**Chapter 2. Storyboard Testing**

**Fariz Saracevic, Lee B. Thomas**

*This chapter explains the Storyboard Testing feature introduced in the 8.1 version of Rational Functional Tester. Storyboard Testing provides all the capabilities of Rational Functional Tester, but displays them in a manner that is easier to adopt and more productive for nontechnical users. The term Simplified Scripting is used interchangeably with Storyboard Testing to identify this new Rational Functional Tester capability. We use the term Storyboard Testing throughout the book to make it simpler for readers.*

*To enable you to easily compare Storyboard Testing to the traditional perspective, this chapter reuses the example from* [*Chapter 1*](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01)*, “*[*Overview of Rational Functional Tester*](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01)*.” By the end of this chapter, you should understand how to record, play back, and edit scripts with Storyboard Testing, and you should know what options are available.*

**Overview of Storyboard Testing**

This section provides an overview of the Storyboard Testing feature. Storyboard Testing is a new feature introduced in Rational Functional Tester version 8.1. Its purpose is to enable automated test creation by testers who might have significant subject matter expertise with application under test but might not have programming skills.

**A Look Back at Test Automation**

Originally, all test automation was done by programming. To use those early tools, testers had to learn a programming language and become proficient at it. This was a barrier for testers who might have had expertise in the subject matter of the business, but who had no training in programming.

Record-and-playback technology enabled nonprogrammers to create automated test scripts and execute them. However, if the application being tested was changed (as one would expect it would be in the course of maintaining it), then the test scripts often needed to be recorded again. Unless the automated script was executed frequently, the savings of a tester’s time through test automation would decrease because maintenance of test automation required almost as much time as executing the tests manually.

The use of wizards simplified some tasks that would otherwise have required rerecording or programming, but for other tasks, the only recourse was to learn the programming language or assign the task to someone who did know the language. Again, the nontechnical user was shut out.

Whether the script was programmed or recorded, the fact that the script was represented as a program required the tester to visualize what the application looked like at each statement of the script, rather than by seeing the interface itself. This also tended to favor those with programming backgrounds because this kind of visualization is an essential part of a programmer’s training.

Overall, test automation required too much skill in programming. Subject matter experts needed an easier way to create test automation, one that would not require them to become coders.

Rational Functional Tester provides this easier way. Storyboard Testing enables nontechnical users to see their scripts in a human language instead of in a programming language.

**Similarities and Differences to** [**Chapter 1**](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html)

This section shows which sections of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) are unchanged when you use Storyboard Testing and which sections of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) do not apply without modification. Although Storyboard Testing gives nontechnical users great power to easily automate tests, when you use it you are creating, editing, and executing the same assets as in the traditional perspective. None of the underlying components of RFT are different; Storyboard Testing is just a new set of views of those components. You can refer to most of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) whether you use Storyboard Testing or you don’t. The following sections explain this.

**Sections That Are Unchanged from** [**Chapter 1**](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html)

These sections at the beginning of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) are unchanged for both the traditional Functional Test Perspective set of views and the new Storyboard Testing views:

• Architecture of Rational Functional Tester

• Installation and Licensing

• Enabling the Environment for Testing

• Configuring Applications for Testing

• Configuring Object Recognition Properties

Similarly, these sections at the end of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) are unchanged for both perspectives:

• ScriptAssure

• Playback Settings

This means that you can install and configure the Rational Functional Tester in the same way, whether you use the Storyboard Testing feature or not. In fact, your Functional Test Project can contain scripts created with the feature and scripts created without the feature. You can edit both types of scripts in the same instance of Rational Functional Tester; there is no need to exit from Rational Functional Tester and restart it. All the integrations with other tools work identically for both types of Rational Functional Tester scripts.

Storyboard Testing works with all the environments you can enable and supports all the applications you can configure for testing. ScriptAssure and the playback settings work identically for both script types.

If you are administering Rational Functional Tester, you can deploy it to all your users in the same way, provided you or your nontechnical users enable the Storyboard Testing preference that is shown in the section “[How to Enable Storyboard Testing](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02sec2lev4)” later in this chapter. If you are a nontechnical user, you do not need any special setup or configuration to use Storyboard Testing. Plus, your Storyboard Testing scripts can use all the features of Rational Functional Tester and can include any extensions created by a Java programmer. Your scripts are executed by Rational Functional Tester just like scripts created with the traditional perspective.

**Sections That Differ from** [**Chapter 1**](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html)

Using Storyboard Testing is different from using the traditional Functional Test perspective set of views for these sections of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01):

• Recording a First Script

• Playing Back a Script and Viewing Results

• Script Editing and Augmentation

This means that you can install and configure the Rational Functional Tester in the same way whether you use the Storyboard Testing feature or not. In fact, your Functional Test Project can contain scripts created with the feature and scripts without the feature. You can edit both types of scripts within the same instance of Rational Functional Tester; there is no need to exit from Rational Functional Tester and restart it. All the integrations with other tools work identically for both types of Rational Functional Tester scripts.

Storyboard Testing works with all the environments you can enable and supports all the applications you can configure for testing. ScriptAssure and the playback settings work identically for both script types.

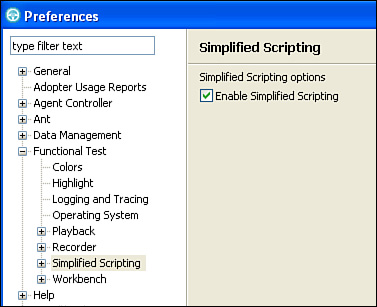
**Enabling Storyboard Testing**

Storyboard Testing is enabled by default in Rational Functional Tester; you do not need to do anything to be able to use it unless it has been disabled in your installed copy of Rational Functional Tester. The next section shows you how to disable and re-enable this feature. The section “[Considerations for Enabling Storyboard Testing](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02sec2lev5)” explains how Rational Functional Tester operates when Storyboard Testing is enabled and disabled.

**How to Enable Storyboard Testing**

To enable Storyboard Testing, first open the Preferences dialog box by clicking **Window > Preferences**. In the dialog box, expand **Functional Test** in the left navigation pane, and then click **Simplified Scripting**, as shown in [Figure 2.1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig01). Finally, click the **Apply** button to save this preference. If you have no other preferences to change at this time, you can click the **OK** button to close the dialog box.

**Figure 2.1** The option in Functional Test preferences to enable Storyboard Testing



**Considerations for Enabling Storyboard Testing**

When you check this preference, from that time forward, all new scripts you create (whether by recording or by editing an empty script) use Storyboard Testing. To return to the traditional perspective, uncheck this preference and all new scripts are recorded and edited in the traditional manner until you change this preference again.

This preference has no effect on scripts that already exist; Rational Functional Tester stores the type of script (Storyboard Testing or traditional) with the script, and opens the appropriate views in the Functional Test Perspective. You can edit Storyboard Testing scripts alongside traditional scripts within the same running instance of Rational Functional Tester and as you switch between the different script types. As you select a different type of script, the Functional Test Perspective changes the views to match the type of script being edited.

The Storyboard Testing feature includes additional preferences you can modify. Those preferences are explained later in this chapter. First, we cover how Storyboard Testing affects recording, playback, editing, and augmentation of scripts.

**Recording a First Script**

Storyboard Testing does not directly affect the recording process. The following sections are identical to the sections in [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01), except explanations have been added to areas where you might notice a difference.

**Before Recording**

Prior to recording, you must have already created a Functional Test Project, enabled your application’s environment within Rational Functional Tester, configured your application for testing, and ensured that the test environment has been configured correctly.

• To create a Functional Test Project, click the menu item **File** > **New** > **Project**, expand **Functional Test**, and then click **Functional Test Project**. Follow the steps of the wizard.

• To enable your application’s environment, click the menu item **Configure** > **Enable Environments for Testing**, and then ensure that your application’s environment is already enabled or add your environment with the help of the dialog.

• When Rational Functional Tester is installed, it enables the environments installed on your computer at that time. You should need to enable only environments that you have added after RFT is installed. For example, if you download Mozilla Firefox after installing RFT, you need to enable Firefox.

• To configure your application for testing by Rational Functional Tester, click the menu item **Configure** > **Configure Applications for Testing**, and if your application is not already configured, click the **Add** button and follow the steps in the wizard.

• To configure the test environment, ensure that your application has been installed correctly and can be executed outside of Rational Functional Tester (as if you were testing manually). Also, ensure that any data, files, or other items that your application uses have been set to the correct state for the test you are recording.

Note

It is advisable to disable instant messages, email programs, and any other applications in your computer’s work environment that might interfere with your recording.

In the following sections, you use the ClassicsJavaA application that is provided with Rational Functional Tester as a sample. ClassicsJavaA does not require you to provide or reset any data or add or delete files to your file system, even if you execute it many times.

When the application under test is open, you can perform the steps as required by your test. You perform the test with these steps:

• Start the application.

• Choose a CD.

• Log in as the default user.

• Enter the required data into the form.

• Verify that the price of the CD is correct. (For this exercise, simply assume that the price is correct. You learn how to capture data. In actual testing, presumably you would have a reference that would enable you to verify the correctness of the displayed data.)

• Place the order.

• Exit the application.

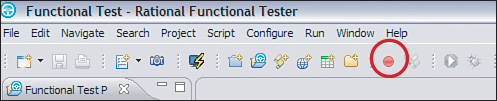
It is normal for first-time users to perform actions that are unintentional and result in erroneous recorded steps in the script. Any user errors can be corrected later, so there is no need to stop recording and start over if you make a mistake.

**Recording**

To start recording, follow these steps:

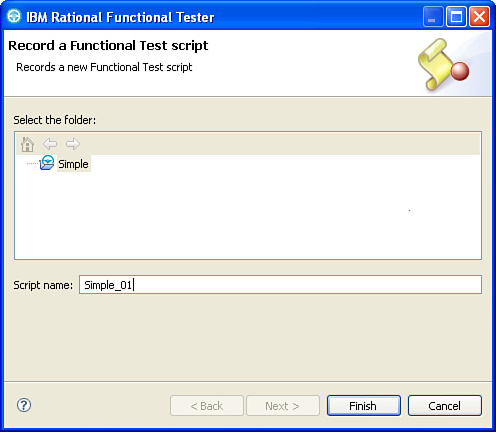
**1.** Click the **Record a Functional Test Script** button, as shown in [Figure 2.2](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig02).

**Figure 2.2** The Record a Functional Test Script button in the default Rational Functional Tester workbench



**2.** A new window opens up, as shown in [Figure 2.3](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig03). Enter the test script name and select the project where it will be stored. The test script name must not contain any spaces or any special characters except **$** and **\_** (the underscore character). For this exercise, you can select any Functional Test Project, and you can use the name of the script as it was created by Rational Functional Tester.

**Figure 2.3** The Record a Functional Test Script window, where you select the project to store the script and define the name of the script



**3.** Click **Finish.** The Rational Functional Tester window minimizes, and the Recording Monitor becomes available, as shown in [Figure 2.4](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig04).

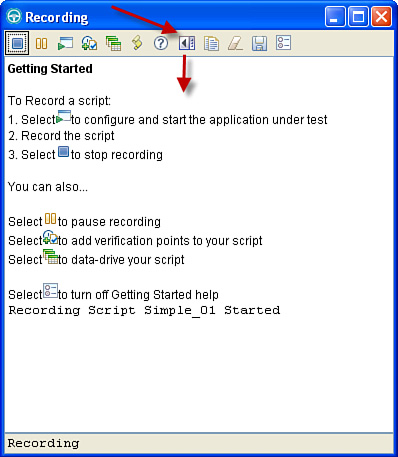
**Figure 2.4** The Recording Monitor. The recorded interactions are shown here. The various icons give access to wizards accessing functions such as verification points.

image

You can interact with the Recording Monitor, and those interactions do not generate statements in the script unless you explicitly press a button in a dialog that is intended to create a statement. By default, the Recording Monitor displays a small row of buttons. You can expand it to display a list of reminders for common recording actions and to display a log of actions you perform.

**4.** Click the **Display Monitor** icon to open the Recording Monitor, as shown in [Figure 2.5](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig05).

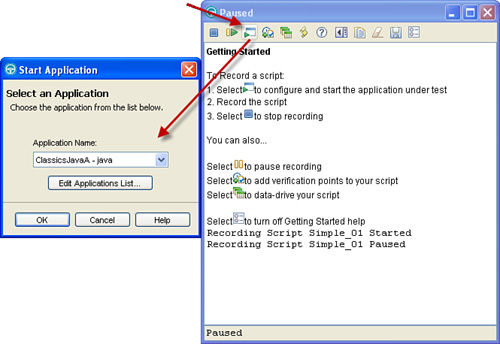
**Figure 2.5** The Start Application window where you can select the application under text. This starts the application and generates an action in your script to start the application.



You use another button on the Recording Monitor to start your application. This causes Rational Functional Tester to start your application every time the script is executed, which eliminates the need for you to open the application yourself. When you record the starting of the application in your script, you should remember to close your application at the end of recording. In addition, prior to executing your script, you must remember to close all copies of your application that you might have open, so that Rational Functional Tester does not need to choose between multiple open copies, including the one started by the script.

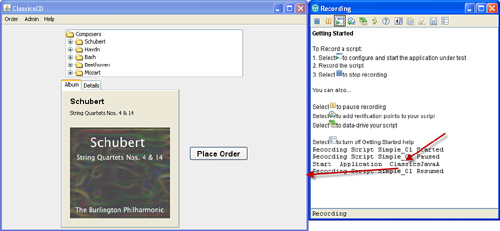
**5.** You now start the application under test. Click the **Start Application** icon and then click the application on the drop-down list, as shown in [Figure 2.6](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig06). For this exercise, click **ClassicsJavaA**.

**Figure 2.6** The Start Application window where you can select the application under text. This starts the application and generates an action in your script to start the application.



**6.** Click **OK** to start the application. This causes Rational Functional Tester to add a statement to the script to start your application. You can see the statement that is in your script by looking at the Recording Monitor; see [Figure 2.7](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig07).

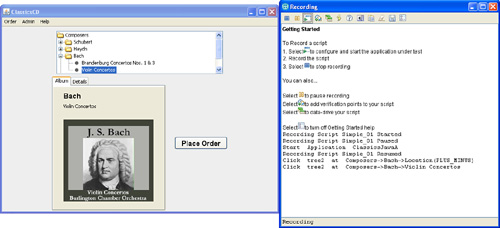
**Figure 2.7** The Recording Monitor dialog where you can see an entry for the action to start the application under test



In the Recording Monitor, you can see the difference between recording in the traditional perspective and recording with Storyboard Testing. Instead of a line from a programming language for starting the application, there is a line in English natural language.

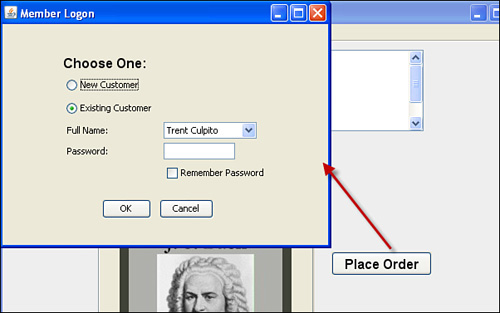
**7.** Click the **+** to the left of Bach to expand the tree, exposing two CD selections for that composer. Next, click **Violin Concertos** to select it. The ClassicsJavaA application and the Recording Monitor should now look like [Figure 2.8](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig08).

**Figure 2.8** The ClassicsJavaA application with the Bach Violin Concerto selected



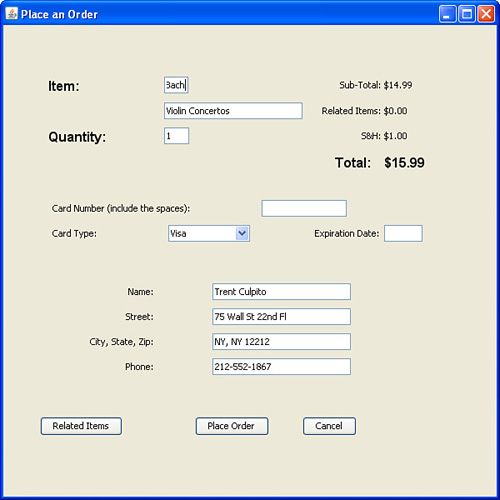
**8.** Click the **Place Order** button. The Member login dialog box should display; see [Figure 2.9](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig09).

**Figure 2.9** The Member Login dialog box



**9.** Log in with the default user by clicking the **OK** button. The Place an Order dialog box should display, as shown in [Figure 2.10](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig10).

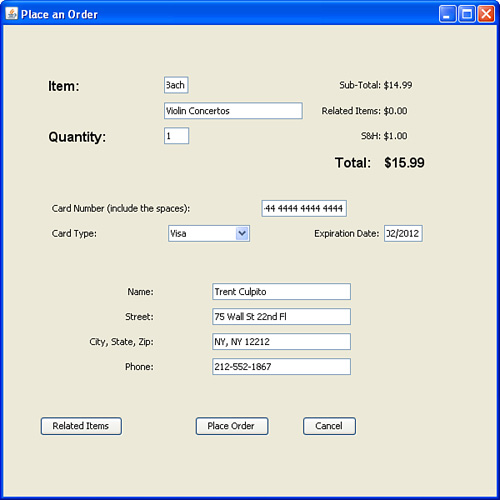
**Figure 2.10** The Place an Order dialog box



**10.** Fill in the Card Number field with data such as 4444 4444 4444 4444.

**11.** Click the **Expiration Date** field and enter data such as 02/2012. See [Figure 2.11](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig11).

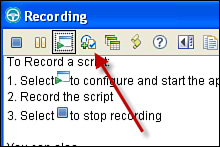
**Figure 2.11** The completed Place an Order dialog box



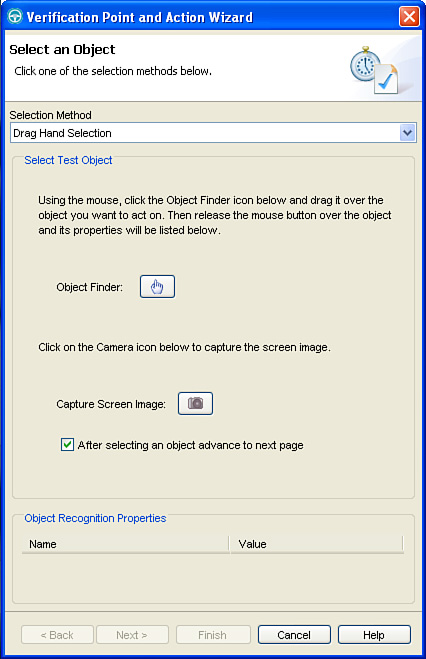
To prove that the application under test produces the same answers when you run the test script as when you recorded it, your test script must be enhanced with verification points. A *verification point* is a check that the actual data and its properties shown in the currently executing application match your expectations for the data and properties. Our expectations are called the baseline. A difference between actual and baseline causes Rational Functional Tester to write a fail status for the verification point into the execution log. Verification points are important because they are the only means in Rational Functional Tester to confirm that the application under test gives correct information to its users.

**12.** As part of your testing of ClassicsJavaA, you need to verify that the total price displayed is $15.99. Click the **Verification Point and Action Wizard** icon in the Recording Monitor, as shown in [Figure 2.12](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig12). The Verification Point and Action Wizard displays, as shown in [Figure 2.13](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig13).

**Figure 2.12** Clicking the Verification Point and Action Wizard icon causes the Recording Monitor window to disappear and the Wizard to display.

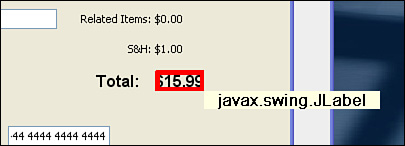


**Figure 2.13** The Verification Point and Action Wizard gives you the options to identify the object to be verified.



**13.** Click the **hand** icon, but hold down the mouse button. Drag the **hand** icon to the $15.99 value. To indicate your selection, a red rectangle is drawn around the text of the total, as shown in [Figure 2.14](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig14). Release the mouse button to select this text for verification.

**Figure 2.14** When you drag the hand icon over an object in the application, Functional Tester displays a red rectangle that enables you to confirm that you have selected the correct object.



When you release the cursor, the properties of the object selected become visible at the bottom of the wizard. You can use those properties to confirm that you have selected the correct object.

In this window, you can define what kind of verification point has to be created. These are the available options:

• Data Verification—Use to validate the displayed text of an object.

• Properties Verification—Use to validate one or more properties of an object, including its text (if any), font, color, or any combination of properties (for example, if it is both selected and has a particular color).

• Get a Specific Property Value—Use to get a specific property value so it can be tested in your script.

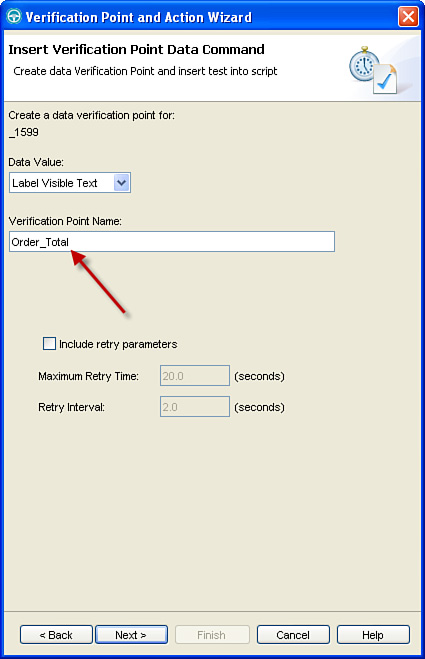
• Wait for Selected Test Object—Rational Functional Tester waits until this object becomes available. This verification point is used in situations where Rational Functional Tester must wait for the application under test to create a window or dialog box.

• Perform Image Verification Point—A graphical verification point.

**14.** For this exercise, use the Data Verification Point. Click **Next**.

**15.** In the Insert Verification Point Data Command dialog box, change the name of the verification point from \_1599\_text to Order\_Total, as shown in [Figure 2.15](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig15). Click **Next**.

**Figure 2.15** The verification point gets its name from the name of the object you select, but you can use a more meaningful name instead.

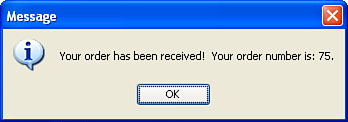


**16.** In the Verification Point Data dialog box, you can modify the data or affect how it is verified. For this exercise, you can use the defaults that are supplied by Rational Functional Tester. Click **Finish**.

At this moment, the Data Verification Point is created and inserted as code in the program. We continue with recording the interactions. While recording, you can add various verification points.

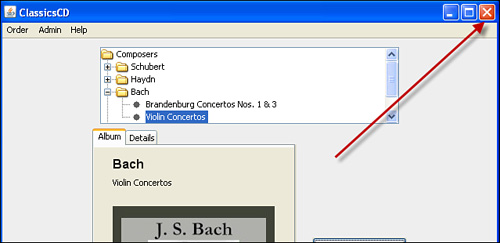
**17.** In the ClassicsJavaA Place an Order window, click the **Place Order** button. A dialog box should display, showing that your order has been received; see [Figure 2.16](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig16). Click the **OK** button, and this dialog box should disappear.

**Figure 2.16** Your order has been received, and the dialog shows your order number.



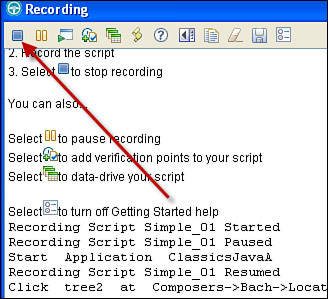
**18.** Close ClassicsJavaA by clicking on the **X** in the upper right-hand corner of the ClassicsJavaA window; see [Figure 2.17](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig17). All ClassicsJavaA windows (there should be only one at this point) should disappear.

**Figure 2.17** Close ClassicsJavaA by clicking the X to close the window.

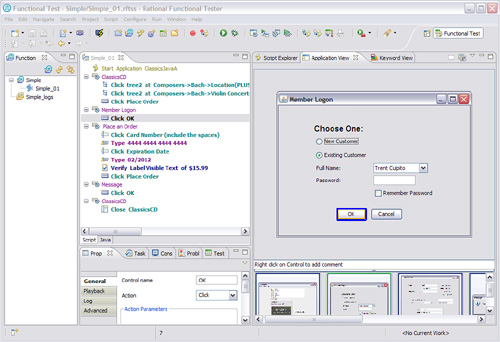


**19.** Click the **Stop Recording** button in the Recording Monitor window, as shown in [Figure 2.18](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig18). The Rational Functional Tester main screen automatically reappears, displaying your test script (see [Figure 2.19](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig19)).

**Figure 2.18** The Stop Recording button stops recording, causing the Rational Functional Tester main window to reappear.



**Figure 2.19** The appearance of Rational Functional Tester after recording a script with Storyboard Testing enabled.



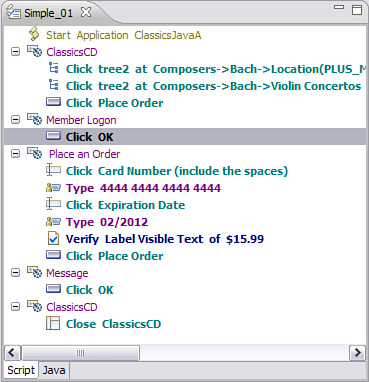
**Your Recorded Script**

After recording, you can view your test script in a test view. This view is also an editor, so you can make changes at this time. Editing is discussed in the “[Script Editing and Augmentation](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02sec1lev6)” section.

**Test View, Script Tab**

The most noticeable change from the traditional views is the scripting language. For Storyboard Testing, the test script is represented in English statements. During recording, Rational Functional Tester inserts statements into the script corresponding to the actions you performed in recording the test. See [Figure 2.20](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig20).

**Figure 2.20** A Storyboard Testing script, showing statements in groups. Nontechnical users find that these scripts are easier to read than the equivalent programming language.



Each time a new window or dialog box displays in the application under test, Rational Functional Tester creates a new group. The name of the group is the same as the caption for the window or dialog box. All the actions taken in that window or dialog box are indented under that group. You can rename a group by clicking on the name of the group and typing a new name.

Each statement begins with an icon indicating the type of object acted upon by the statement. A statement clicking in a tree has a tree icon, a statement clicking on a button has a button icon, and so on.

The action, such as click or type, is followed by the name of the object being acted upon. In some cases, this is followed by where in the object the action should occur, as with clicking on the portion of the tree that contains the Bach Violin Concertos CD being represented as **Composers > Bach > ViolinConcertos**.

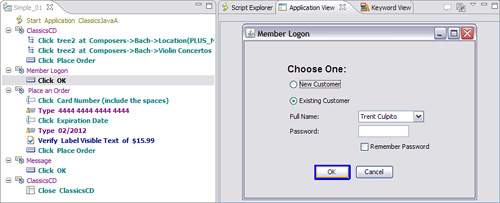
Verification points begin with an icon for a checkmark on a page, followed by the word Verify. This is followed by the verifying action that you selected: Verify Data in the object, Verify Properties of the object, Get Property of the object and which property to get, Delay Execution for the existence of the object, and Verify Image of the object.

Note that many times, text objects do not include a proper object name; this is true of the total in the example of $15.99. In these cases, Rational Functional Tester uses the text of the object as its name; this is why we needed to rename the verification point to Order\_Total. The Verification Point statement is Verify Data in $15.99, reflecting Rational Functional Tester’s treatment of the text as the name of the object.

**Application View**

Another change from the traditional perspective is the presence of the Application view. This view shows a screenshot of the application for each action in the script. If you click a particular line in the script, the screenshot displays in the Application view, and the particular object you are acting upon is highlighted with a blue rectangle. See [Figure 2.21](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig21).

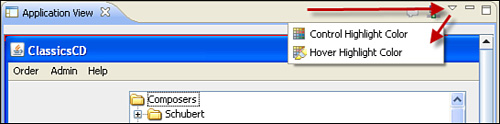
**Figure 2.21** A Storyboard Testing script, showing the screenshot in the Application view. You have just clicked the line Click Place Order in the script, and the Place Order button is highlighted in the screenshot.



If you hover your mouse cursor over an object in the screenshot in the Application view, the object is outlined in red. This enables you to see the various “user interface controls”—the objects you can act on or verify—that are used in the application.

You can change the colors used for highlight and for hover by clicking the View menu at the upper left of the Application view; see [Figure 2.22](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig22). Selecting one of the items brings up a color palette that permits you to change the color to one you prefer.

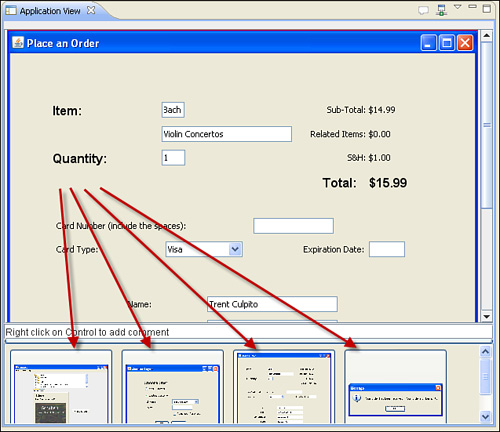
**Figure 2.22** The View Menu for the Application View enables you to change the color used for highlighting the object (or control) that you acted upon when you were recording and change the color used when you hover your mouse cursor over an object.



If you click successive lines of your script, you can use the Application view to visualize how you drove the application during recording. This is useful if someone other than you created the recording; you can see what they did by reading the script or by looking at the screenshots of the application.

In addition, the bottom of the Application view shows all the screenshots from the application under test, in small thumbnail form. See [Figure 2.23](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig23). If you hover your mouse over one of these screenshots, it automatically enlarges into a new dialog so you can see it clearly. To close one of these enlarged views, click anywhere in the Rational Functional Tester main window. You can also add comments to these thumbnail screenshots to give a reminder to yourself or someone else about a particular screen or object.

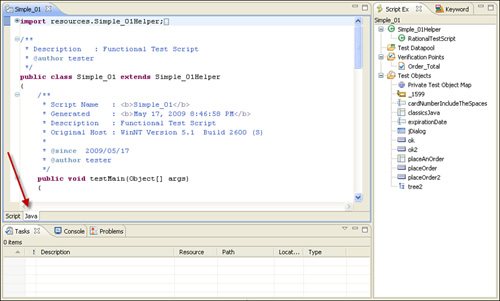
**Figure 2.23** The bottom of the Application view contains thumbnail views of all the screens you visited when you created your recording. Any screens you did not visit are not shown.



**Storyboard Testing View, Java Tab**

Click the **Java** tab at the bottom of the Script view; it is the tab to the right of the Script tab. You can click these two tabs to switch between the English statements and the Java statements. See [Figure 2.24](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig24). Note that when you switch, the Application view is hidden and the Script Explorer view becomes visible. If you click the **Script** tab, the Application view reappears.

**Figure 2.24** A Storyboard Testing script, showing the Java equivalent to the English language statements



You can edit your script in either view, but changes you make in the Java view are not shown in the Script view. See the section “[How to Correct or Update a Test Script](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02sec2lev16)” later in this chapter for the recommended way to add Java statements to your script so that the statements are visible in both views.

**Playing Back a Script and Viewing Results**

Storyboard Testing does not directly affect the playback process. The following sections are identical to the sections in [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01), except explanations have been added to areas where you might notice a difference.

**Resetting the Test Environment Prior to Playback**

To play back successfully, you must reset your application to its original state. For the example, the ClassicsJavaA application was closed. If you create a recording that begins with your application open and display a particular screen, you should reset the application to that screen prior to playing back the script. You should ensure that no other windows or dialogs from your application are open. You should also close any other applications or browser windows that are open; your desktop should look the same as it did when you recorded your script.

In addition, you should reset the test environment to the same condition it was in when you created the recording. This might require that you delete data you created during the recording process. For example, your application might not allow you to create two records by entering the same data. To play back your script, you should first delete the record.

ClassicsJavaA does not require you to provide or reset any data or add or delete files to your file system, even if you execute it many times. For this exercise, you can proceed directly to playing back the script. However, when you are testing your own application, you should ensure that you have correctly reset your test environment.

**Playing Back the Script**

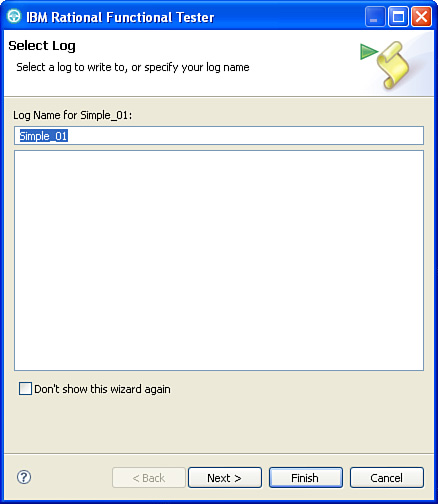
It is possible to detect differences between what you see in the current application build and excepted result by playing back a test script.

Note

When you use Rational Quality Manager as your test management solution, executing a test case causes the test script to be executed. Execution is started from the web interface of Rational Quality Manager.

**1.** Click the **Run Functional Test Script** button in the menu (you can also right-click a specific test script and click **Run** on the pop-up menu). You are prompted to specify log file name as shown in [Figure 2.25](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig25).

**Figure 2.25** The Select Log window



**2.** Define the name of the test log in the **Select Log** window. For this example, you can select the default, which is the same as the name of your script but uses a different file extension.

**3.** Click the **Finish** button. Test execution begins. Because Rational Functional Tester requires exclusive access to the mouse and keyboard, it is not possible for you to use other applications, lock the computer, or allow the screensaver to become active while the test is executing. If you move the mouse during execution, Rational Functional Tester may click on the wrong object. If you make a window active during execution, Rational Functional Tester will direct all its input to that window. Either of these actions can cause your script to fail.

**4.** Watch the progress of the playback. Note any behavior in your application that does not match the behavior it exhibited when you recorded the script.

In the Playback Monitor, you can see which statement Rational Functional Tester is executing. Rational Functional Tester spends the bulk of the time during playback waiting for objects to appear or to become active. When the test execution ends, the Rational Functional Tester editing window reappears.

**View Results**

If you selected the HTML log type or left the log file preference set at the default, a browser appears to display the execution log file. You can also double-click a log file in Rational Functional Tester to view it. The HTML log file contains three sections that enable you to navigate quickly through the display:

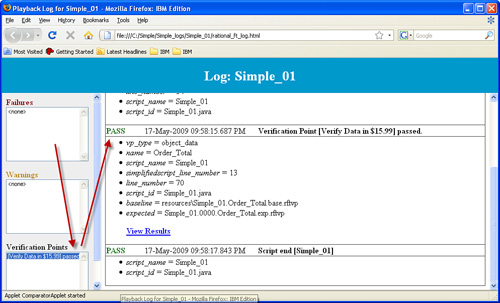
• Failures

• Warnings

• Verification Points

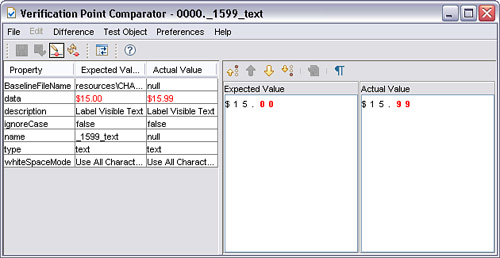
You can click any of these entries in these sections, and the browser scrolls to display that entry in detail, as shown in [Figure 2.26](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig26).

**Figure 2.26** An example log file in HTML format



In the case of a failing verification point, you can view the difference between expected and actual by clicking the **View Results** hyperlink. This activates the Verification Point Comparator, as shown in [Figure 2.27](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig27). You should not have a failing verification point; the figure has been captured from a different script.

**Figure 2.27** When a verification point fails, you can view the differences between expected and actual. The baseline can be updated.



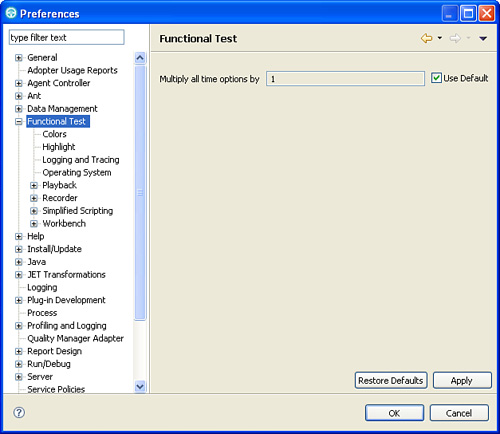
With the Verification Point Comparator, you can update the baseline with the Replace Baseline with actual result option. It is also possible to start the Verification Point Comparator directly from the logs in Rational Functional Tester.

**Timing Issues**

It might be that playback failed with a timeout error. This occurs because Rational Functional Tester attempts to play back your script as fast as possible, and sometimes your application runs slower than usual. For example, excessive network traffic might slow your application’s capability to retrieve data from a server. One way to correct for this and enable your script to execute is to slow down the script execution. Several options are available:

• Slow down by modifying the Rational Functional Tester “Multiply all time operations by” preference, as shown in [Figure 2.28](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig28). If you change this value to a 2, your script takes roughly twice as long to execute. Setting the value to 3 makes your script run 3 times as long. For example, a script that would run in 5 minutes if the value was 1, would take about 15 minutes for a value of 3, and so on for other values. However, changing this value to 3 also makes Rational Functional Tester wait 3 times as long for responses from your application, which is exactly the desired effect when the application is responding 3 times slower than usual. You should remember to reset this parameter because otherwise all test scripts you execute run slower.

**Figure 2.28** The Multiply all time operations by preference for Rational Functional Tester. Reset when running in production.



• Adding a sleep statement. This statement causes Rational Functional Tester to wait for a specified number of seconds. Note that Rational Functional Tester always waits for this number of seconds, even if your application happens to be running fast. You can add a sleep statement by finding a line of your script that is before where the timeout occurs; the sleep statement is added after the line you select. You then click the **Insert Recording into Active Functional Test Script** icon on the Rational Functional Tester toolbar. When the Recording Monitor displays, click the **Insert Script Support Commands** icon. In the Script Support Functions dialog box, click the **Sleep** tab. Enter the number of seconds you would like Rational Functional Tester to wait, then click the **Insert Code** button. Next, click the **Close** button. Finally, click the **Stop Recording** icon.

• Adding a waitForExistence verification point. If your script fails with a timeout error because a window, dialog box, or some other object does not display quickly enough, you can insert a waitForExistence verification point. As with adding a sleep statement, you find the point in your script where the dialog should display. Typically, this is the first statement in a group where the group contains the name of the new window or dialog. Click the **Insert Recording into Active Functional Test Script** icon on the Rational Functional Tester toolbar. When the Recording Monitor displays, click the **Verification Point and Action Wizard** icon. Drag the hand onto the window or dialog or other object, and then release the mouse button. In the Verification Point and Action Wizard dialog box, click **Wait for Selected Test Object**. Click **Next**. The default is for Rational Functional Tester to test to see if the object exists, and if it does, proceed with the remainder of the test script. If the object does not exist, Rational Functional Tester waits for the Check Interval, which is by default 2 seconds, and then test again to see if the object exists. This test-and-wait cycle continues until either the object finally exists, or else the Maximum Wait Time is reached. The default Maximum Wait Time is 120 seconds, or 2 minutes.

• If a waitForExistence verification point fails with a timeout error, you can change the Maximum Wait Time to a larger value. You can add the retry parameters to the other verification points (except for Get a Specific Property Value), either during recording or while editing.

**Script Editing and Augmentation**

Storyboard Testing makes several major changes to the editing and augmentation process. The following sections are similar to the sections in [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01), but the major simplifications that Storyboard Testing permits are described here.

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**

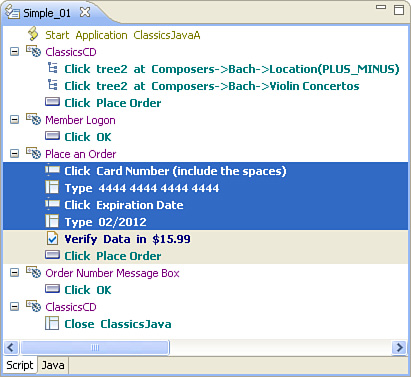
The most frequent type of editing you probably perform to a test script is fixing or updating. After you finish reading this book and start employing the best test script development practices, these corrections and updates should be short and simple. The two general steps in this activity are removing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

**Removing Lines from a Test Script**

You can remove unwanted lines of a Storyboard Testing test script in two ways: by deleting the lines or by disabling them. You always want to begin with the latter because you might need to restore the original lines. You can disable multiple lines of a test script as follows:

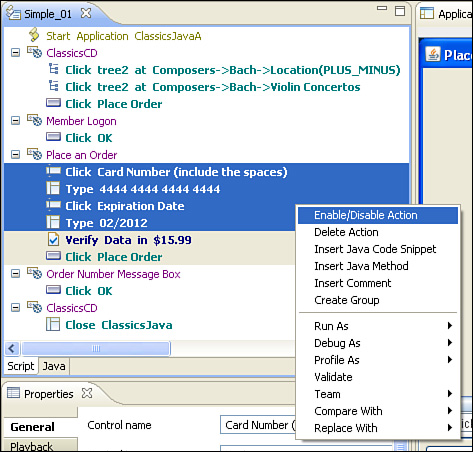
**1.** Select multiple lines of the script by clicking the first line to disable, holding down the Shift key, clicking on the last line to disable, and releasing the Shift key. All the lines should have a dark blue background indicating they are selected. See [Figure 2.29](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig29).

**Figure 2.29** Highlighting multiple lines in a script



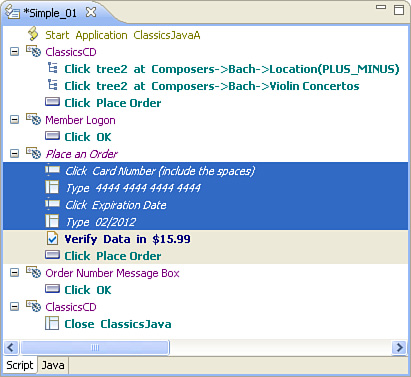
**2.** With your mouse cursor over the highlighted lines, right-click to bring up the popup menu, and click **Enable/Disable Action**. (The word Action refers to the action Rational Functional Tester takes against the application under test, not your action of enabling or disabling.) See [Figure 2.30](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig30).

**Figure 2.30** The popup menu that displays when you right-click with one or more lines highlighted



**3.** Click another line in the script to see that the selected lines are disabled. See [Figure 2.31](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig31).

**Figure 2.31** The lines are disabled because they appear in a light, italic font.



**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 Use the Application View](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02sec2lev17)” later in this chapter 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 in an existing script as follows:

**1.** Execute the application under test, driving it to the point where you want to add steps to your script. You can start your application, if necessary, either as you normally do outside of Rational Functional Tester or by starting it from the menu item **Configure** > **Configure Applications for Testing**, where the dialog has a Run button.

**2.** Click the line in the test script where you want to add new steps. The steps are added after the line you select.

**3.** Click **Script** > **Insert Recording** from the menu, or click the **toolbar** button. You immediately go into recording mode.

**4.** Perform the actions you want to add to your script.

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

Just as you must ensure that you choose the correct starting point for your new steps, you must also ensure that the step immediately after your new steps is able to execute. For example, if you finish recording without closing a dialog box, the next step in your script attempts to act on that dialog box. If it should act on a window instead, you should be certain to close the dialog box before stopping your recording.

**Moving Lines in a Test Script**

You can move a line in your test script by clicking on it to highlight it, then dragging it to the new location and releasing the mouse button. An I-cursor displays the location where the line moves when you release the mouse button.

**How to Use the Application View**

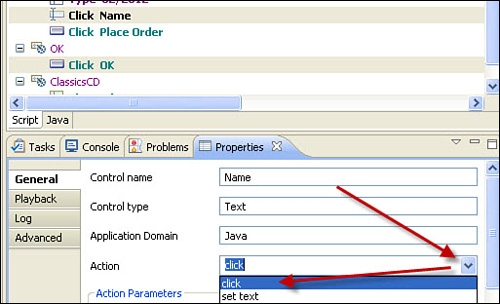
The next most frequent type of editing you likely do to a test is update your script by selecting an object in the Application view and adding an action against it to your script.

You can do this by clicking on an object to highlight it, right-clicking, and selecting an action from the pop-up menu. The statement is added to your script.

The specific actions you are able to perform depend on the type of object you have selected, but in general, you are able to insert a command, insert a comment, insert a verification point, or update the visual.

For this example, you add selecting a credit card type to the script, after entering the card number and expiration date. You can also drag any object from the Application view and drop it into your script. This inserts an action on the object, which is typically to click it. After the statement is inserted, you can go to the Properties view to change the type of action; see [Figure 2.32](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig32).

**Figure 2.32** Changing the action on an object via the Properties view



Note

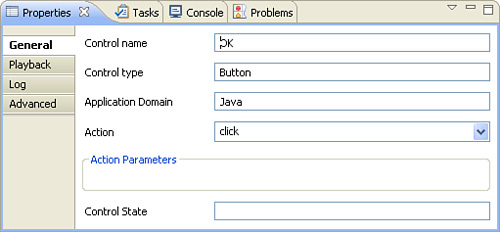
The default actions can be modified by editing the file C:\Program Files\IBM\SDP\FunctionalTester\bin\simplifiedScriptAction.rftssa.

Finally, you can select a Group statement in the Script view, and then in the Application view, you can add verification points for all the objects on a screen from the right-click menu.

**Properties View**

The Properties view enables you to see and edit properties for the line you have selected in your script. The properties are grouped into General, Playback, Log, and Advanced tabs. Each tab contains specific properties that depend on the particular line you have selected. See [Figure 2.33](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig33).

**Figure 2.33** The Properties tab enables you to change properties of any statement in your script.



**Datapools**

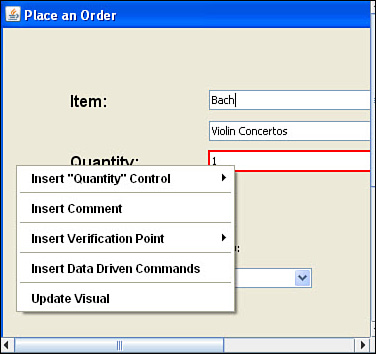
While you are recording, you can insert data drive commands in the same way as described in the “[Datapools](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html" \l "ch01sec2lev25)” section of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01). The command that is inserted for Storyboard Testing is shown in [Figure 2.34](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig34).

**Figure 2.34** In Storyboard Testing, a data drive command appears as Set Value...From Datapool Retrieved.



In addition, you can insert a data drive command from the Application view. Find the location in your script where you would like to drive an object using datapool data. In the Application view, right-click the object, and click **Insert Data Driven Commands**, as shown in [Figure 2.35](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig35).

**Figure 2.35** Inserting a data drive command from the Application view



**Adding Programming Code to Your Tests**

You can insert (or have someone insert for you) Java code into your script. This can be done either by right-clicking in the **Script** view and selecting **Insert Java code snippet**, or **Insert Java method**. When you select one of these options, template Java code with TODO comment will be inserted under **Java** tab in the Test View. Your Java code should be placed instead of TODO comment. All changes made with either right-click option are visible in the Script view as a single line indicating additions in Java.

**Other Script Enhancements**

So far, you have seen 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. Most likely, other people 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 people 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. To add a comment, right-click a statement in the Script view and click **Insert Comment**. Your comment is added after the statement you selected.

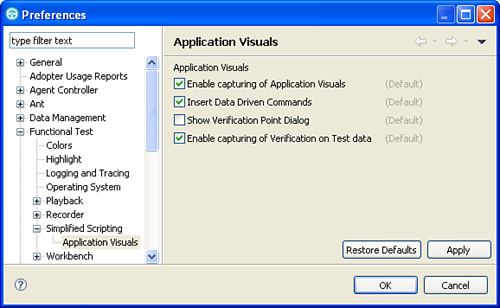
**Setting Preferences for Storyboard Testing**

The Storyboard Testing feature includes additional preferences you can modify. This section describes each preference and gives an example of how that preference affects your script when the preference is checked and when it is not checked.

**Preferences for Application Visuals**

The Storyboard Testing preference can be expanded in the left-hand navigation area of the Preference dialog box. Expanding it shows a selection that enables you to change preferences for Application Visuals, as shown in [Figure 2.36](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch02.html#ch02fig36).

**Figure 2.36** The options within Storyboard Testing for Application Visuals



If this preference is unchecked, the other preferences are not available.

**Enable Capturing of Application Visuals**

By default, Rational Functional Tester captures application visuals while you are recording. If you click the check box to deselect this preference, application visuals are not recorded for any script you subsequently record. You need to rerecord the script to add application visuals to it. Deselecting this preference also causes the other preferences for Application Visuals to be grayed out.

**Insert Data-Driven Commands**

If you select to insert data-driven commands, a datapool is associated with your script during recording. As you interact with objects during recording, you can add the objects to the datapool as columns, and the statements include actions to retrieve the data from the datapool instead of using the data you type.

**Show Verification Point Dialog**

If you select to show Verification Point dialog, the Verification Point wizard dialog box will be presented when you are selecting the Data Verification Points using the Application View.

**Enable Capturing of Verification on Test Data**

Select the check box to enable the option to capture verification on test data, and during recording, Rational Functional Tester will capture and persist all the verification point data associated with the object available on an application under test page. This will allow you to insert Data Verification Points using the application visuals displayed in the Application View.

**Summary**

This chapter provided an overview of the Storyboard Testing feature of Rational Functional Tester. The chapter started with similarities to the traditional perspective of Rational Functional Tester and showed that installation, configuration, and major options work unchanged in either perspective. Then the chapter duplicated the example of [Chapter 1](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html#ch01) in recording and playing back against an application under test. You should now be able to create, modify, and execute scripts in a natural manner.

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html)

[Chapter 1. Overview of Rational Functional Tester](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch01.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch03.html)

[Chapter 3. General Script Enhancements](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch03.html)