**Chapter 1. Overview of Rational Functional Tester**

**Chip Davis, Daniel Gouveia, Fariz Saracevic, Lee B. Thomas, Marc van Lint**

*This chapter provides an overview of the basic concepts and the primary usages of Rational® Functional Tester. The goal is not to duplicate what is documented in the product’s Help, although some of this information can be found there. This chapter does not attempt to go into great detail about any particular topic; subsequent chapters fulfill that purpose. By the end of this chapter, you should have a general understanding of what Rational Functional Tester is and how to use its key capabilities.*

**Architecture of Rational Functional Tester**

This section introduces the general architecture of Rational Functional Tester, which is described in detail throughout the rest of the book. You can think of Rational Functional Tester as having three different modes of operation: normal edit mode, recording mode, and playback mode. Most of the time, you will work in edit mode. Recording and playback modes are significant because they are not passive about what you do with the keyboard and mouse of your computer. This is explained in the sections “[How Tests Are Recorded](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01sec2lev3)” and “[How Tests Are Executed](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01sec2lev4).”

**How Test Assets Are Stored**

Rational Functional Tester is a desktop file-based tool and does not have a server component. All Rational Functional Tester information is stored in files, primarily Java™ and XML, which users typically access from shared folders. You can use a configuration management tool such as Rational Team Concert, IBM® Rational ClearCase®, or simpler tools such as PVS to version control individual test assets. There is no database component, although it is possible to get test data directly from a database table. This is less common, however, and you will typically keep test data in files along with the other test assets.

If you use Rational Functional Tester Java scripting, your Functional Test project is a special kind of Java project in Eclipse™. If you use Rational Functional Tester Visual Basic® .NET scripting, your Functional Test project is a special kind of Visual Basic project in Microsoft® Visual Studio®. Both kinds of projects are created through Rational Functional Tester.

You can become highly proficient with Rational Functional Tester, even an expert user, without needing to concern yourself with most of the underlying files created by the tool. All of the test assets that you work with are created, edited, and maintained through the Rational Functional Tester interfaces. Most of the test assets, such as a test script, consist of different underlying files on the file system. The exact file structure for the Java scripting and Visual Basic .NET scripting versions are almost exactly the same with only minor differences.

**How Test Results are Stored**

Test results are stored in test logs, which can be stored in several different formats of your choice. You can choose how to save test logs based on the nature of the particular testing effort and what you use for test management. For example, a small informal testing effort might simply save all test results into HTML files. Another larger testing effort might send the test results to a test management tool, such as IBM Rational Quality Manager. Following are the options for storing test logs in Rational Functional Tester:

• HTML

• Text

• Test and Performance Tools Platform (TPTP)

• XML

• Rational Quality Manager

Note

Note that IBM Rational Quality Manager is not required for test logging or for any other functions or uses described in this book. This is presented only as an optional tool for test management, which is often employed in testing efforts.

**How Tests Are Recorded**

You are likely to use the recorder in Rational Functional Tester to create new test scripts. The reason for this is that the recorder is usually the fastest and easiest way to generate lines of test script, even if that script is extensively modified later. Whether you capture long linear test procedures, developing a keyword-driven test framework, or do something in between, the recording mode is the same. When Rational Functional Tester goes into recording mode, it captures all keyboard and mouse input that goes to all enabled applications and environments. Every time you press a key or perform anything with a mouse, other than simply moving the pointer, it gets captured into the test recording. The exceptions to this are: Rational Functional Tester does not record itself and it does not record applications that are not enabled. You must be careful when you are recording and be sure that you do not click or type anything that you do not want to be part of your test.

Rational Functional Tester creates the test script as it is recording; there are no intermediate files or steps to generate the test script, and you can view the steps as you record. Some information about the test is stored in files that are different from the test script; this includes test objects, verification points, and test data. These files are hidden, and you see abstractions of them only as test assets in the test script.

The test scripts are either Java or Visual Basic .NET files, which must be executed through Rational Functional Tester. These are not just any Java or Visual Basic .NET files, however. They are extensions of the com.rational.test.ft.script package and include several other packages for functional testing, which is what makes them automated tests. Using the recorder or creating a new blank test from within Rational Functional Tester automatically sets up the required packages so you do not have to manually do this.

Although there are many techniques for recording tests, you always capture steps that interact with an application or system interface. Unlike many unit testing or developer testing tools, there is nothing that automatically generates tests by “pointing” to a class, interface, or package.

**How Tests Are Executed**

When you run a test in Rational Functional Tester, the machine goes into playback mode. In playback mode, Rational Functional Tester sends all of the mouse and keyboard actions that you recorded to the application under test. While Rational Functional Tester is “driving” the computer, it does not lock the user out from also using the mouse and keyboard. In general, you should not touch the keyboard or mouse when Rational Functional Tester is in playback mode. However, at times you can run tests in interactive mode to manipulate potential playback issues.

A test script is comprised largely of statements that interact, including performing tests, with various objects in the application under test. When you execute a test script, Rational Functional Tester first has to identify and find each object by matching recognition properties in the script’s saved test object map against the actual objects that are present at runtime. If Rational Functional Tester cannot find a close enough match, it logs an error and either attempts to continue or aborts. If Rational Functional Tester does find a matching object, it performs the action on the object. These actions might be user interactions, such as clicking, selections, or other operations such as getting or setting values. Finally, the actions performed on the object might be a test (a verification point), in which case Rational Functional Tester compares some saved expected value or range with an actual runtime result. Although every statement (line of code) in the script produces a result, you normally see only verification points (the tests) and other key results in the test log that are created for every test run.

You can either play a test back on the same machine or on any other machine running a Rational Agent, which gets installed by default with Rational Functional Tester. You can also run multiple tests on multiple remote machines for distributed functional testing. This makes it possible to complete much more testing in a shorter period of time. A given machine can run only one test at a time, or many sequentially, but you cannot run multiple tests in parallel on the same machine. Although it is not required, you can also execute Rational Functional Tester tests on remote machines using test management tools, such as Rational Quality Manager.

**Integrations with Other Applications**

Rational Functional Tester is a stand alone product that does not require other tools or applications. Integration with other tools or applications is optional and based on your particular needs. Following are some of the common types of applications that you can integrate with Rational Functional Tester:

• Test management or quality management, such as IBM Rational Quality Manager

• Defect tracking or change request management, such as IBM Rational ClearQuest

• Configuration management or version control, such as IBM Rational Team Concert and IBM Rational ClearCase

• Unit or developer testing tools, such as JUnit

• Automated testing tools, such as IBM Rational Performance Tester

• Development tools, such as IBM Rational Application Developer

Most of the applications listed previously, especially those developed by IBM, require little or no work to set up the integration. Many applications and tools, such as JUnit, Rational Service Tester, Rational Software Architect, or WebSphere® Integration Developer (to name a few) run in the Eclipse shell and can share the same interface as Rational Functional Tester. With these tools, you can switch between Rational Functional Tester and the other tools simply by switching perspectives (tabs).

In addition to these applications, you can also integrate Rational Functional Tester with many other kinds of applications. This requires varying amounts of work to implement the integration, although you can find existing examples on IBM developerWorks® (www.ibm.com\developerworks\rational\). These include:

• Custom-built test harnesses (extending the test execution)

• Spreadsheets (for logging or simple test management)

• Email notifications

**Installation and Licensing**

You learn about product installation and licensing in the subsequent sections.

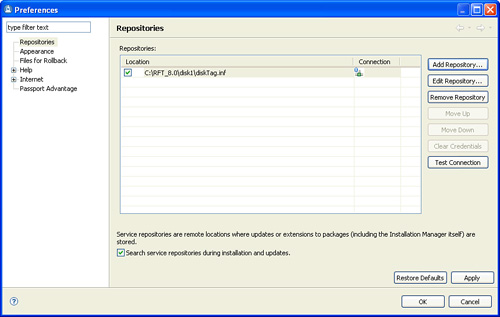
**Product Installation**

It is important to properly install Rational Functional Tester. The installation process has been streamlined with the Rational Functional Tester v8.0, and Installation Manager is used for its installation.

If you do not have Installation Manager, you are installing a Rational product using Installation Manager for the first time. In that case, you install Rational Functional Tester v8.0 either by clicking on launchpad.exe or install.exe, located under ..\disk1\InstallerImage\_win32\. In this scenario, Installation Manager and Rational Functional Tester are installed at the same time.

This installation is based on the Rational Functional Tester installation via the existing Installation Manager. You can start Installation Manager if you select **Start > Programs > IBM Installation Manager > IBM Installation Manager**. Click **File > Preferences** to select **Repository used for Rational Functional Tester installation** as shown in [Figure 1.1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig01).

**Figure 1.1** Installation Manager preferences



When you select Repository, be sure that you use the letter drive for the installation path (for example, C:\RFT\_8.0\disk1\diskTag.inf). Using the Universal Naming Convention (UNC) path is known to cause issues.

After Repository is selected, click the **Install** button in the main window of the Installation Manager. You might be prompted to install the updated version of the Installation Manager. If that is the case, a new version of the Installation Manager is installed first and Installation Manager restarts. You need to click the **Install** button again, and the Rational Functional Tester installation process starts:

• Install Packages window provides details about the installation package. You can click the Check for Other Versions and Extensions button. This provides details about other available versions for installation. If you do not select the Show all versions checkbox, only the latest version is displayed. Select the desired version of the Rational Functional Tester to proceed with the installation.

• On the next window, you are prompted to read and accept the terms in the license agreement. You need to accept the terms, so that the Rational Functional Tester installation can continue.

• Upon acceptance of the terms of the license agreement, you are prompted to specify the location of the shared resources directory. For the best results, you should select the drive with the most available space because it must have adequate space for the shared resources of future packages. The default location is C:\Program Files\IBM\IBMIMShared. Click **Next** to continue with the installation.

• You need to select the package group next. A *package group* is a location that contains one or more packages. If this is the first IBM Rational installation, you need to create a new package group. If there is an existing package group, you can select to reuse the existing package group. You need to specify the installation directory. The default value is C:\Program Files\IBM\SDP. Click **Next** to continue with the installation.

• If you want to extend an existing version of Eclipse, Rational Functional Tester installation enables you to select Eclipse IDE and JVM. Be sure to have a compatible version of the Eclipse IDE on your system. If you are not sure if you have a compatible version of the Eclipse IDE, do not extend an existing Eclipse. After you make your selection, click **Next** to continue with the installation.

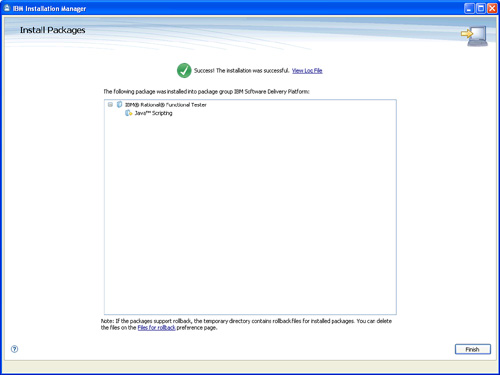
• If you are interested in using one of the ten supported languages, you need to select which language you would like to install. The default value is English. Selection of the language is used for translations for the user interface and the documentation being installed. Click **Next** to continue with the installation.

• You need to select features that you want to install. The default feature value is Java Scripting. You might select to install .NET 2003 or 2005 scripting instead. To see the dependency relationships between features, click **Show Dependencies**. Rational Functional Tester 8.1 has an option for Rational Team Concert.

• On the subsequent window, you are prompted to review summary information that contains information about packages, features, environment, and the repository selected. After you verify summary information, the installation can start.

• Installation Manager provides a status at the end of the installation. If successful, you should get a success message with the listing of the installed package as shown in [Figure 1.2](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig02).

**Figure 1.2** Install packages—successful installation message



**Product Licensing**

The market is shifting away from charging for individual GUI control sets for functional testing. Therefore, starting with Rational Functional Tester v8.0, all GUI control set extensions are included in the base installation.

The Rational Functional Tester v8.0 license is configured using Installation Manager. There are two supported license mechanisms. One uses a product activation kit and the other uses a floating license. The product activation kit is stored in a \*.jar file. Configuring with the activation kit is considered as a permanent license key.

If you use a floating license enforcement, you can enable floating licenses for Rational Functional Tester with a permanent license key. You need to obtain the license server connection information from your Rational administration. You then configure a connection to a license server in the Installation Manager to obtain access to floating license keys installed on a license server.

**Enabling the Environment for Testing**

Before you start creating test scripts against your application, you need to enable the environment for testing. Enabling the environment for testing ensures that controls used within environments are accessible by Rational Functional Tester.

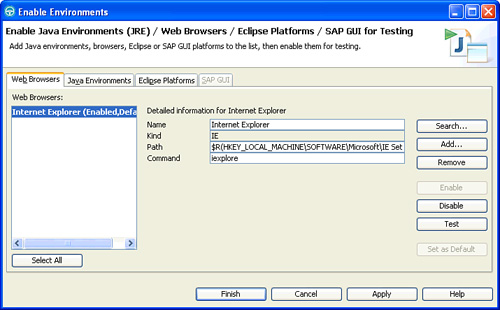
Clicking **Configure > Enable Environments for Testing** from the Rational Functional Tester menu opens a dialog box where you can add web browsers, Java environments, and the Eclipse or SAP GUI platform to the list, and then enable them for testing.

**Enabling Web Browsers**

The Web Browsers enabler must be run before you can use Rational Functional Tester to test an HTML application. Depending on the platform where Rational Functional Tester is installed, enabler behaves differently. On Windows® systems, the enablers look in the registry to discover any installed browser, and on UNIX®, the enabler scans hard drive(s) for any installed browsers.

Rational Functional Tester automatically enables Internet Explorer if it is present on the system. If you have different browsers such as Mozilla or Netscape 6 or 7, you must add and enable them manually via the Enable Environments Wizard shown in [Figure 1.3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig03).

**Figure 1.3** Enabling Web Browsers



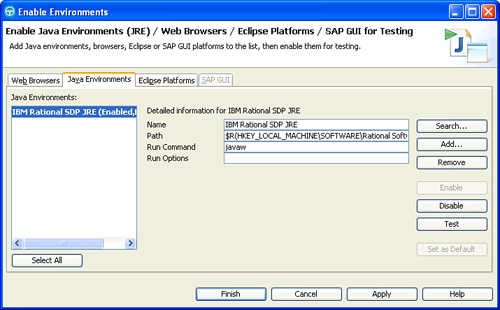
You should use Test button to validate if the web browser was properly enabled. Clicking the **Test** button invokes the Brower Enablement Diagnostic Tool. You need to run a diagnostic test to validate proper enablement of the web browser. If the web browser-enablement test result displays a success status, you have properly enabled web browser. If there are issues, the web browser-enablement test result shows a state failure with an explanation of the failure. Click the **Problem and Solution** tab for instructions about how to fix the problem.

Also, note that the first time you run Rational Functional Tester, it automatically enables the JVM of your browser’s Java plug-in so that HTML recording works properly. If you install a different JVM, you must rerun the enabler to enable it.

**Enabling Java Environments**

The Java Environment tab is used to enable Java environments and to add or configure Java environments, as shown in [Figure 1.4](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig04). Java enabler must be run before you can use Rational Functional Tester to test Java applications. The enabler scans hard drive(s) looking for Java environments. It enables the Rational Functional Tester to “see” your Java environments by adding files to the directory where your Java Runtime Environments (JRE) are located.

**Figure 1.4** Enabling Java environments



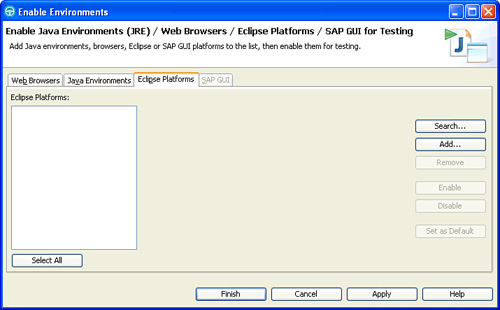
You should always perform a test when you add a new Java environment. Click the **Test** button, which invokes the JRE Tester. If the test runs successfully, JRE Tester displays the JRE version, vendor, and a message that JRE successfully enabled. If there is an issue, you need to rectify the issue before recording a test.

Rational Functional Tester is shipped with a JRE that is automatically enabled during installation. It is called IBM Rational SDP JRE. After the name of the Java environments, the enabler indicates in parentheses whether that environment is currently enabled.

**Enabling Eclipse Platforms**

The Eclipse Platforms tab is used to enable Eclipse or WebSphere WorkBench based platforms. Rational Functional Tester supports testing applications based on Eclipse 2.0 and 3.0, as shown in [Figure 1.5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig05).

**Figure 1.5** Enabling Eclipse platforms



The Eclipse enabler must be run before you can use Rational Functional Tester to test Eclipse or WebSphere WorkBench based platforms, or applications hosted in one of these platforms. The enabler scans your hard drive(s) looking for any installed versions of Eclipse or WebSphere WorkBench.

If your Eclipse shell is not enabled, the Recording Monitor is blank when you try to record against it. For this reason, leave the Recording Monitor in view while recording. If you see this symptom, you need to run the enabler.

**Enabling SAP GUI**

To use Rational Functional Tester to test SAP applications, you need to enable the SAP GUI client. You can enable the SAP GUI client in the SAP GUI tab only for the Windows operating system. The SAP GUI tab is not present in Linux®. Also, you need to ensure that you have administrator privileges so that you can use the SAP GUI enabler.

If you do not have the supported version of SAP GUI client in the Windows operating system, the SAP GUI tab is disabled.

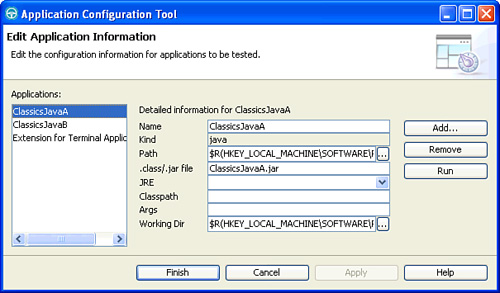
**Configuring Applications for Testing**

Rational Functional Tester enables you to configure your applications for testing, so you can start an application under a test within Rational Functional Tester. This method is simple to configure and does not require knowledge of the scripting language used in Rational Functional Tester. This is the best practice because it makes playing back the tests more reliable.

You configure the application for testing by opening the Application Configuration Tool. This is performed either from Rational Functional Tester by clicking **Configure > Configure Applications for Testing**, or by clicking the **Edit** button in the Start Application dialog box.

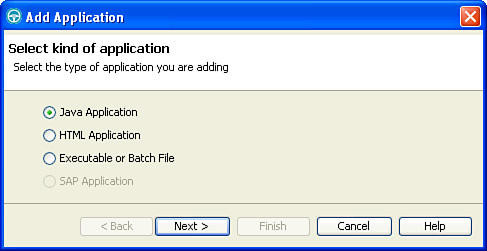
When the Application Configuration Tool is opened, as shown in [Figure 1.6](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig06), click the **Add** button. The Add Application dialog box opens. Here, you add the application configuration, so when you start recording against your application under a test, Rational Functional Tester can open the application during recording.

**Figure 1.6** The Application Configuration Tool



There are four different types of applications that can be added, as shown in [Figure 1.7](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig07):

**Figure 1.7** Add an application



• Java Application—You need to browse to a .class or .jar file type.

• HTML Application—You need to browse to either Local or URL. If it is Local, you need to browse to an .html or .htm file. If it is URL, you need to specify a URL address.

• Executable or Batch File—You need to browse to any executable or batch file.

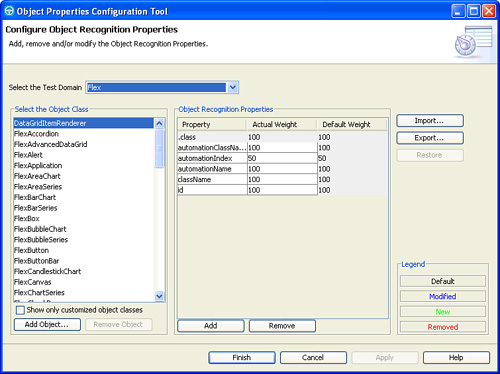
• SAP Application—You need to select SAP executable from the drop-down list. You can also browse to the SAP shortcut file with the .sal or .sap extension. You must have SAPGUI installed in your computer to be able to select SAP Application.

**Configuring Object Recognition Properties**

Rational Functional Tester v8.0 introduces a new feature that enables you to configure object recognition properties. This feature uses the Object Properties Configuration Tool to configure the object recognition properties in the customized object library. While recording test scripts, the customized object library file is used as a reference for setting object recognition properties and the property weights in the object map.

The Object Properties Configuration Tool lists the default objects and the customized objects that are used by Rational Functional Tester, as shown in [Figure 1.8](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig08). If the required test object of the application under test is not listed in the Object Properties Configuration Tool, you can add the test object to the object library and customize its recognition properties and weights.

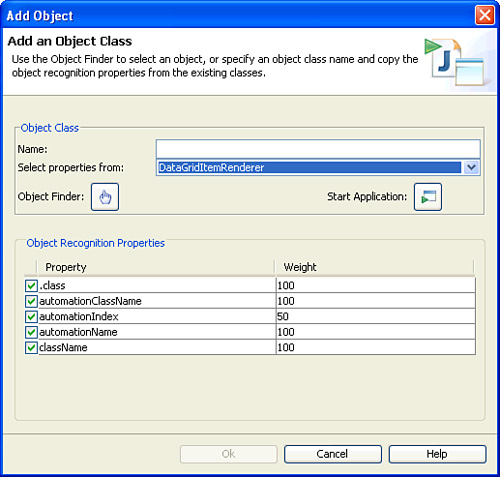
**Figure 1.8** Object Properties Configuration Tool



You can use the Add Object dialog box to add the recognition properties for the existing test objects that are listed in the Object Properties Configuration Tool.

You need to select Test domain from the list, and then click **Add Object**. This opens a new dialog box to add a new test object for the selected domain. There are two options for adding the test object: you can manually specify the test object name and have it inherit the recognition properties from other test objects or you can select the test object using the Object Finder tool, as shown in [Figure 1.9](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig09).

**Figure 1.9** Add Object



**Recording a First Script**

Rational Functional Tester is a flexible tool that enables various ways of working. This section describes the process of a basic recording. Some enhancements are added using wizards. At various points in the scenario, options or additions can be clarified. This is done in a limited way to keep the scenario simple.

**Before Recording**

Before you start recording, you need to take care of several things:

• Be sure the application under test is available including the correct setup of its environment. Doing this results in an expected behavior.

• The application under test must be configured for testing. See the “[Configuring Applications for Testing](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01sec1lev4)” section for more information.

• A Rational Functional Test project is created and available. This is the area where you store your work.

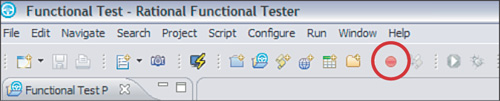
• Before recording, you should already have a test script describing the interactions and the points of interest to verify. This can be a manual test script defined in Rational Quality Manager.

When recording, all interactions are captured and converted into the test script. Experience proves that it is wise to stop or disable any interfering programs, such as messaging programs.

**Recording**

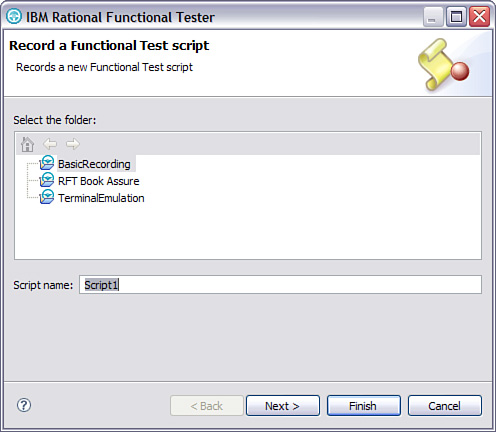
To start recording, click the **Record a Functional Test Script** button in the Functional Test perspective, as shown in [Figure 1.10](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig10).

**Figure 1.10** The Record a Functional Test Script button in the default Rational Functional Tester workbench



A new window opens, as shown in [Figure 1.11](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig11). You need to enter the test script name and the project where it is stored. With the exception of the dollar sign ($) and the underscore (**\_**) spaces and special characters are not allowed for test script names.

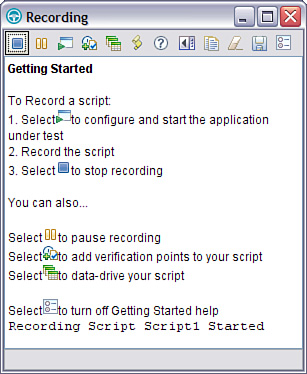
**Figure 1.11** Record a functional test script window where you define the project to store the script and define its name.



When you click **Next**, a second screen opens where some advanced settings can be defined. These settings are discussed in [Chapter 5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05), “[Managing Script Data](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05),” and [Chapter 10](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch10.html#ch10), “[Advanced Scripting with Rational Functional Tester TestObjects](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch10.html#ch10).”

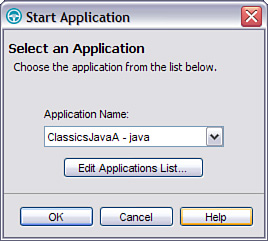
When you click Finish, the Functional Test window disappears and the recording window becomes available, as shown in [Figure 1.12](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig12). From now on, all interactions are recorded! The recording window shows various icons that give access to wizards and functions, such as verification points and data pooling while recording. The recorded interactions are also displayed.

**Figure 1.12** The recording window.



Any interaction against the Recording window is not part of the test script. You first have to start the application under test. Select the **Start Application** icon and then select the application from the drop-down list, as shown in [Figure 1.13](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig13).

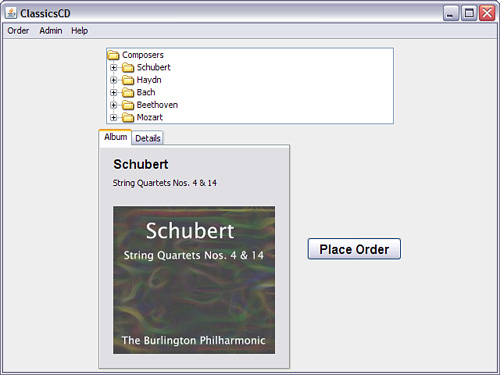
**Figure 1.13** The Start Application window where you can select the application under text. This starts the application and generates the steps in the script.



Selecting OK starts the application. Remember that this action is recorded and it results in an action statement added to the script. This is also visible in the recording window. It is normal for first-time users to perform actions that are not considered part of the intended test script, and as a consequence, results in erroneous recorded steps in the script. All these user errors can be corrected at a later time.

When the application under test is open as shown in [Figure 1.14](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig14), you can perform required test steps. First select composer **Bach**, and then specify the CD selection. Click the **Place Order** button and log in as the default user.

**Figure 1.14** The ClassicsCD application



To validate expected execution of the application under test, the test script must be enhanced with check points called verification points. A *verification point* is a check that the current execution corresponds with your expectations, which is called the *baseline*. A difference between actual and baseline results in a fail status in the execution log. Differences in the consecutive application builds that are not checked with a verification point are not captured and do not result in failures. It is best practice to verify only what makes sense because verification points act as your eyes.

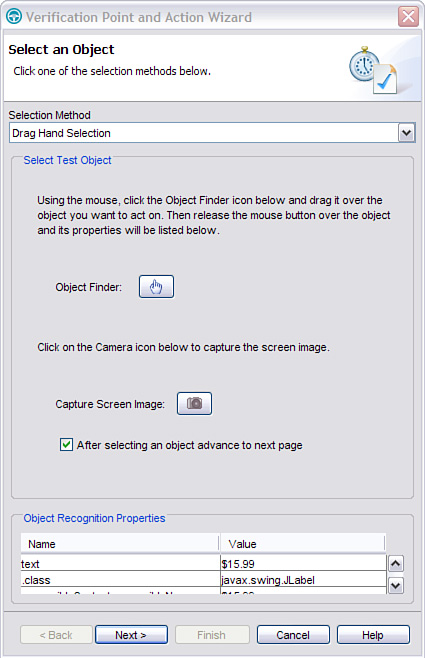
The Verification Point icon shown in [Figure 1.15](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig15) enables access to Verification Point. Let us say that we need to verify that the total price is $15.99. While recording, select the **Verification Point** icon on the recording window.

**Figure 1.15** The Verification Point icon gives access to the verification points.

image

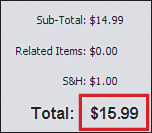
The Verification Point and Action Wizard is displayed as shown in [Figure 1.16](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig16).

**Figure 1.16** The Verification Point and Action Wizard gives you the options to identify the object to be verified.



Drag the hand icon over the $15.99 value. As a preselection, a red square is drawn around the object, as shown in [Figure 1.17](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig17). When you release the cursor, this object is selected.

**Figure 1.17** When you drag the hand icon to objects, Rational Functional Tester provides a preselection for easy identification.



When you release the cursor, the properties of the object selected become visible at the bottom of the wizard. You can validate that you have selected the correct object. Click the **Next** button to advance to the next window. In this window, you define what kind of verification point has to be created; following are the available options:

• Data Verification—Use this for validating the actual data in an object.

• Properties Verification—Use this for validating one or more properties of an object (for example, if it is selected or it is color).

• Get a Specific Property Value—Use this to get a specific property value into a Java variable.

• Wait for Selected Test Object—Rational Functional Tester waits until this object becomes available. Use this as an intelligent mechanism to synchronize with the application under test.

• Perform Image Verification Point—A graphical verification point.

This scenario uses the Data Verification Point. Click **Next**. In the next screen, you can give an appropriate name and influence the default wait for settings. Click **Next**. You can see data here and make modifications if necessary. Click **Finish**. The Data Verification Point is created and inserted as code in the program. You can continue recording the interactions. While recording, you can add various verification points.

After closing the application under test, you have to stop recording by clicking the **Stop Recording** button in the Recorder window. When it is selected, Rational Functional Tester’s main screen displays and test script is created.

**After Recording**

After recording, you can improve the recording by:

• Adding comments where possible. Any tester should be able to read test script and understand the logic.

• Correcting the user’s mistakes, which were recorded and converted into statements.

• Correcting the actions by removing the recorded errors and backspaces.

Note

Data-driven testing is not covered in this simple recording. It is covered in more detail in [Chapter 5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05).

**Validating Execution**

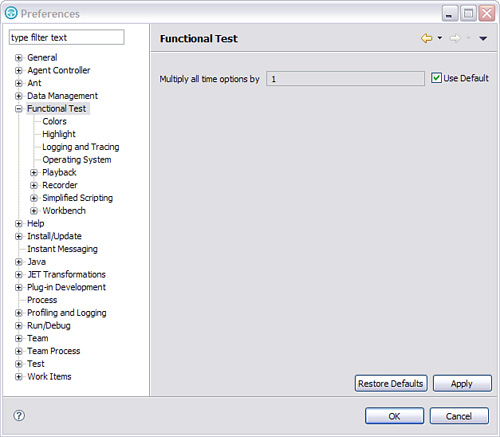
A first validation of correctness must be done by executing the test script against the environment where it was recorded. Keep in mind that not only the application is under test, but so is the test environment, which should be reset to its original state. For example, a double creation of the same customer probably results in an execution error.

**Timing Issues**

It is common for an application to be slower than the Rational Functional Tester expects. For example, an interaction with a web page might be hindered by slow network traffic. This results in a problematic execution. In this case, you have to slow down the execution. Several options are available:

• Get an overall slowdown using the Rational Functional Test parameter shown in [Figure 1.18](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig18).

**Figure 1.18** The overall slowdown parameter for Rational Functional Tester; 30 is roughly 1 interaction per second. Reset it again when running in production.



• Add hard sleep statements:

sleep(2.0);

• Add wait-for-existence:

ObjectInApplication().waitForExistence();

• Lengthen the wait-for parameters in waitForExistence or VerificationPoints:

ObjectInApplication().performTest(ObjectSubmit\_textVP(), 2.0, 40.0);

It is normal to have these kinds of troubles. The default settings will typically work, but which application is normal?

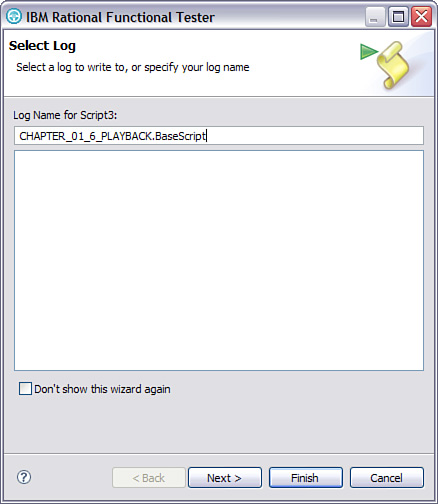
**Playing Back a Script and Viewing Results**

You learn about playing back a test script and viewing test results in subsequent sections. This section provides basic playback and behavior information based on normal settings. Only an overview is provided here.

**Playback**

It is possible to detect differences between what you see in the current application build and the expected result by playing back a test script. To start playback of a script, click the **Run Functional Test Script** button in the menu or right-click a specific test script, and then select **Run**. You are prompted to specify a log file name, as shown in [Figure 1.19](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig19).

**Figure 1.19** The define log window



You can define the name of a test log in the Select Log window. After selecting Finish, the execution begins. Because Rational Functional Tester uses a mouse and keyboard, it is impossible to do work in parallel during execution, or lock the computer.

The progress of the playback can be followed on a screen, and anomalies can be noted. In the playback monitor, you can see which statement Rational Functional Tester executes. Normally, Rational Functional Tester waits for objects to display or to become active.

When the execution ends, Rational Functional Tester returns to its normal state and shows a log file. If the HTML log type is selected, the web browser displays the execution log file. Again, assume that you have default settings active for Rational Functional Tester.

**View Results**

This section discusses the analysis of the log file in the HTML variant, which is the default setting. After execution of a test script, the browser shows a log file. You can also double-click a log file in Rational Functional Tester to view it. HTML log file shows you three types of information:

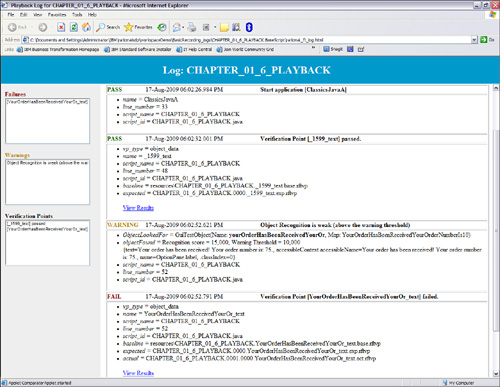
• Failures

• Warnings

• Verification Points

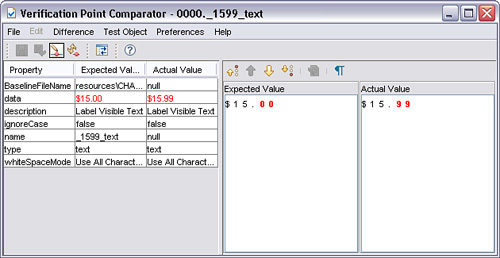
You can select any of these options to quickly show you more detail in the main window, as shown in [Figure 1.20](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig20).

**Figure 1.20** An example of a log file in HTML format



In the case of a failing verification point, you can view the difference between the expected and actual by activating the Verification Point Comparator as shown in [Figure 1.21](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig21), and by selecting View Results of each verification point.

**Figure 1.21** Verification Point Comparator



When a verification point fails, you can view the differences between expected and actual. The baseline can be updated. With the Verification Point Comparator, you can update the baseline with the Replace Baseline with actual result option. It is also possible to start the Verification Point Comparator directly from the logs in Rational Functional Tester.

**Script Editing and Augmentation**

There are many reasons why you should edit and augment a test in Rational Functional Tester. You edit a test to:

• Correct an error or unintended behavior in the test

• Update tests to work with newer application builds

• Separate longer tests into smaller modular tests

• Integrate tests with other automated tests

• Verify functionality or other system requirements

• Associate test data with a test

• Modify or manipulate playback timing

• Modify or control test flow

• Add logging and reporting of test results

• Improve test readability and reuse

**How to Correct or Update a Test Script**

The most frequent type of editing you will probably perform to a test script is fixing or updating. After you finish reading this book and start employing the best test script development practices, these corrections and updates should be short and simple. The two general steps in this activity are removing unwanted test script lines and adding new lines. This section does not go into the details of debugging, but it does describe the general editing steps for doing this.

**Removing Lines from a Test Script**

You can remove unwanted lines of a test script by deleting the lines or by commenting them out (making them into comments). You should begin with the latter because there is always a chance that you might need to restore the original lines. You can comment out multiple lines of a test script as follows:

**1.** Select the multiple lines of script. This assumes that you know which lines of the test you want to disable.

**2.** Comment the lines: For Java in Eclipse, choose the menu **Script > Toggle Comment** or press Ctrl+/. For Visual Studio, choose the menu **Edit > Advanced > Comment Selection** or press Ctrl+K, and then press Ctrl+C.

**Adding Lines to a Test Script**

You can add new lines to a test script by inserting a recording into an existing script or by manually adding lines. Refer to the section, “[How to Manually Add Script Code](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01sec2lev26),” for an explanation of manually adding code to a test script.

It is typically easier to add lines with the recorder. This is easy to do, although you have to ensure that the newly recorded steps flow correctly with the existing recorded steps. You can record new lines into an existing script as follows:

**1.** Get the application under test to the initial state for the new recording. You can play back the test script in debug mode, breaking (pausing) at the spot where you want to add or rerecord steps.

**2.** Carefully position the cursor on a blank line in the test script where you want to add new steps.

**3.** Select **Script > Insert Recording** from the menu, or click the **Insert Recording into Active Functional Test Script** button from a toolbar. You immediately go into recording mode.

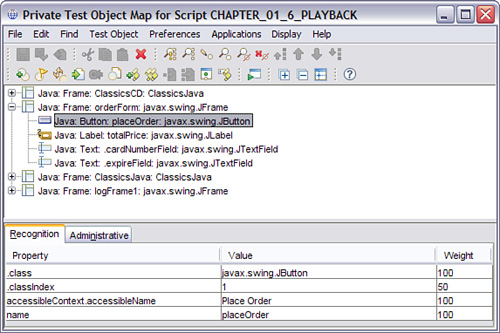
**4.** Click the **Stop** button to finish the recording.

Just as you must ensure the starting point of the new recording is carefully set, you must also ensure that the point that you stop recording flows correctly into the next steps of the test script.

**How to Use Test Object Maps**

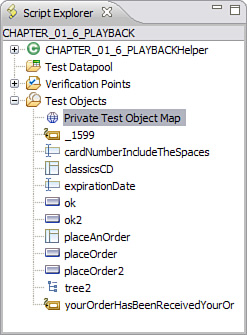
The next most frequent type of editing you are likely do to on a test is update and modify test object maps. A test object map is normally created when you record a new test script, as shown in [Figure 1.22](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig22). You can also create an object map independently from script recording. Every test script has a test object map to use, and every test object map needs a test script to have a purpose.

**Figure 1.22** Editing the test object map



Each test script also contains a list of test objects, visible in the script explorer. This is only a subset of all test objects, as shown in [Figure 1.23](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig23). The list contains only the objects required for this particular test script.

**Figure 1.23** Script explorer test objects



The most common kind of editing that you perform on a test object map is:

• Adding new objects

• Updating objects for a newer version of the application under test

• Modifying object recognition properties

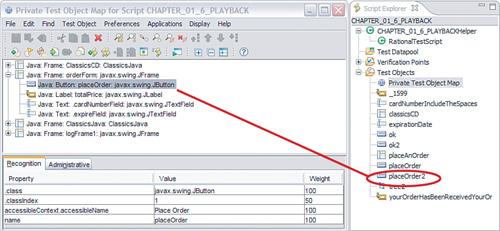
Test objects are explained in greater detail in [Chapter 3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch03.html#ch03), “[General Script Enhancements](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch03.html#ch03),” and [Chapter 9](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html#ch09), “[Advanced Rational Functional Tester Object Map Topics](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html#ch09)”; this section explains only the basics of editing test object maps.

Test object maps have a separate window where you can view, edit, and manage the objects used by test script. The test script itself contains a reference of an object and the action that is performed on it. The following is an example of a line of test script in Java that references a test object named placeOrder.

placeOrder().click();

If you want to learn more about this test object, you can open it from the Script Explorer, which opens the test object map and highlights the object as shown in [Figure 1.24](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig24).

**Figure 1.24** Opening a test object from the script



**Private Versus Shared**

There are two types of test object maps: private and shared. The only difference between the two is that a private map is associated exclusively to one test script, whereas a shared object map can be used by many test scripts. You can open a private test object map from the Script Explorer. You can open a shared test object map from the Project Explorer or from the Script Explorer. The test object map editor and almost all editing capabilities are the same for both private and shared test object maps.

You can create a shared test object map from a private map, and then associate the original test script with the new shared map, changing the private map into a shared one. You can also merge objects from one shared map into another, combining and reducing the number of shared maps. You cannot revert a shared object map back into a private map, but you can merge objects from a shared map into an existing private map.

Ultimately, you need to have primarily shared test object maps instead of private object maps. As a general rule, you should reduce the overall number of different maps.

**Adding, Modifying, and Deleting Test Objects**

Over time, the objects in the application under test that your test scripts interact with will change. You therefore need to add, modify, and delete test objects in Rational Functional Tester. These changes occur at two levels: in the map containing the test objects and in the test scripts that reference the test objects. The test object map is the primary storage location for a test object. This is where you ultimately maintain the test objects and their current recognition properties.

**Adding Test Objects**

You can add a new test object to a test script, as shown in [Figure 1.25](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig25), which adds it to the associated test object map as follows:

**Figure 1.25** Insert New Test Object toolbar button

image

**1.** Get the application under test to the point with the object (graphical or other interface object) that you need to add to the test.

**2.** Position the cursor to a blank line in the test script where you want to add the reference to the new object. This is typically an action on the object, such as clicking or selecting from the object.

**3.** Select the menu **Script > Insert Test Object**, which opens the **Insert a GUI Object into the Object Map** dialog box. This is essentially the same as going into recording mode, except that you do not see the recorder monitor.

**4.** Use the Object Finder to select the object you want to add, and then click **Finish**.

Selecting an Object

Alternately, you could use the Test Object Browser to select the object you want to add. Refer to the product Help documentation for more information on using the Test Object Browser.

As a consequence, this adds the object to the test object map and to the list of test objects for the script, and it will add a reference to the object in the test script where you positioned your cursor. The initial object reference in the script will not be complete since it will not contain any operation (for example, a click action). You can either manually add the desired operation, assuming that it works with the test procedure recorded in the script, or you can simply comment or delete the line and add some actions for the object at a later time.

**Modifying Test Objects**

You can modify a test object by double-clicking on an object in the script explorer, which will open the test object map and highlight the object. If you want to modify an object that is not in the list of test objects in the script explorer, then you can open the test object map and either browse or search to find the object. There are a number of reasons you might modify a test object, most of which are described in [Chapter 9](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html#ch09).

One kind of modification that you can make to a test object that does not require opening the test object map is renaming. You can rename an object directly from the script explorer, as explained in this chapter. Note that you can rename objects from the test object map as well.

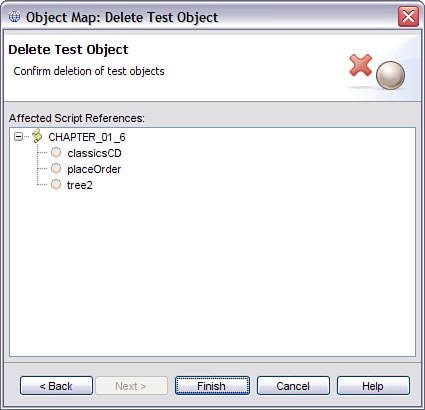
**Deleting Test Objects**

There are two reasons why you might want to delete a test object. First, you may want to remove an object from a test script but leave it in the test object map. You might do this if the object map is shared and the object is used by other test scripts, or you might simply want to leave the object in the map in case you need to add it back into the script at a later time. The second reason you would delete a test object is when you really know that it is no longer needed by any script and you want to delete it from the test object map.

For the first case, when you simply want to remove an object from a script, you can delete an object from the list of test objects in the script explorer. This will not affect the test object map or any other scripts that may use the same map. This also will not remove the reference to the object in the script; you will have to delete or comment the line of code referencing the deleted object yourself. Rational Functional Tester will automatically indicate an error which makes it easier to clean up the script. If you comment out the lines then it will be easier to add it back again later, if needed.

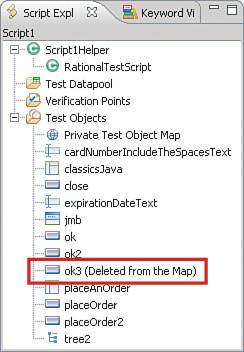
For the second case, when you want to completely delete the object from a test object map, you can open the map and delete the test object. When you do this, Rational Functional Tester will run a short wizard to help ensure you are not deleting something that you need. The first step simply shows the name and recognition properties of the object. The second step, as shown in [Figure 1.26](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig26), shows all of the test scripts that will be affected by the deletion.

**Figure 1.26** Deleting a test object from the map



If you realize that you do not want to delete the object you can cancel. Otherwise clicking on **Finish** will delete the object from the map and all references to the object. Similar to deleting an object from the script explorer, this will *not* delete the line of code in the script that references the object but it will be reflected in object map as shown in [Figure 1.27](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig27).

**Figure 1.27** Object in script deleted from the map



**How to Add Verification Points to a Test Script**

Verification points are what make test script a test as they are primarily what provide the pass or fail results. You will find that it is generally easier to add verification points when you first record a script. However, you might want to first record the user scenarios and steps and then add verification points later. You might also realize additional or more effective verification points after recording the script. In these cases, you can add verification points as shown in [Figure 1.28](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig28) using following steps:

**1.** Get the application under test to the appropriate point for what you want to check. You might want to play back the test script in debug mode, breaking (pausing) at the spot where you want to add your verification.

**2.** Position the cursor on a blank line in the test script where you want to add the verification point.

**3.** Select the menu **Script > Insert Verification Point**, which will open the Verification Point wizard. This is essentially the same as going into recording mode except that you will not see the recorder monitor.

**4.** Complete the wizard to create your verification point. You can find more about the verification point wizard in the online Help.

**5.** Click on **Finish** to return to the test script.

**Figure 1.28** Insert New Verification toolbar button

image

Choosing a verification point

Your choice of verification points should not be arbitrary or chosen on the fly. You should always determine the best way to validate the test case or requirement that the test is implementing. You also need to consider the possibility of errors in proper verification across all test environments and conditions.

**Datapools**

You often record tests using specific values, for input and expected values, which become hard-coded (literals) into the script. Even if you choose certain test data to be variable using the data driven test wizard, you may not realize other hard-coded values that will later need to be changed. You can change static test data in a script into dynamic values by adding datapool variables. You can use datapools for both test input and for expected values (verification points). Datapools are explained in greater detail in [Chapter 5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch05.html#ch05) but we explain the basics of editing tests to include datapools.

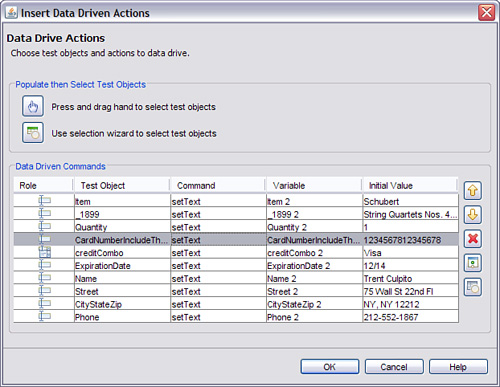
You can add a data driven code (commands that input test data from a datapool) using the Insert Data Driven Actions wizard as follows:

**1.** Get the application under test to the appropriate point where the test data should be added. You might want to play back the test script in debug mode, breaking (pausing) at the data input form or dialog.

**2.** Position the cursor on a blank line in the test script where you want to add the data driven commands.

**3.** Select the menu **Script > Insert Data Driven Commands**, which will open the Data Driven Commands wizard as shown in [Figure 1.29](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig29). This is essentially the same as going into recording mode except that you will not see the recorder monitor.

**Figure 1.29** Insert data driven actions wizard



**4.** Complete the wizard to select the data input objects. You can find more about the Data Driven Commands wizard in the online Help.

If you had not already created or added a datapool for the test script, then a new one will be created. Note that this wizard will add new lines of code to set (input) datapool values to the test object selected in the wizard. If you had already recorded typing or selecting values then the script will set the values twice. In this case, you can either delete the redundant script lines, or you can replace the literal test input values as described in the following paragraphs.

If you already have a shared datapool then you can add it to your test script as follows:

**1.** Right-click on the Test Datapool folder in the Script Explorer and select **Associate with Datapool.**

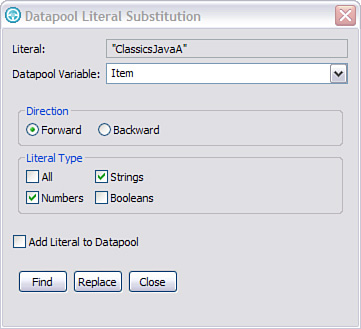
**2.** Select one datapool, and then click **OK**. You can then see the datapool listed in the Script Explorer. You can associate only one datapool with a test script.

At this point, you have associated only the datapool with the test script. You need to replace the literal strings (hard-coded values) with values from the datapool. You do this as follows:

**1.** Select the menu **Script > Find Literals and Replace with Datapool Reference**. Script menu selection Find Literals and Replace with Datapool will be enabled if you have a script open that already has a datapool associated with it. Otherwise, this menu line is disabled.

**2.** In the Datapool Literal Substitution window, as shown in [Figure 1.30](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig30), select the variable (column) you want to use from the Datapool Variable drop-down list.

**Figure 1.30** Replacing literals with datapool values



**3.** Click **Find** repeatedly until the correct value is highlighted in the script and then click the **Replace** button.

**4.** Repeat step 2 until you have replaced all occurrences in the script.

**5.** You can select another Datapool variable from the drop-down list and repeat steps 2 through 3 for other values.

**6.** Click **Close** when you have finished replacing values.

You might already have a datapool with some substitutions for script values, but realize you need to replace additional literal values. You can repeat steps 3 through 6 to add more datapool substitutions to your test script at any time.

**How to Manually Add Script Code**

So far, we have discussed different ways to add recorded lines and test objects, verification points, and datapool commands to a test script using wizards and other recording or capture techniques. If you are familiar with the test script syntax, then you might find it useful to manually add these things directly to the script.

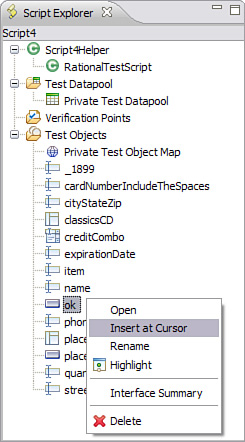
**Adding Test Steps**

If you have already captured test objects in a map, you can add them to your script along with actions for the test to perform as shown in [Figure 1.31](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig31), or with verification points to check. You do this as follows:

**1.** Position the cursor on a blank line in the script where you want to add the new test step.

**2.** Right-click either a verification point or a test object in the Script Explorer and select **Insert at Cursor**.

**Figure 1.31** Adding an existing test element to the script



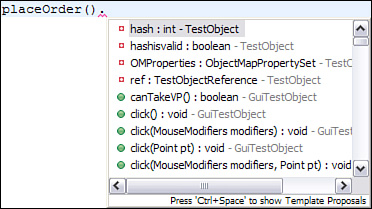
You now have an incomplete reference to either a test object or a verification point in the script. If you add an object, you need to then select an operation (test action) to perform. If you add a verification point, you need an object and not just any object but one capable of returning the expected value. Complete the test script statement for either a test object or verification point:

• For a test object, select an operation from the drop-down list of methods. For example, select click() to make the script click on the test object during test playback. This action is added *after* the object reference.

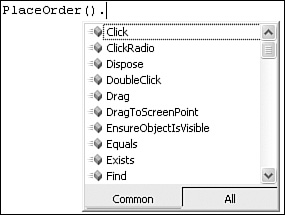
• For a verification point, you must add a test object reference. You can either type or copy the object name, or you can add it from the Script Explorer using **Insert at Cursor**. This object must be added *before* the performTest operation.

With both Java and Visual Basic .NET scripting, if you position your cursor at the end of a test object call (class), just after the parenthesis, and type a period, you will see the list of methods available for that class. If you add the object from the Script Explorer, you might have to press Backspace over the period and retype it. If you then select the method from the list, the editor adds the operation to your script. [Figures 1.32](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig32) and [1.33](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig33) show this for both Eclipse and Visual Studio.

**Figure 1.32** Selecting an operation for a test object in Java



**Figure 1.33** Selecting an operation for a test object in Visual Basic .NET



For Java, you also have to type the ending semi-colon yourself.

**Adding Programming Code to Your Tests**

Rational Functional Tester scripts are implemented in either Java or Visual Basic .NET and are in fact just programs with specific testing functions. Therefore, in addition to adding test steps as described in the previous section, you can also add virtually any programming devices or functions that you would develop for any other Java or Visual Basic program. This includes not only simple looping or conditional constructs, but also calls to more elaborate programming classes. The constraints to this are that the test script must be an extension of com.rational.test.ft.script.RationalTestScript and they must be executed from Rational Functional Tester execution mechanisms.

If you add more programming to your test scripts, you should take advantage of the many development features of the test script editor being used, either Eclipse for Java or Visual Studio for Visual Basic .NET. [Chapter 12](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch12.html#ch12), “[Extending Rational Functional Tester with External Libraries](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch12.html#ch12),” provides Java examples of open-source solutions.

**Other Script Enhancements**

So far, we have discussed several ways to edit test scripts to modify, complete, enhance, or extend their capabilities. Another purpose of editing tests is to improve their readability and potential reuse. This is done by adding comments and descriptions, naming or renaming test elements, and possibly restructuring test scripts into smaller modular tests.

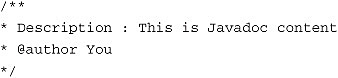
**Comments and Descriptions**

In a given testing effort, you create many tests, object maps, datapools, and other test elements. There are most likely be other people who have to use or reference these same test artifacts. Comments and descriptions should be added to test artifacts and elements to explain their purposes, usages, and any other relevant information. This makes it much easier for someone else other than the test’s creator to understand. This also increases the value of the tests as software development artifacts.

You add comments directly into the test scripts. You can add as many as you like without affecting the execution, and in general, the more, the better. You can add comments during recording using the Script Support Functions, or at any time after recording. Here are examples of comments in each scripting language:

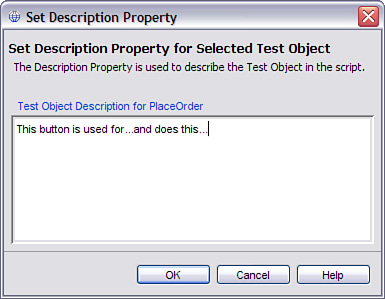
// This is a comment in Java  
' This is a comment in Visual Basic

If you are using Rational Functional Tester with Java scripting, then you can also use Javadoc documentation in your test scripts. Some of this is generated automatically at the beginning of each test script in Java, and you can add more text or tags if you need. More information on Javadoc can be found in the Rational Functional Tester Help.



You can add *descriptions* for certain test elements including test scripts, test objects, and verification points. Descriptions for test scripts are simply Javadoc comments. You can add a description for test objects by opening the object map, selecting an object, and then selecting **Test Object > Description Property** from the menu, as shown in [Figure 1.34](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig34).

**Figure 1.34** Adding a description for a test object



You can add a description for a verification point by opening the verification point and editing the description property, as shown in [Figure 1.35](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig35).

**Figure 1.35** Adding a description for a verification point



**Naming and Reuse**

In addition to adding comments and descriptions, you can improve the readability of a test by renaming test elements to more accurately reflect their meaning or purpose. This is perhaps most important for verification points because you interpret test results largely from these. You have a much harder time understanding a test log or report that has a failure on \_1695Text than one with a failure on OrderTotal. You might also consider renaming test objects, datapool variables (columns), and test scripts.

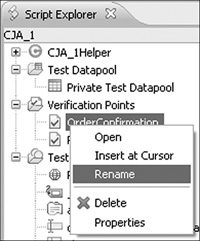
Names for Your Test Elements

Although this discussion is about renaming things, the best time to name your test elements, especially verification points, is when you first record or develop your test.

**Renaming Objects and Verification Points**

You can rename test objects and verification points by right-clicking the item in the Script Explorer and selecting **Rename**, as shown in [Figure 1.36](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig36). This automatically renames the reference to the object or verification point in the test script.

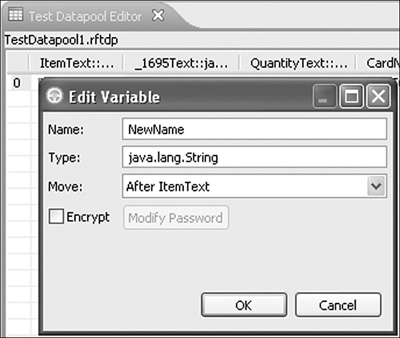
**Figure 1.36** Renaming script elements in the Script Explorer



**Renaming Datapool Variables**

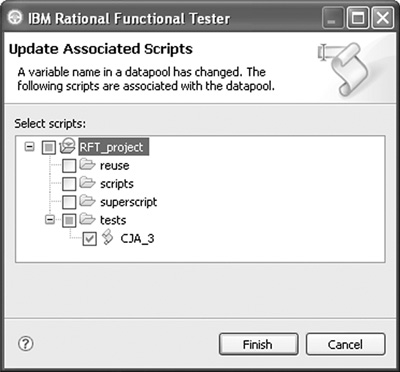
You may need to rename datapool variables, especially if they are generated by the test data wizard, which copies the test object names. The variables (columns names) are used to reference the values in the script, and the name should reflect the real value or purpose. You can rename a datapool variable by opening the datapool (or the script for a private datapool), clicking on the variable name (column header), and entering a new name for the variable as shown in [Figure 1.37](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig37). This automatically renames the reference to the datapool variable in the test scripts.

**Figure 1.37** Renaming datapool columns



If you rename a variable in a shared datapool, the Rational Functional Tester automatically lets you know which scripts are updated with the new name as shown in [Figure 1.38](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig38).

**Figure 1.38** Updating all scripts with a new datapool variable name



**Renaming Scripts**

You might want to rename a test script to better reflect its function or purpose or to comply with project naming conventions. You can do this by right-clicking the item in the Project Explorer and selecting **Rename**. This will automatically rename all the hidden files associated with the test script, such as the helper and verification point files.

**ScriptAssure**

This section describes the integrated technique called ScriptAssure®, which is the recognition algorithm.

**Property Value Weights**

The success of an automated test is highly dependent on the robustness of the test script. Small changes often force development to adapt test scripts before running successfully. This reduces productivity because maintenance must be applied to the test scripts for a successful run.

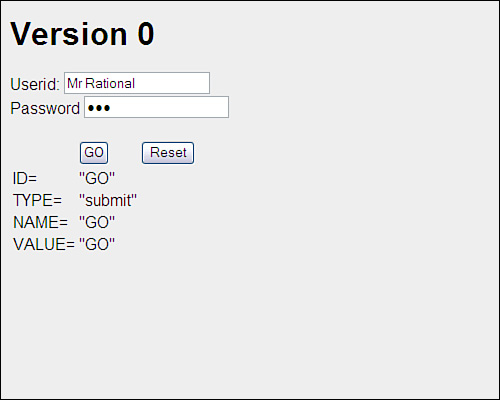
For example, the old record and playback tools recorded based on recording graphical interaction. The actual x, y coordinates of a selection were stored in the script. When a button was moved or a different screen layout was applied to the application, the script became useless. It resulted in a lot of rework and crash maintenance to get scripts running.

Rational Functional Tester recognizes objects in the application under test. This means that objects can be moved around or changed graphically. Recognition is done based on all the properties of the object. Not only the visible label (the OK), but object properties are recorded. When one or more properties changes, Rational Functional Tester keeps running!

**Sample 1: Login**

An example is the button GO in a login screen, as shown in [Figure 1.39](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig39).

**Figure 1.39** The login screen and GO button



The base GO button has the following properties:

• ID = “GO”

• TYPE = “submit”

• NAME = “GO”

• VALUE = “GO”

One can easily find these values in the HTML source of the button (this is true of any programming environment):

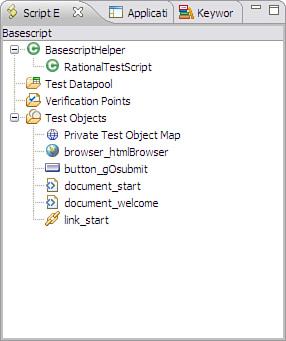
<INPUT ID="GO" TYPE="submit" NAME="GO" VALUE="GO"  
ONCLICK="logincheck(username.value,password.value);">

When recording a script, the script contains the following line:

button\_gOsubmit().click();

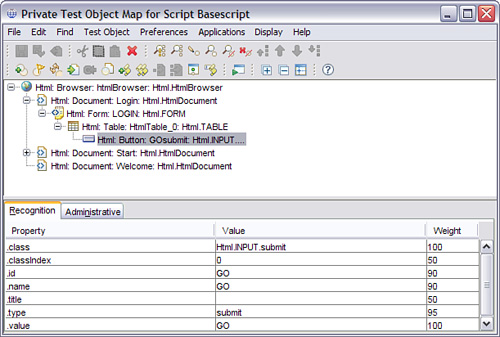
The button\_gOsubmit is the object in the Script Explorer view, as shown in [Figure 1.40](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig40). A click is the action to be performed on that object.

**Figure 1.40** Script, Explorer view



Double-clicking the object in the tree opens the object map; here, you can interrogate the properties of the object, as shown in [Figure 1.41](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig41).

**Figure 1.41** Test Object Map browser

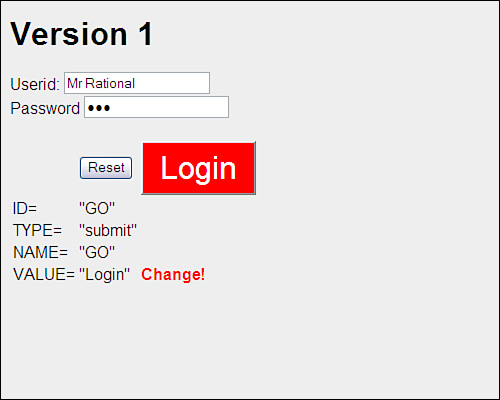


When the script is run to validate the script, it finds the object with all properties matching. The recognition algorithm is as easy as it is powerful; every mismatch results in 100 times the weight. The object with the minimum penalty points is selected.

**One Property Change**

Suppose the visible property GO is changed to Login, as shown in [Figure 1.42](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig42). An automatic test tool that recognizes only object visible labels fails, resulting in the test process stopping and an urgent need for maintenance.

**Figure 1.42** Changing one visible property

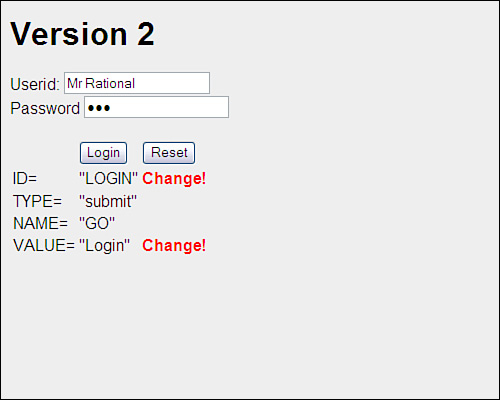


Rational Functional Tester uses the same object or the one with minimal difference. In this case, Rational Functional Tester selects the Login, despite the difference in label, position, and format. The Reset button is not selected because there is a difference in all properties.

**Two Properties Changed**

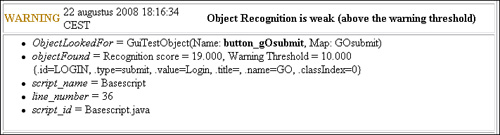
Now with version 2, there are two properties changed, as shown in [Figure 1.43](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig43). Does Rational Functional Tester find the button?

**Figure 1.43** Changing two visible properties



With the default settings, Rational Functional Tester stores a message in the test log, as shown in [Figure 1.44](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig44). The Object Recognition is weak message indicates that Rational Functional Tester has found one object that is comparable. Rational Functional Tester uses this object to continue execution. Additionally, there is a recognition score (failing score) visible that is 19.000. The calculation can be derived by the rule that every mismatch results in 100 times the weight. This example shows a miss at ID (weight 90) and Value (weight 100), resulting in a 19.000. When no differences are found, this recognition score is 0.

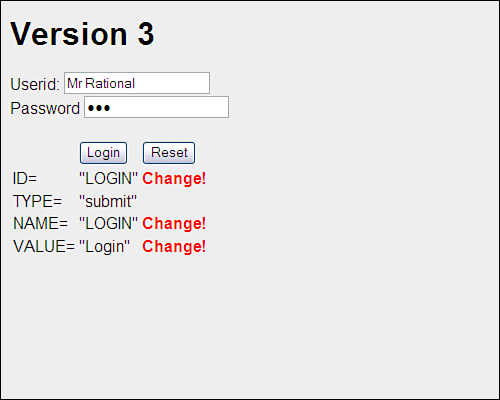
**Figure 1.44** Viewing the message in the test log



**Three Properties Changed**

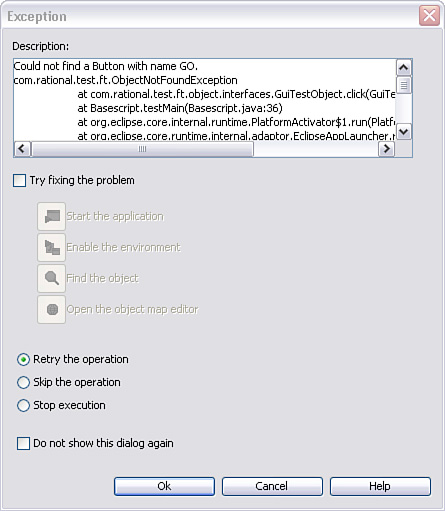
In the next version of the application, there are three properties changed, as shown in [Figure 1.45](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig45).

**Figure 1.45** Changing three visible properties



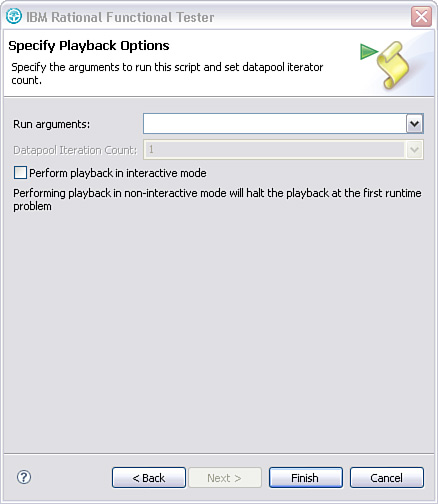
Now the object to be found and the object available in the application differ too much. A window opens to ask the user for advice, as shown in [Figure 1.46](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig46).

**Figure 1.46** Exception window



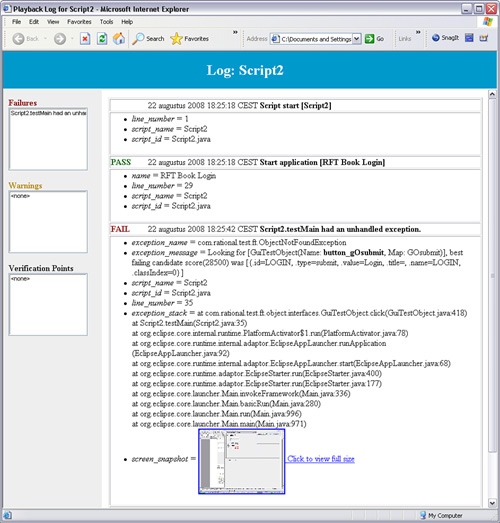
With the object browser, one can update the properties of the object to be found. The Playback Options become available just after starting a script. The previous interactive menu can be suppressed by deselecting the option Perform playback in interactive mode in the second screen of the Playback Options, as shown in [Figure 1.47](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig47).

**Figure 1.47** Specify Playback Options



If you update the properties of the object, you get a similar error message as shown in [Figure 1.48](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig48).

**Figure 1.48** Test log with error message



Rational Functional Tester does find an object candidate, but it has a failing score of 28500, as described in the log. This is above the value Last chance recognition score, which is set by default to 20000. If you increase this value to 50000, the script does find the Login button and provides a warning, as shown in [Figure 1.49](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig49).

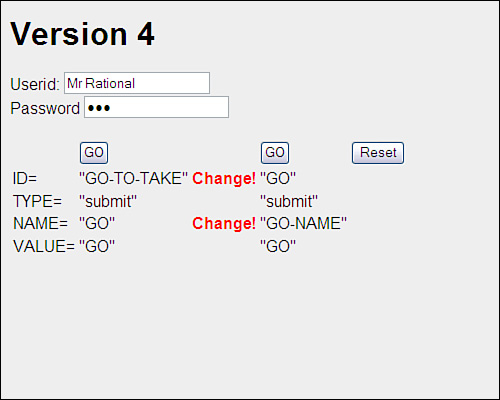
**Figure 1.49** Test log with warning message



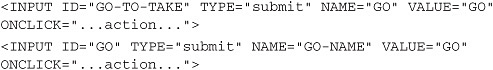
**Sample 2: Two Buttons**

The previous examples clarify the behavior of ScriptAssure. The following example is somewhat more complex while we have two buttons, which are similar as shown in [Figure 1.50](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig50).

**Figure 1.50** Sample 2: two buttons

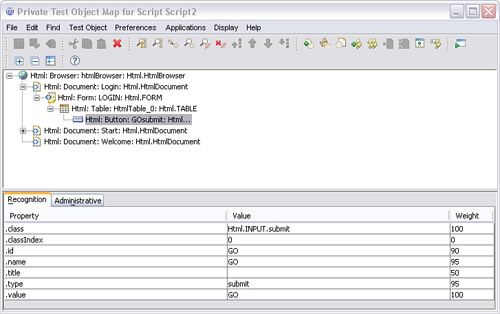


When the script is run, which of the buttons is selected, the “GO” (left) or the “GO” (right) button? Both have one change in a property. A part of the source of this menu is:



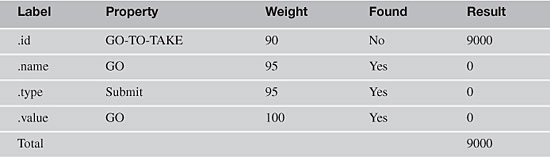
The object to be searched for is defined in the object map, as shown in [Figure 1.51](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig51).

**Figure 1.51** Searched object in the object map

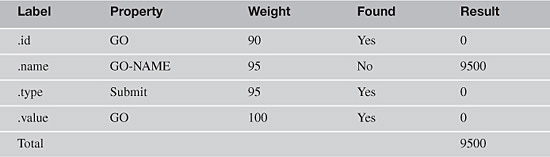


Calculate the penalty points of the “Login” and the “GO” button, as shown in [Table 1.1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01tab01) and [Table 1.2](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01tab02).

**Table 1.1** Table for the First Go Button

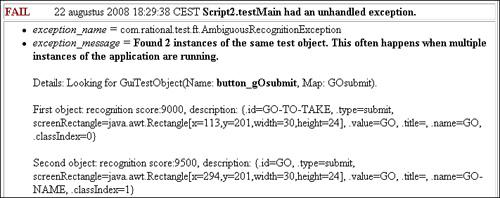


**Table 1.2** Table for a Second Go Button



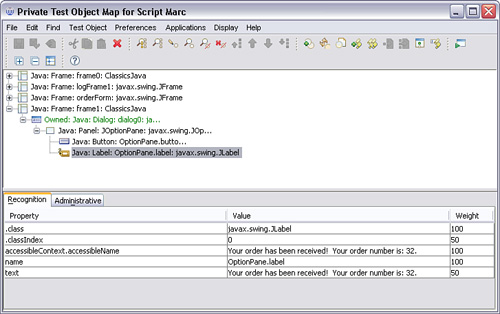
If you replay with default ScriptAssure settings, you get the message AmbiguousRecognitionException, as shown in [Figure 1.52](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig52). This is because the two GOs are much the same.

**Figure 1.52** Log file with two instances of the same test object



When we decrease the Ambiguous recognition scores difference threshold to 200, for example, the script continues. So, the action attached to the GO-TO-TAKE button is used. If you are interested in verifying the properties of the object, the following is created in the object map, as shown in [Figure 1.53](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig53).

**Figure 1.53** Created object in the object map



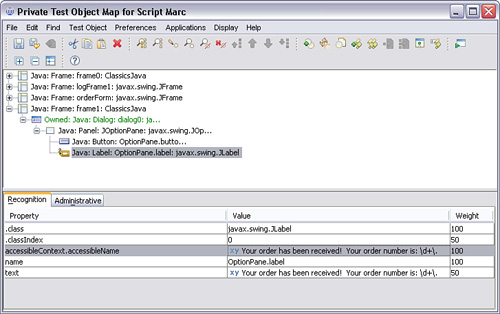
When you execute test script next time, the order number will not be 25. This results in penalty points, as shown in [Figure 1.54](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig54).

**Figure 1.54** Log file displays a weak object recognition



By setting the weight of accessibleContext.accessibleName and text to 0, there is a full match, but the recognition power is weaker. A better approach is to apply a regular expression as a value. This can be created via the contextual menu on the value. In this case, use a decimal definition \d+ as shown in [Figure 1.55](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig55). The point is preceded by a backslash because the point is also a special character. For additional information about regular expressions, refer to [Appendix B](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/app02.html#app02), “[Regular Expressions in Rational Functional Tester](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/app02.html#app02).”

**Figure 1.55** Property value with regular expression set



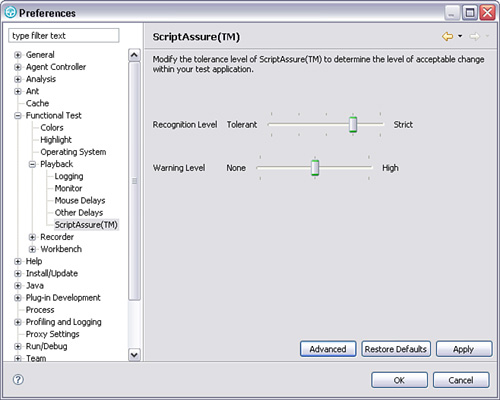
This sample is about the object to be searched for. For the changing value in the verification point, you can use a regular expression in the verification point.

**ScriptAssure Playback Settings**

The recognition and the warning levels can be influenced with settings at **Window > Preferences > Functional Test > Playback > ScriptAssure**.

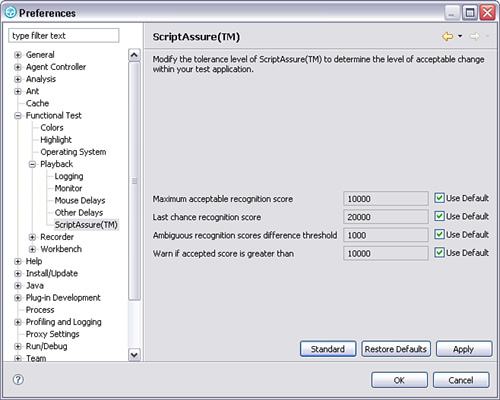
The standard visualization gives two sliders to move in either direction, as shown in [Figure 1.56](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig56). In line with the error messages and the calculation described previously, you can use the advanced visualization.

**Figure 1.56** ScriptAssure preferences



If you click the Advance button, you get what’s shown in [Figure 1.57](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig57).

**Figure 1.57** Advance ScriptAssure preferences



The ScriptAssure Advanced page has the following controls:

• **Maximum acceptable recognition score—**Indicates the maximum score an object can have to be recognized as a candidate. Objects with higher recognition scores are not considered as matches until the time specified in Maximum time to attempt to find Test Object has elapsed.

• **Last chance recognition score—**Indicates the maximum acceptable score an object must have to be recognized as a candidate, if Functional Tester does not find a suitable match after the time specified in Maximum time to attempt to find Test Object has elapsed. Objects with higher recognition scores are not considered.

• **Ambiguous recognition scores difference threshold—**Writes an AmbiguousRecognitionException to the log if the scores of top candidates differ by less than the value specified in this field. If Rational Functional Tester sees two objects as the same, the difference between their scores must be at least this value to prefer one object. You can override the exception by using an event handler in the script.

• **Warn if accepted score is greater than—**Writes a warning to the log if Rational Functional Tester accepts a candidate whose score is greater than or equal to the value in this field.

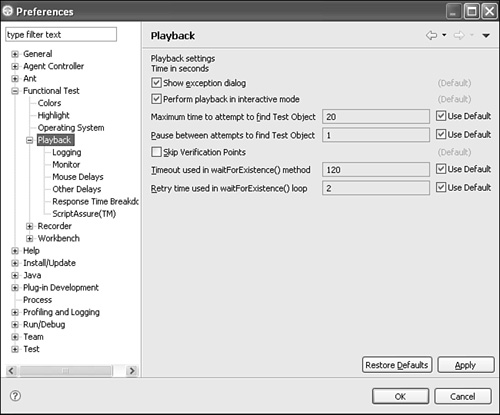
The Maximum time to attempt to find Test Object is defined in the general playback of Rational Functional Tester.

What are interesting settings? Defaults to start with because they work well. If you are in more of a dynamic user interface, you can increase the various values to acceptable levels. If you are doing acceptance testing, you can tighten the values and set the warn if option to 1. You always get a warning when something changes, but Rational Functional Tester continues to run.

**Playback Settings**

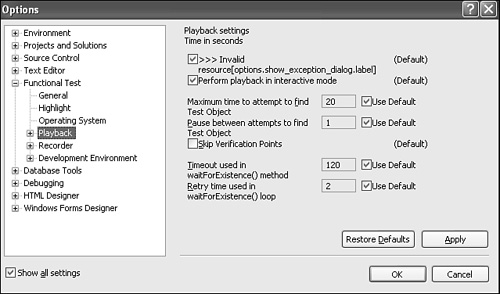
Rational Functional Tester provides a myriad of playback settings, available through its user interface. These settings enable you to control the behavior of how Rational Functional Tester handles such things as delays, logging, and object recognition. You can access the playback settings by selecting **Window > Preferences > Functional Test > Playback**. This launches a window similar to the one in [Figure 1.58](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig58).

**Figure 1.58** Primary playback settings for Eclipse



If you are using Rational Functional Tester VB.NET, you can access the playback settings by selecting **Tools > Options > Functional Test > Playback**. [Figure 1.59](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig59) shows what the resulting Options window should look like when displaying the Playback settings.

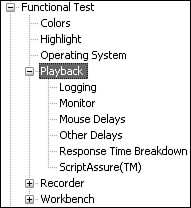
**Figure 1.59** Primary playback settings for .NET Studio



You usually turn to the primary playback settings displayed in [Figure 1.58](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig58) ([Figure 1.59](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig59) for VB.NET) when you need to override Rational Functional Tester’s default synchronization with the application under test (for example, its capability to wait long enough for the necessary GUI objects to render). When Rational Functional Tester isn’t waiting long enough for your application to render a GUI object, you can access the playback settings and increase the Maximum time to attempt to find Test Object setting to more than 20 seconds. This tells Rational Functional Tester to wait longer for the object to render. This is useful when your application tends to have slow days.

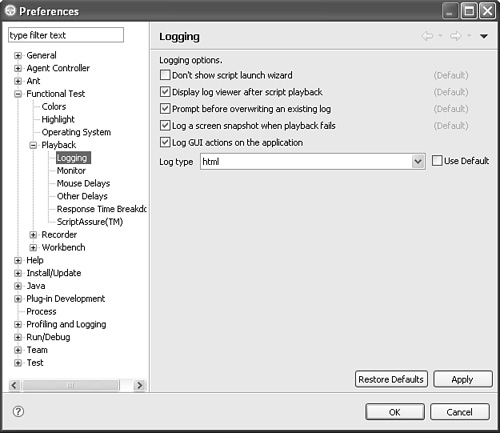
When you further drill down into the playback settings, as shown in [Figure 1.60](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig60), you can control mouse delays, keyboard delays, and Rational Functional Tester’s Script Assure technology. Typically, you should focus mainly on adjusting your logging and Script Assure settings.

**Figure 1.60** Playback settings: subsettings



Selecting the Logging option enables you to choose the type of log to write (for example, text, HTML, none, XML, and so on), and it enables you to specify the level of information to log, outside of what is captured by default. These options are displayed in [Figure 1.61](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig61).

**Figure 1.61** Playback settings: logging



There might be instances where you have to increase your level of logging to include GUI actions performed by your script. This is useful when trying to debug playback issues. You can see the actions that were performed up to the point where your script failed. To accomplish this, simply select the **Log GUI actions on the application** check box. [Figure 1.62](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig62) shows a comparison between two RFT logs. The one on the left shows the default information that gets logged. The one on the right displays the default information along with the added GUI actions, enabling you to see the sequence of events your script performed.

**Figure 1.62** Logs showing the difference between logging your script’s GUI actions and not logging them

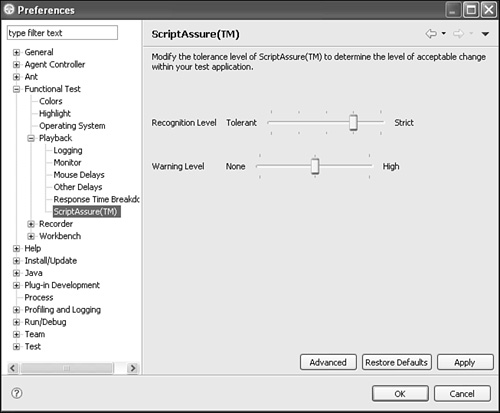


The first log shows the default level of logging. It captures the major events of playback. These are things such as Script start, Start Application, and so on. The second log shows the major events plus it shows the GUI actions that were performed by IBM Rational Functional Tester. These are delineated by phrases such as, “click on the tree,” “click on the button,” and so on. You should choose to add the GUI actions into your log when runtime errors occur in your script. The extra logging helps narrow down where the problem is occurring.

The key to success for any automated test tool is the capability to find GUI objects (for example, buttons, checkboxes, combo boxes, and so on). This must occur *every* time a script plays back. Rational Functional Tester addresses this via its Test Object Map and Script Assure technology, as discussed earlier in this chapter.

Rational Functional Tester’s playback settings provide you with the capability to control the fuzzy matching logic that Script Assure uses for object recognition. This is shown in [Figure 1.63](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01fig63). The Script Assure setting is especially useful when you begin automating early in the GUI development lifecycle. GUI objects and their properties are unstable at this stage. Their internal values can, and often do, change multiple times. You can use the Script Assure playback settings to adjust IBM Rational Functional Tester’s object recognition capabilities to handle this. Sliding the Recognition Level to the left (Tolerant) enables IBM Rational Functional Tester the freedom to find GUI objects, even when a couple of object recognition properties change. Clicking the **Advanced** button enables you to fine tune the fuzzy matching. You also come here to adjust the level of information to display in the test log when GUI object recognition values change. This is helpful for identifying where you need to make updates in your test object map. Warning messages are placed in your log file, citing which objects had recognition values that changed.

**Figure 1.63** Playback settings: Script Assure



This section covered the more widely used playback settings. You might explore some of the other settings found here. Some reasons for doing so include:

• Tweaking a drag-and-drop action and using the mouse up and mouse down options found in the Mouse Delays subsetting.

• Turning the monitor display off during playback and using the Show monitor during playback option found in the Monitor subsetting.

• Turning off the log to test script playback and using the Log type option found in the Logging subsetting.

**Summary**

This chapter provided an overview of major IBM Rational Functional Tester functionalities. It started with a brief introduction to IBM Rational Functional Tester and how it can be installed. It then discussed its preference capabilities, followed by the simple record/playback against application under test. This chapter also explored how IBM Rational Functional Tester finds particular objects and the value of ScriptAssure. This chapter is a great place to learn the basics.

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/fm.html)

[Title Page](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/fm.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html)

[Chapter 2. Storyboard Testing](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html)