cases. Be sure to use line breaks between text strings in the text files you create to import in TestFactory.

6. To export listed values to a text file, click **Export**, and then use the Export To dialog box to name and save the file. You do not need to specify a file type.

7. To save your changes and close the StringCases dialog box, click **OK**.

Specifying Mask Cases to Use During Testing

TestFactory can use entry data generated from mask cases to test input controls. To specify mask cases used to generate test data for an input control, you can change the MaskCases property of the UI object and components that represent the control.

To modify the MaskCases property for a UI object or component.

1. In the Properties view or Property List dialog box, expand the **Pilot** properties group.

2. If the value for **UseMaskCases** is set to **No**, change the value to **Yes**.

3. To open the MaskCases dialog box, double-click the **Value** field for **MaskCases**.
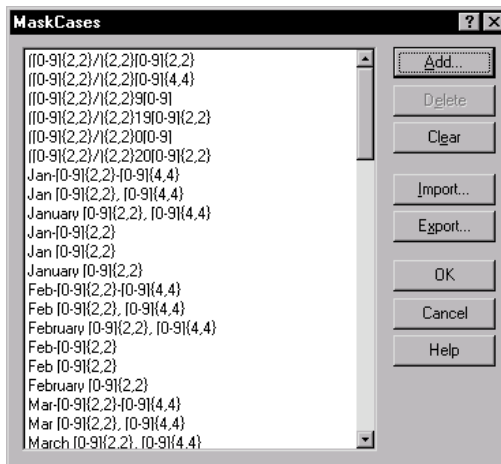
MaskCases dialog box for an input UI object that has no data entry style assigned to it

4. To change the MaskCases list, do one or more of the following:

    –   To add a value to the list, click **Add**, and then type a value in the active box.

    –   To delete a listed value, click the value, and then click **Delete**.

    –   To delete all listed values, click **Clear**.

    –   To edit a value, double-click the value, and then modify or replace it.

5. To import a text file, click **Import**, and then use the Import From dialog box to locate and load the file.

6. To export listed values to a text file, click **Export**, and then use the Export To dialog box to name and save the file. You do not need to specify a file type.

7. To save your changes and close the MaskCases dialog box, click **OK**.

Specifying Random Entry Data to Use During Testing

The **UseIntegerValues**, **UseFloatValues**, and **UseStringValues** properties in the Pilot properties group correspond to the **Use random entries** suboptions in the Edit Data Entry Style dialog box.

To keep a random data entry type as part of the mix of entry data used to test a control:

▶   Leave the **Value** field for the random entry data property set to **Yes**.

To exclude a random data entry type from the mix of entry data used to test a control:

▶   Double-click the **Value** field for the random entry data property, and then click **No**.

Excluding Controls from Testing

If the AUT contains a print control, a control that is unstable, or a control that could delete data that you want to keep, you probably want to exclude it from testing.

To exclude a control from testing by an individual Pilot, you can add the UI object mapped for the control to the **Exclude** tab. If the UI object that you add to the **Exclude** tab has been added as a component to an interaction object to which the Pilot has access, then you must also exclude the accessible UI object components from testing.

To exclude just the component from testing:

1. In the application map, click the interaction object that contains the component.

2. In the Interaction Object view, click the component name, and then click **Make Unavailable** on the Interaction Object toolbar.

To exclude the interaction object that contains the component from testing:

1. In the application map, right-click the interaction object that contains the component.

2. If **Use During Testing** on the shortcut menu is checked, click the command. If the command is not checked, click outside the shortcut menu to close it.

If a project contains a UI script or best script that touches a control you have excluded from testing, and you run the script while the exclusion is in effect, the run will fail. We recommend that you do not run scripts such as these as long as the control is excluded from testing.

To exclude a control from testing during *all* Pilot runs, you can modify the value of the ExerciseDuringTesting property for the UI object mapped for the control.

To exclude a control from all testing:

1. In the application map, click the UI object mapped for the control that you do not want to test.

2. In the Properties view, click **Pilot**.

3. Click the **Value** field for **ExerciseDuringTesting**, and then select **Never**.

4. If the UI object mapped for the control that you want to exclude from testing has been added as a component to one or more interaction objects, do one of the following for every interaction object that contains the component:

   – Exclude the component from testing.

   – Exclude the interaction object that contains the component from testing.

If the project contains a UI script or best script that touches a control that you exclude from testing, and you run the script while the exclusion is in effect, the run will fail. We recommend that you do not run scripts such as these as long as the control is excluded from testing. If you plan to permanently exclude a control from testing, then delete or edit the scripts that exercise the control.

Restoring the Default Property Values for UI Objects and Components

If you have modified the UI object properties of a UI object in the application map, and you want to restore the default property values for that object, you can do so at any time.

To restore the default property values for a UI object:

▶ In the application map, right-click the object, and then click **Restore UI Properties** on the shortcut menu.
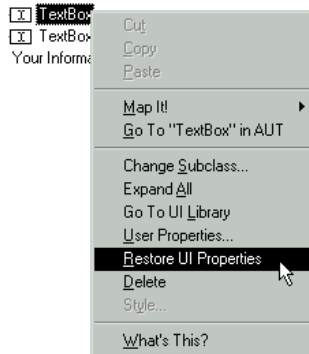
TestFactory restores the original default values of all of the UI object properties.

If you have modified the properties of a component, and you want to restore the original property values, you can do so by reassigning the *Base* style to the component.

1. In the Interaction Object view, click the component name.

2. To open the Edit Data Entry Style dialog box, click **Style Properties** on the Interaction Object toolbar.

3. Under **Style**, click the **Name** box, and then click **Base**.

Developing and Running a Test Suite

This chapter describes Test Suites and how to use them to run scripts on your local machine. This chapter includes the following topics:

▶ Overview of Test Suite functionality

▶ Creating a Test Suite

▶ Running a Test Suite

▶ Viewing the results of a Test Suite run

Overview of Test Suite Functionality

You can insert a Test Suite object in the application map to organize and track scripts and to run them in batches. Test Suites are useful for organizing and running groups of best scripts, defect scripts, and scripts created against a specific build of the AUT.

In addition to Pilot-generated scripts, a Test Suite can include other Test Suites, customized TestFactory scripts, and Robot scripts. Because a Test Suite contains pointers to the scripts that you insert in it, you can include a specific script or Test Suite in several different Test Suites.

If you run a Test Suite on your local machine, TestFactory runs the scripts in the order that you specify. The sequence in which you arrange them depends mostly on which script run results you want to see first. However, a particular script run sequence can affect the AUT in unexpected ways. You can use Test Suites to explore such possibilities.

The Test Suite **Coverage** tab provides an aggregate code coverage value for all of the scripts that it contains. You can use the results to determine how much code was exercised during a Test Suite run.

The Test Suite is a convenient tool for organizing groups of scripts for regression testing. You can drop Pilots into several functional areas of the application map, run them to obtain best scripts and defect scripts, and then place the scripts in Test Suites.

Create Test Suites for the following:

▶ Best scripts, so that you can run them against subsequent builds of the AUT

▶ Scripts that exercise a specific functional area of the AUT

▶ All defect scripts, or defect scripts to run against a particular build

▶ All of your scripts

Use a Test Suite to organize best scripts for functional testing after you have debugged them, inserted verification points, and successfully run the scripts against the AUT.

Creating a Test Suite

You can create a Test Suite using the Find Objects window or by directly inserting a Test Suite object in the application map. This section provides instructions for using both methods.

Creating a Test Suite Using the Find Objects Window

To create a Test Suite using the Find Objects window:

1. Click **Edit > Find Objects**, or click **Find Objects** on the Standard toolbar.
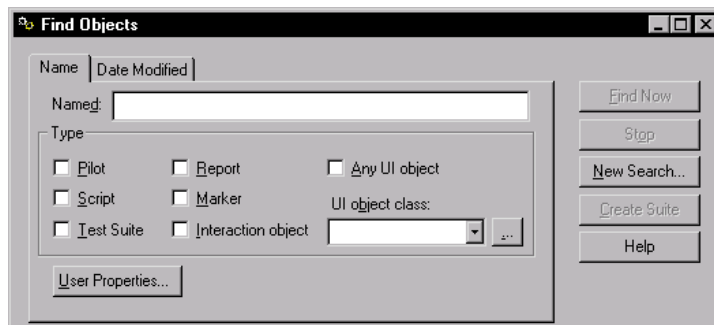
2. To locate scripts and Test Suites to add to the Test Suite, under **Type**, select the appropriate check boxes, and then click **Find Now**.
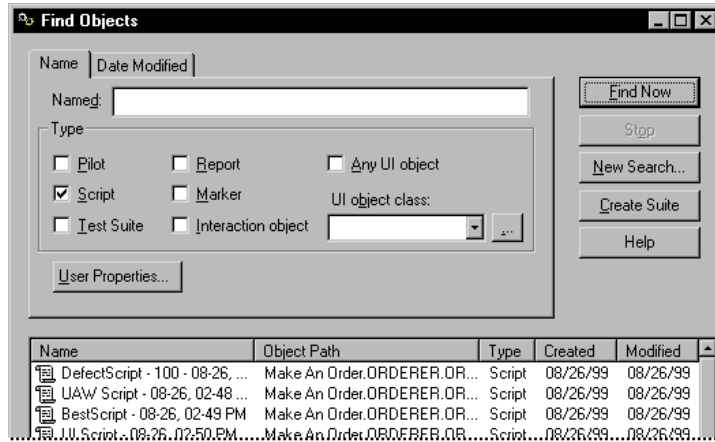
3. To add all of the found test objects to the Test Suite, click **Create Suite**. To add a subset of the objects listed to the Test Suite, press CTRL or SHIFT, select the test objects, and then click **Create Suite**.
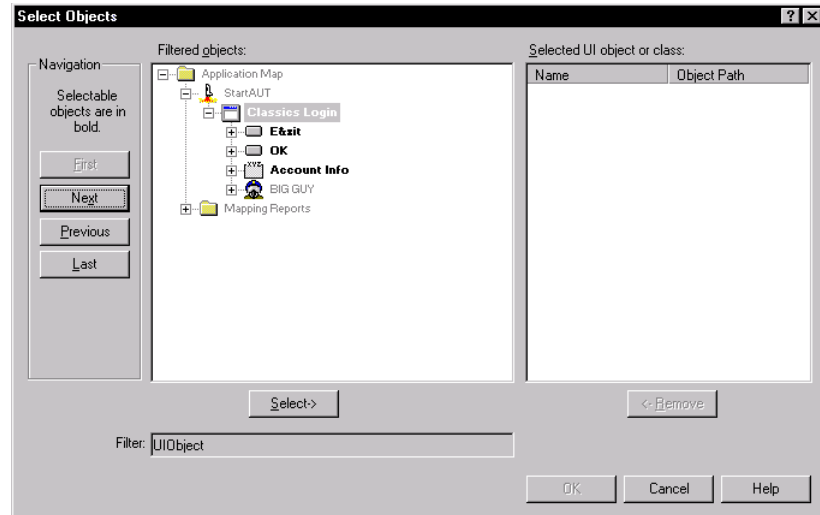
4. In the **Filtered objects** box, click a parent object for the Test Suite, and then click **Select**.

NOTE: The parent object that you select does not limit what you can include in the Test Suite. We recommend that you select a parent object such as a folder that makes sense in terms of project organization.

5. Click **OK**, and then close the Find Objects window.
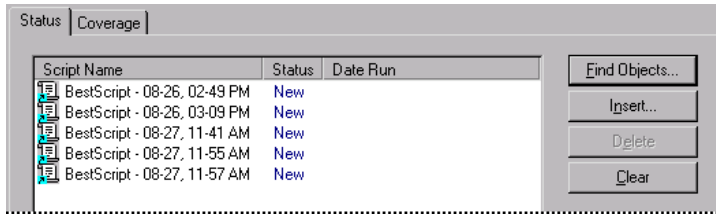
6. To change the order in which TestFactory runs a script or Test Suite, click its name, and then use the **Up** and **Down** buttons. If the Test Suite includes startup and cleanup support scripts, then place them at the top and bottom of the list, respectively.

7. To remove one or more items from the list, select the item(s), and then click **Delete**.

> **NOTE:** If you delete a test object name in the **Status** tab, you delete only the *pointer* to the object, and not the test object itself.

Inserting a Test Suite Object in the Application Map

To create a Test Suite by directly inserting a Test Suite object in the application map:

1. Insert a Test Suite object in the application map, and then name it.

2. To add a script or another Test Suite to the Test Suite, do one of the following:

– Drag a script or Test Suite object from the application map or a script from the Robot Scripts folder to the **Status** tab list box.

Alternatively:

a. On the **Status** tab, click **Find Objects**.

b. To find scripts or other Test Suites to add to the Test Suite, select the appropriate check boxes, and then click **Find Now**.

c. To add all of the found objects to the Test Suite, click **Accept**. To add a subset of the objects listed to the Test Suite, press CTRL or SHIFT, select the objects, and then click **Accept**.

3. To change the order in which TestFactory runs a script or Test Suite, click the test object name in the **Script Name** column, and then use the **Up** and **Down** buttons. If the Test Suite includes startup and cleanup support scripts, then place them at the top and bottom of the list, respectively.

4. To remove a script or Test Suite from the run sequence, click its name in the **Script Name** column, and then click **Delete**.

> NOTE: If you delete a test object in the **Status** tab, you delete only the *pointer* to the object, and not the test object itself.

Running a Test Suite

You can start a Test Suite run from the Test Suite **Status** tab or from the AutoPilot dialog box. This section describes how to use the **Status** tab to start a Test Suite run locally. For information about running a Test Suite using the AutoPilot, see Chapter 7, *Using the AutoPilot*.

> NOTE: In order for TestFactory to calculate code coverage for every script in a Test Suite, each script must start and quit the AUT. For example, if the first script in a Test Suite starts the AUT but does not quit it, and the next script exercises a dialog box and then quits the AUT, TestFactory only calculates code coverage for the first script.
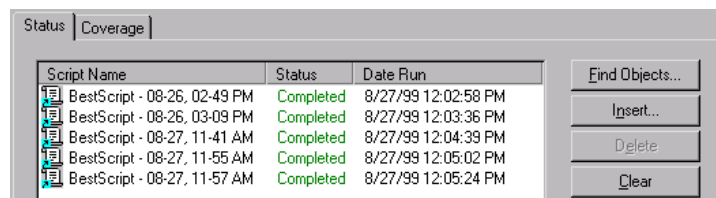
If you run a Test Suite against an object code-instrumented AUT, TestFactory does not automatically calculate code coverage values for the Robot scripts included in the Test Suite. To get code coverage values for a Robot script, you must first insert the Robot script in a customized TestFactory script, and then remove the script code that was recorded in Robot to start the AUT.

Running a Test Suite on Your Local Machine

To start a local Test Suite run from the **Status** tab:

1. In the application map, click the Test Suite object.

2. In the **Status** tab, click **Start**.

3. To stop the Test Suite run before it finishes, click **Stop** in the Script progress bar, or press ALT + F12.

After the Test Suite run ends, the **Status** tab displays summary run results for individual scripts and Test Suites.

Viewing the Results of a Test Suite Run

After a Test Suite run is completed, TestFactory displays the results on the **Status** and **Coverage** tabs in the right pane. This section describes how to examine these results.

Viewing Test Suite Run Results in the Status Tab
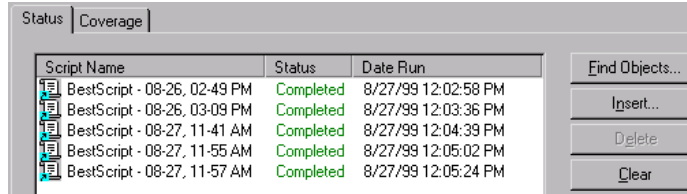
After a Test Suite run is completed, the **Status** tab displays the following:

▶ **Script Name** – The name of the script or nested Test Suite.

▶ **Status** – The completion status for the script or nested Test Suite.

▶ **Date Run** – The date and time the script run was completed.

If a script fails, double-click the script name to jump to the script object in the application map, and then follow the steps for debugging a script that are provided in Chapter 5, *Automatically Generating Scripts*.

Viewing Test Suite Run Results in the Coverage Tab

The **Coverage** tab displays detailed coverage information on the Test Suite run.

To view the code coverage details for a Test Suite run:

1. Click the Test Suite object in the application map, and then click the **Coverage** tab. TestFactory calculates the **Code Coverage** value shown based on what the scripts touched relative to all source code in the application. The values displayed reflect the aggregate code coverage 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 the AUT and the aggregate code coverage values for the scripts that touched them.
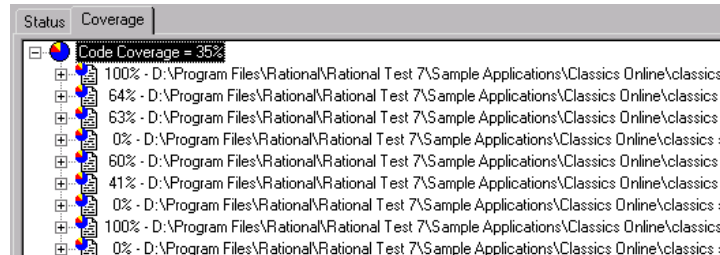
3. To see the coverage values for individual procedures within a source file, expand the source file.
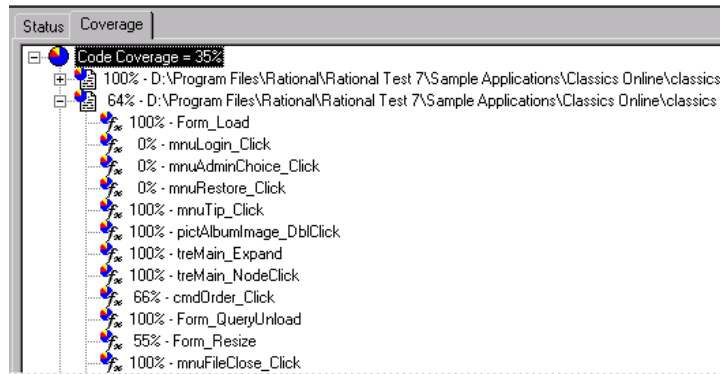
4. If you instrumented the source files for the AUT, then you can view the source code for a procedure in the Coverage Browser. To open the Coverage Browser, double-click the procedure.

Viewing Logs for Scripts in a Test Suite

You can access the logs for every script run in the Test Suite through the individual script objects in the application map. For information about viewing logs, see *Viewing the Log for a Defect Script* on page 5-17.

Viewing Code Coverage Values for Previously Run Scripts

If you have several scripts that were generated for the same instrumented build of the AUT, you can quickly obtain an aggregate code coverage value for the scripts just by inserting them in a Test Suite.

To see aggregate code coverage for previously run scripts:

1. Insert a Test Suite in the application map.

2. Add the previously run scripts to the Test Suite.

3. Click the **Coverage** tab and then view the code coverage details as you would for a completed Test Suite run.

Using the AutoPilot

This chapter describes the AutoPilot and provides instructions on how to queue Pilots, Test Suites, and scripts in the AutoPilot window and run them on your local machine. This chapter includes the following topics:

▶   About the AutoPilot

▶   Using the AutoPilot to run Pilots, Test Suites, and scripts

About the AutoPilot

The AutoPilot automatically runs multiple Pilots, Test Suites and scripts as a batch job. You can use the AutoPilot to run batch tests on your local machine.

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

Running Tests on Your Local Machine

To run multiple Pilots, Test Suites, and scripts on your local machine using the AutoPilot:

1. Click **Tools > AutoPilot**, or click **AutoPilot** on the Standard toolbar.
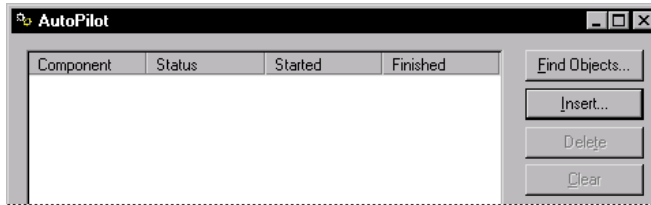
2. To queue Test Suites, Pilots, and scripts to run from the AutoPilot window:

   a. Click **Find Objects**.
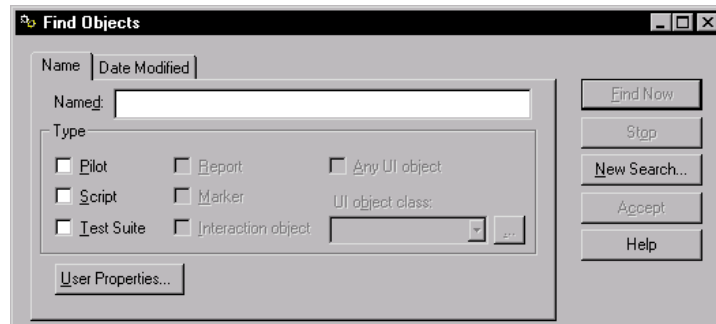
b. To locate Test Suites, Pilots, or scripts, under **Type**, select the appropriate check box(es), and then click **Find Now**.
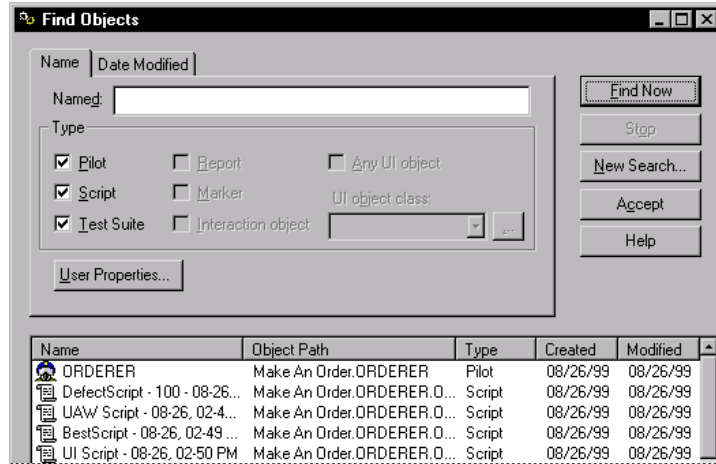
c. To add all of the objects listed to the AutoPilot window, click **Accept**. To add a subset of the objects, select the objects, and then click **Accept**.

Alternatively, do the following to queue Test Suites, Pilots, and scripts to run from the AutoPilot window:

a. On the AutoPilot window, click **Insert**.
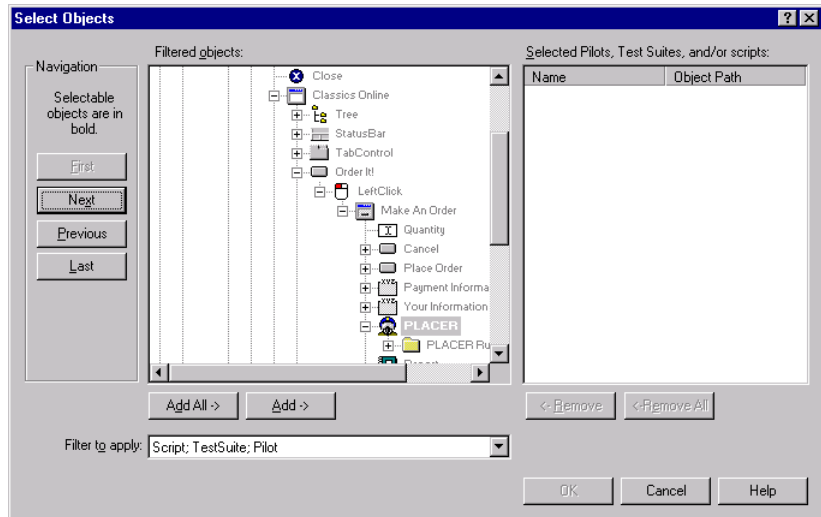
    b. To add all of the selectable objects in the **Filtered objects** list, click **Add All**. Otherwise, select an individual object to add, and then click **Add**. Do this for all of the objects you want to add to the AutoPilot queue.

    c. After you finish adding test objects to the list of selected objects, click **OK**.

3. To change the order of a test object in the AutoPilot queue, click the object, and then use the **Up** and **Down** buttons.

> **NOTE:** If the objects listed include a combination of Pilots, Test Suites, and scripts, group the Pilots either at the top or at the bottom of the list.

4. Click **Start**.

Testing Code Changes in Visual Studio

This chapter describes the TestCodeChanges add-in for Microsoft Visual Studio and includes the following topics:

- ▶ Overview of the TestCodeChanges add-in for Visual Studio

- ▶ Setting up the TestCodeChanges add-in

- ▶ Preparing to test code changes

- ▶ Using the TestCodeChanges add-in

Overview of the TestCodeChanges Add-In for Visual Studio

The **TestCodeChanges** add-in for Microsoft Visual Studio lets you access scripts that test changes you have just made to the source code of a Visual Basic or Visual C++ AUT from within Visual Studio.

After you set it up and load it in your Visual Studio development environment, TestCodeChanges does the following:

- ▶ Tracks all changes that you make to the source code of the AUT.

- ▶ Gives you access to all of the scripts in the project that exercise the changed code.

- ▶ Displays the code coverage values for scripts that touch changed source code files.

- ▶ Lets you filter changed code files based on the calendar dates that the source code was changed.

- ▶ Lets you create a regression suite of scripts, which you can then run from your Visual Studio development environment to test changed files.

- ▶ After testing, automatically displays script run results.

Setting Up the TestCodeChanges Add-In

To set up the TestCodeChanges add-in, you must have Rational TestFactory installed on your system.

To set up the TestCodeChanges add-in:

1. Click **Start > Programs > Rational Suite TestStudio > Rational Test > Set Up Rational TestCodeChanges**.
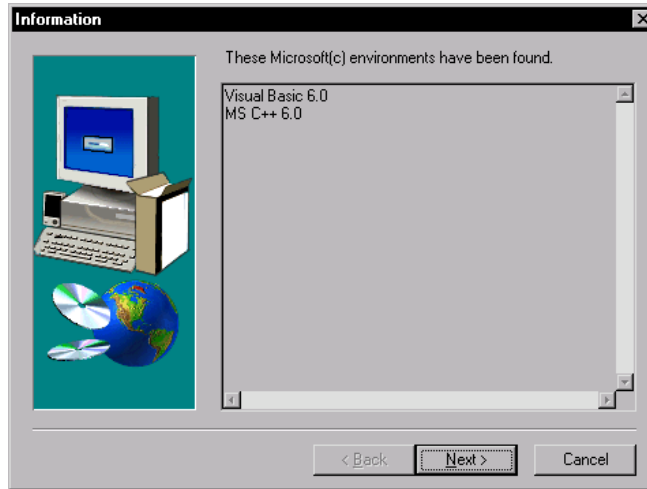
The Information dialog box lists the Microsoft development environments detected on your system.

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

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

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

Preparing to Test Code Changes

The requirements for testing changes that you have made to the AUT source files are as follows:

▶ The AUT must be instrumented.

▶ You must first recompile the executable file for the open project.

▶ If you are working in Visual Basic, you must save the changed source files after you recompile the executable file.

▶ If you plan to run Robot scripts to test code changes, you must first run the scripts from TestFactory so that TestFactory can calculate code coverage values for them.

The add-in automatically displays all of the scripts that exercise the changed files *and* that have code coverage values. You can select some or all of these scripts to include in a regression suite that you can then run to test changed files. You can add scripts that do not exercise changed files to the regression suite. For example, you may want to include a support script that brings the system to a state necessary for testing.

Using the TestCodeChanges Add-In

After you recompile the executable file and save the changed source files for the AUT, you can start the TestCodeChanges add-in, log on to a Rational project, and view the scripts that exercise the changed source code files in the Test Code Changes window.

Starting the TestCodeChanges Add-In

To start the TestCodeChanges add-in:

1. Click **TestCodeChanges** on the toolbar of your Visual Studio application.

2. After the Rational Login dialog box opens, log on to the project for the AUT.
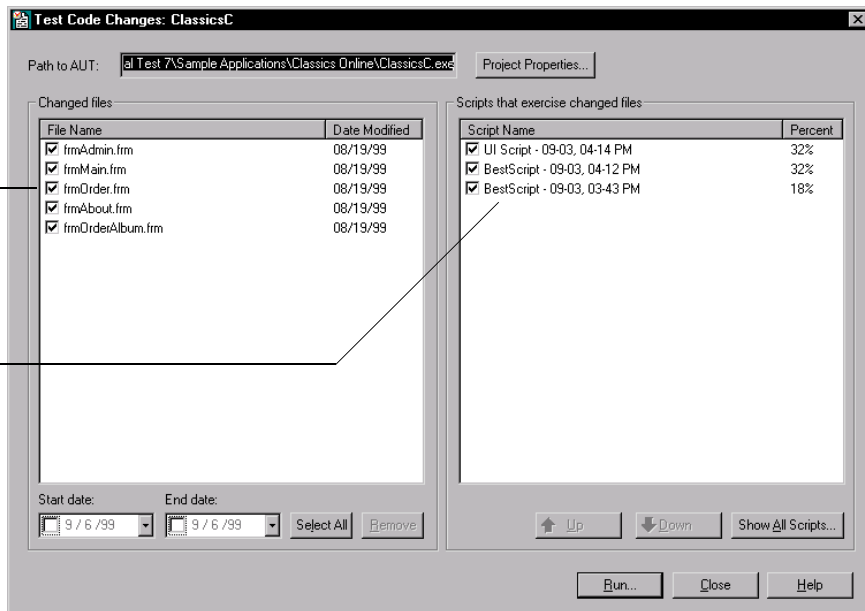
All changed project files are listed in the left pane.

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

Test Code Changes: ClassicsC		☒
Path to AUT:	al Test 7\Sample Applications\Classics Online\ClassicsC.exe	Project Properties...

Changed files

File Name	Date Modified
☑ frmAdmin.frm	08/19/99
☑ frmMain.frm	08/19/99
☑ frmOrder.frm	08/19/99
☑ frmAbout.frm	08/19/99
☑ frmOrderAlbum.frm	08/19/99

Scripts that exercise changed files

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%

Start date: 9 / 6 /99   End date: 9 / 6 /99   Select All   Remove

Up   Down   Show All Scripts...

Run...   Close   Help

Viewing Information in the Test Code Changes Window

This section describes the information displayed in the Test Code Changes window.

Viewing and Modifying the Path to the Executable File to Test

The **Path to AUT** box displays the full path to the executable file to be tested from the add-in window. You can specify a different executable file to test by modifying the project properties.

To specify a different executable file to test:

1. Click **Project Properties**.
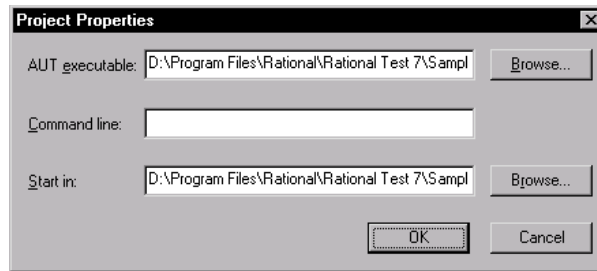
2. In the **AUT executable** box, enter the path to a different executable file.

3. To pass one or more command-line arguments to the AUT when it is started, type the argument in the **Command line** box. Be sure to specify the command-line argument in the syntax that the AUT requires.

4. In the **Start in** box, enter the full path to the working directory. If you do not enter a start path, TestCodeChanges uses the AUT executable path.

5. Click **OK**.

Viewing Changed Files and the Scripts that Exercise Them

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, with the most recently changed file listed at the top.

To toggle the sort order of the displayed source code files:

▶ Click the **Date Modified** column heading.

If the **Changed files** list displays more than one source file, and you want to determine which of the listed scripts exercises one of the files, leave the check box next to the file name selected and clear the check boxes next to all of the other file names.

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 reverse the listed order of the scripts:

▶ Click the **Percent** column heading.

Creating and Running a Regression Suite to Test Code Changes

To create a regression suite to test code changes, select the listed scripts that you want to include in the regression suite, use the Available Scripts dialog box to add other scripts from the project, and then set the run order of the scripts.

To select scripts to add to the regression suite:

1. To filter the changed files displayed based on the date interval during which the code was changed:

   a. To set an interval start date, click the down arrow in the **Start date** box, and then click a calendar date.

   b. To set an interval end date, click the down arrow in the **End date** box, and then click a calendar date.

2. To exclude a changed file from testing, under **Changed files**, do one of the following:

   – Clear the check box next to the file name.

   – Click the file name, click **Remove**, and then click **Yes** in the message box. The next time you open the Test Code Changes window, the file is not listed (unless it is subsequently changed).

   – Right-click the file name, click **Remove** on the shortcut menu, and then click **Yes** in the message box. The next time you open the Test Code Changes window, the file is not listed (unless it is subsequently changed).

3. To exclude a script from testing, do one of the following:

   – To exclude a script from the current testing session, under **Scripts that exercise changed files**, clear the check box next to the script name.

   – To exclude a script from testing and remove it from the window, right-click its name, click **Remove** on the shortcut menu, and then click **Yes** in the message box. The next time you open the Test Code Changes window, the script is not listed (unless you select it manually or a file that it exercises is subsequently changed).

4. To exclude all of the scripts that exercise a changed file from the current testing session, under **Changed files**, clear the check box next to the file name.

If you want to run a support script before a listed script is run, but the setup script does not exercise changed code (and is not included in the **Script Name** column), you can still add it to the regression suite.

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

   a. To view all available scripts, do one of the following:

   – Click **Show All Scripts**.

   Alternatively,

   – Under **Scripts that exercise changed files**, right-click anywhere on the list, and then click **Show All Scripts** on the shortcut menu.

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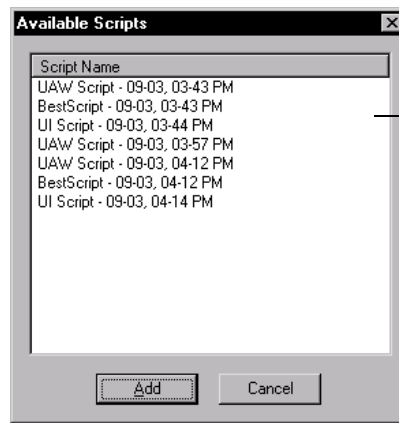
All of the scripts that you can add to the regression suite are listed in the Available Scripts dialog box.

   b. To add a listed script to the suite, click the script name, and then click **Add**.

   c. To add a multiple scripts, hold down the CTRL or SHIFT key, select multiple script names, and then click **Add**.

6. To add a copy of a listed script to the bottom of the **Script Name** list, right-click the script name, and then click **Duplicate** on the shortcut menu.

   The add-in runs scripts in their listed order. If the **Script Name** column lists a support script that must be run before another listed script, you can change its order in the list.

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

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

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

Run Results for the Regression Suite

After the regression suite run is completed, TestCodeChanges displays the run status for each script in the **Status** column of the Status Report dialog box. 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, or that the script run was interrupted).

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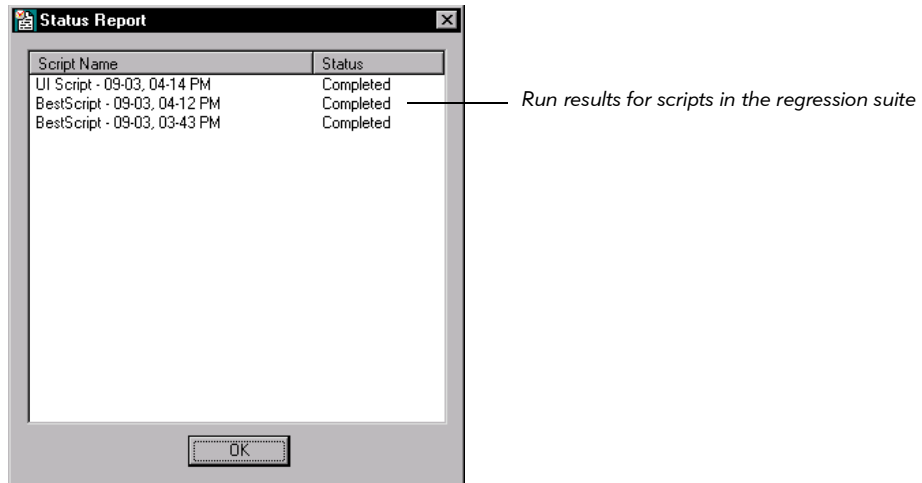
If the run results for scripts that exercise a changed file were successfully completed, you can check in the modified code. If a run result failed, examine the logs, make necessary changes to the source code, and then rerun the regression suite.

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

Using TestFactory Command-Line Arguments

TestFactory Command-Line Arguments

You can start TestFactory from the command line and include arguments to control its behavior on start-up. TestFactory command-line arguments can be specified in the Run dialog box, or in a TestFactory shortcut Properties dialog box.

TestFactory accepts the following arguments:

Logon Arguments

Use	With	To
-u	user_name	Enter a user name for TestFactory.
-w	password	Enter a password for TestFactory.
-p	project_path	Open a project.

Run Arguments

Use	With	To
-o	`object_path`	Select a UI object in the application map.
-o	`parent_object_path.TFobject_name`	Select a TestFactory object in the application map.
-l	`pilot_name`	Run a full-depth Pilot.
-q	`pilot_name`	Run a single-level depth Pilot.
-s	`script_name`	Run a script.
-t	`test_suite_name`	Run a Test Suite.

Application Mapper Arguments

Use	With	To
-g	`object_path`	Map the AUT to full depth from a specified starting object.
-n	`object_path`	Map the AUT to single-level depth from a specified starting object.
-a	`AUT_path`	Specify a complete path to the AUT.
-c	`AUT_arguments`	Specify command-line arguments to pass to the AUT.
-y	`startup_dir`	Specify a start-up or working directory for the AUT.

Coverage Dictionary Arguments

Use	With	To
-i	dictionary_path	Import the instrumented coverage dictionary from the specified location.
-e	dictionary_path	Export the instrumented coverage dictionary to the specified location.

NOTE: Be sure to specify the full path to the coverage dictionary, including the .vcd file name extension.

Control Argument:

Use	With	To
-x	\<none>	Exit TestFactory.

NOTE: TestFactory executes the exit argument only after it executes all run and Application Mapper arguments.

Command-Line Argument Format

Begin each command-line argument with the minus (-) or the slash (/) character, followed by a single letter. Place the argument itself immediately after the argument letter.

To use any of the command-line arguments, you must specify the user name, project, and, if set, a password, as in the following example:

```
rtfact.exe -u"user_name" -p"project_path" -wPassword
```

TestFactory executes multiple run arguments in the order in which you list them in the command line.

Rules for Using TestFactory Command-Line Arguments

You can use logon arguments by themselves to bypass the logon dialog box of the AUT. However, you must use run and Application Mapper arguments in conjunction with logon arguments.

If you enter the -g or the -n Application Mapper argument without specifying a starting object path, the Application Mapper will start mapping from the "Application Map.StartAUT" object path.

The Application Mapper arguments -a, -c, and -y are not required to run the Application Mapper. If you do not use the -g or the -n argument, then TestFactory ignores other Application Mapper arguments.

If you use the -o command-line argument to select a UI object in the application map, use the application map path for the UI object. Be sure to use an ampersand for mnemonics in the object path name. To find the correct object path to use, do the following:

1. In TestFactory, click **Edit > Find**.

2. Select the check box for the object type to find, and then click **Find Now**.

3. In the list of objects found, check the value in the **Object Path** column for the object that you want to select. Use this value in the command-line argument.

If you use the -o command-line argument to select a TestFactory object such as a Test Suite or folder, use the object path for the parent UI object of the TestFactory object, followed by the TestFactory object name. Be sure to use an ampersand for mnemonics in the object path name.

►►► Index

T

V

W