[Skip to content](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html" \l "container)

* [*Safari*](https://www.safaribooksonline.com/home/)
* [Recommended](https://www.safaribooksonline.com/r/)
* [Queue](https://www.safaribooksonline.com/s/)
* [Search](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)
* [Expand Nav](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)
  + [Recent](https://www.safaribooksonline.com/recent/)
  + [Topics](https://www.safaribooksonline.com/t/)
  + [Highlights](https://www.safaribooksonline.com/u/p253751/)
  + [Settings](https://www.safaribooksonline.com/u/)
  + [Feedback](mailto:feedback@safaribooksonline.com)
  + [Sign Out](https://www.safaribooksonline.com/accounts/logout/)
  + [Settings](https://www.safaribooksonline.com/u/)**10** days left in your trial. [Subscribe](https://www.safaribooksonline.com/subscribe/).
  + [Feedback](mailto:feedback@safaribooksonline.com)
  + [Sign Out](https://www.safaribooksonline.com/accounts/logout/)

[Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)

* [Search in book...](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)

Top of Form

Bottom of Form

* [Toggle Font Controls](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)
* [Share this](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)
  + [Twitter](https://twitter.com/share?url=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html&text=Software%20Test%20Engineering%20with%20IBM%20Rational%20Functional%20Tester%3A%20The%20Definitive%20Resource&via=safari)
  + [Facebook](https://www.facebook.com/sharer/sharer.php?u=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)
  + [Google Plus](https://plus.google.com/share?url=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html)
  + [Email](mailto:?subject=Safari:%20Chapter%204.%20XML%20and%20Rational%20Functional%20Tester&body=http://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch04.html%0D%0Afrom%20Software%20Test%20Engineering%20with%20IBM%20Rational%20Functional%20Tester%3A%20The%20Definitive%20Resource%0D%0A)

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html)

[Chapter 7. Managing Script Execution](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html)

[Chapter 9. Advanced Rational Functional Tester Object Map Topics](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html)

**Chapter 8. Handling Unsupported Domain Objects**

**Chip Davis**

*This chapter discusses how you can use Rational Functional Tester with applications and systems that contain interface objects that do not automatically work well with automated tests. These objects can be from old applications, customized or modified interfaces, or less common controls that are not currently supported by Rational Functional Tester.*

**Rational Functional Tester and Unsupported Objects**

Several things must occur for Rational Functional Tester to create and execute an automated test script that interacts with an application made up of objects:

• The test script must have a reference to a given object; this is typically created by the recorder and stored in a test object map.

• There must be at least one line of test script code to interact with or test the object.

• Rational Functional Tester must be able to find the object using recognition properties during test playback.

• Rational Functional Tester must be able to execute the correct action against the object during test playback.

If any one of these does not work correctly, you cannot effectively automate test steps against the given object. Test automation might still be possible, but there is a significant added level of effort to execute and maintain such tests.

An object that is not supported by Rational Functional Tester means that it is not able to automatically recognize the object and its properties. You cannot generate a new test object and line of test script code using the recorder. You also cannot use any of the usual test creation wizards, such as Verification Points or data-driven commands, on the object.

Running Without a Map

A test object map is not required for a test script if you create the object references in the script code. The examples in this chapter demonstrate how to do this.

What Is Supported and What Is Not?

You can find the latest list of objects supported by Rational Functional Tester in the product release notes, the Help documentation, and online at [www.ibm.com/software/awdtools/tester/functional/index.html](http://www.ibm.com/software/awdtools/tester/functional/index.html).

In many cases, Rational Functional Tester might identify the parent object or window and record coordinates for where the object is located in this window or parent object. In these cases where coordinates are used, Rational Functional Tester is not able to determine the object’s properties. It also does not know the possible actions that are available. For example, is it a drop-down list or is it a text box for entering a string? Even if you are able to use coordinates where the object is located, the test playback is likely to send incorrect actions to the object or fail to coordinate the timing required to interact with the object.

**Painted Objects**

Unsupported objects in this chapter are typically, but not always, visible GUI controls that are still objects in either the Windows or Linux operating systems. This accounts for most GUI controls that you use in a test procedure. However, there are some special interfaces that might look like traditional GUI controls, such as push buttons and menus, which are not objects. These interfaces *paint* controls, meaning that the appearance of the object is simply an image rendered on a larger object such as a window, region, or screen. One common example of this is an Adobe® Flash® (formerly Macromedia Flash) interface, which might contain painted objects for using parts of web-based applications.

Because painted controls are not implemented through the operating system as objects, you have to rely on coordinates (in the parent object) and other manual timing techniques to create automated test scripts.

**Using Rational Functional Tester with Unsupported Objects**

As previously mentioned, unsupported objects are still implemented as objects in either Windows or Linux. You can therefore still get Rational Functional Tester to interact with them even if they are not officially supported. You do this by determining the object structure and hierarchy of what you are trying to test, and then manually add appropriate lines of code. These steps produce a test script that can reliably identify the object and properly interact with it for testing. The general approach to do this is as follows:

**1.** Determine how to best find the object: by name, location in a sequence, or other means.

**2.** Identify the parent object, and then search down through the object hierarchy to the desired control or interface object.

**3.** After you find the object to test, you can add methods to perform actions, such as sending keys against the object.

**4.** To create a Verification Point, you must get an actual value and manually compare it to a baseline value. You must also manually log the test result.

**5.** Depending on the application and test procedure, you might have to add some steps or verification to synchronize the test script with the application.

Testing Other Interfaces

Although Rational Functional Tester might have difficulty with Adobe Flash, it does fully support Adobe Flex™. Rational Functional Tester also supports many terminal-based (sometimes called “green screen”) applications, which are usually painted interfaces. Finally, you can create automated functional tests for applications through Citrix® (another painted interface) using IBM Rational Performance Tester.

Finding Objects

You might have to add looping structures, such as do-while or other conditional structures, to find the object.

Collaborating with the Development Team

If you test an application interface that was developed in house or you have access to the team that developed the interface, you might consider reaching out to that team. It might be able to provide you with valuable information about the objects you are try to interact with and test. This can be an easy path to developing automated tests for unsupported objects.

The methods and script code that you use to interact with unsupported objects are in the Rational Functional Tester Application Programming Interface (API) and you add these programmatically, not using the recorder. This chapter looks at some of the key components of this API that you use to do this and shows several examples of how to do it.

The Rational Functional Tester API is well documented in the product Help.

**Key Components of the Rational Functional Tester API**

The Rational Functional Tester API consists of many packages, classes, and methods. Most of these are in the package com.rational.test.ft.object.interfaces, which enables you to work with the different types of objects that Rational Functional Tester interacts with. This section provides an overview of the parts of this API that are used to work with unsupported objects. This is not intended to fully document all possible methods that you might use, but it gives you a general sense of the API classes and methods before going into the examples.

**Interface Classes**

Two common interfaces you use from com.rational.test.ft.object.interfaces are:

• **IWindow—**Used to access the operating system’s windows and objects. This can be used on either Microsoft Windows or Linux, although the same test script code will not work on both. The methods in this interface are useful in identifying and finding unsupported objects that are in windows.

• **IScreen—**A similar interface that is also used to access and interact with windows and other objects from the operating system. Just like IWindow, this applies to both Microsoft Windows and Linux.

There are several useful methods available for these interfaces; getActiveWindow() and getActiveScreen() are commonly used with unsupported applications. For example, the following line of code gets the current active window when a script is running:

IWindow thisIsYourActiveWindow = getScreen().getActiveWindow();

**Key Methods**

The following list describes the common methods you use; these methods are from several different interfaces and classes in the com.rational.test.ft.object.interfaces package. Each of these methods apply to windows and objects that exist during test execution.

• The getScreen() method can be used to get a screen object.

• The getTopWindows() method can be used to get the top-level windows available during runtime.

• The getActiveWindow() method can be used to get the current active window.

• The getText() method can be used to get values (the text for the window) from a particular window.

• The inputKeys() method can be used to send a string or keystroke (such as Ctrl, Alt, Shift, and so forth) to the current active window.

• The inputChars() method can be used to send a string value (as a sequence of key events with no special interpretation of characters) to the current active window.

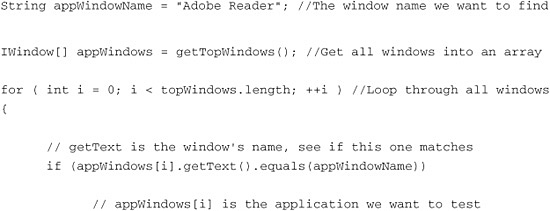
**Test Objects**

When you develop test scripts for an unsupported application interface, there is a chance that you might need to use Rational Functional Tester’s TestObject class. Any object you interact with this way is technically not an unsupported domain, but you might use a combination of the interface methods mentioned previously along with the TestObject methods. [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),” discusses this topic in greater detail.

**Identifying Objects**

The first example in this section shows you how you can identify the objects you want to interact with. Keep in mind that this is only one possible way to do this. This method enables the automated test script to find test objects; it is roughly equivalent to having an object stored in a test object map. We do not use the recorder or any wizard to do this; instead, we simply enter (or copy from other examples) the script code.

This example uses the IWindow interface along with the getTopWindows, getText, and other methods to find a window with a particular caption. This example finds the first Window with the caption “Adobe Reader.” In this case, the window caption is known, but there are also other methods available to the IWindows class that can be used to match a particular window. Note also that any application running in Windows or Linux has at least a container window, so this can be used with any unsupported domain.



About These Examples: Application Under Test

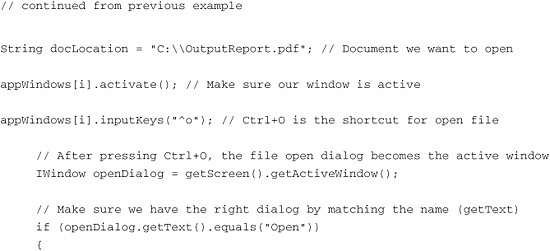
The examples shown here work with the Adobe Reader application, which is actually a supported domain interface (Win). You can use the normal Rational Functional Tester recorder to interact with these objects. This example was chosen as a common application that many readers on different operating systems might have.

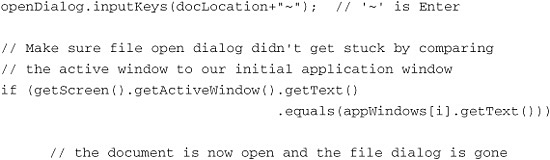
**Interacting with the Objects and Entering Data**

Now that you can identify the application you want to work with, you can interact with it. By interacting, you can click, drag, send keyboard input, or perform other general window manipulations. This is how you perform test steps and procedures by exercising the application under test, and it is equivalent to capturing test steps with the recorder. Like the previous example, you do not use the recorder or any wizard to do this.

In general, it is easier to send keyboard shortcuts to an unsupported domain application rather than clicking. The reason is that clicking typically requires either coordinates or the ability to identify and manipulate objects in the window. The latter is not always possible in unsupported applications. Most applications have various keyboard shortcuts to perform various functions, and Rational Functional Tester needs only to know what the application is (which was the end result of the previous example) to use one of these shortcuts.

This next example shows you how you can make the application under test open a document as part of the test procedure. To do this, you use a keyboard shortcut to bring up the Open File dialog box. You then send a static string for the file location and verify that the document was opened. The interactions use methods such as activate, inputKeys, and getActiveWindow to accomplish the test steps.





Error Handling in the Example

There is additional error-handling code that is not shown in the short examples here. This error handling is shown in the final listing of the complete example at the end of this chapter.

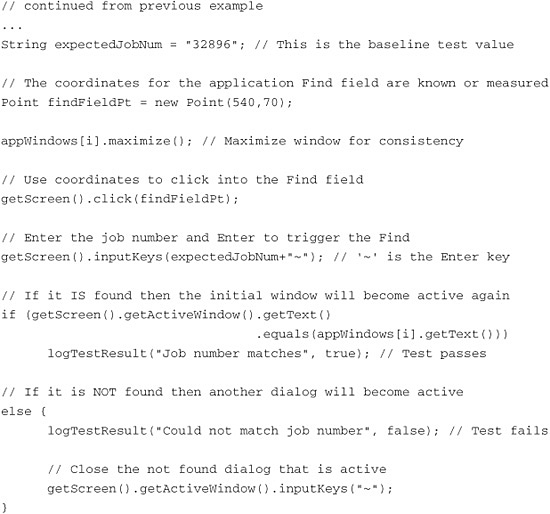
**Getting Data and Testing an Object**

The last thing you need to do to make your script into a test is to actually test something. To compare an existing state or value with an expected (correct) value, you first have to get information from the application under test. After you can get data from an application, creating a Verification Point (the test) is easy.

The technique you use to get information with Rational Functional Tester and an unsupported domain application might not be the same one that you would use if you were testing the application manually. This might also be a different technique than if you were recording supported objects with Rational Functional Tester. You need a simple and reliable way to both manipulate the application under test and get information about its behavior and the data it contains. In the example shown here, you rely on one of the application’s features (the find feature) along with the state of a dialog window to automate the validation of the application’s functionality. In many cases, with unsupported applications, the Clipboard Verification Point discussed 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),” can be easy and useful.

The document that was opened in the previous example contains a job number, presumably produced by the application being tested; this can be validated against a known correct job number. There are many ways to test an application; this example demonstrates using a function of the application under test to achieve the validation. The Find feature is used to locate the expected job number. If the value is found in the opened document, the test passes; if the value is not found, the test fails.

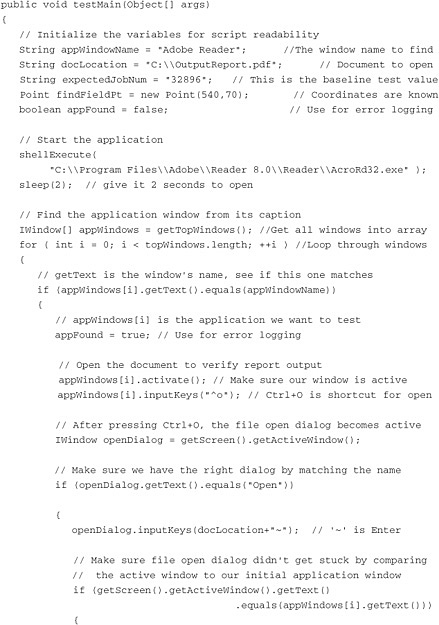
To do this, use some of the methods in the previous examples plus the click method and a simple if-then conditional with logTestResult to act as the Verification Point. Here, the click method with coordinates is used, which is sometimes necessary with unsupported domains. In actuality, the Adobe Reader application is supported by Rational Functional Tester, and you could recognize the Find toolbar object, but you are treating this as if it is an unsupported screen.

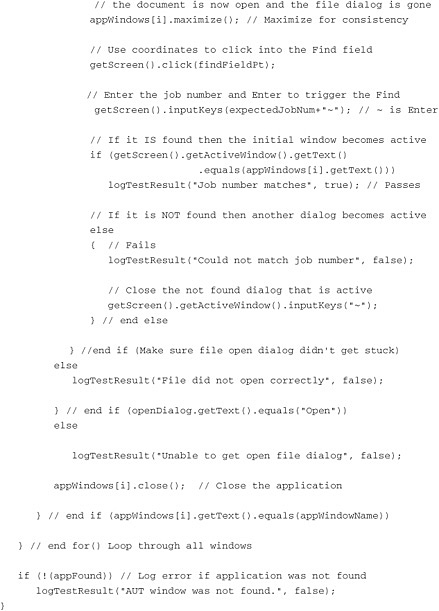


More About These Examples

In addition to the script code shown here, you can also add more lines using the recorder. The examples are shown in Java, although they can also be implemented in Visual Basic. The examples were created on the Windows platform, but they could have been created on Linux.

The following is the complete test script combining each of the previous examples. Some comments and error handling have been added.





**Unsupported Objects on Windows and Linux**

Although the API and methods described in this chapter can be used on both Windows and Linux operating systems, it is unlikely that a test script implemented on one platform using these methods will work on the other. However, it is even more unlikely that you would have the same unsupported objects to test on both platforms. Rational Functional Tester scripts developed in this way for unsupported objects should be considered specific to one particular platform.

**Summary**

Although the list of supported domains and objects keeps growing, there are always applications that are not supported by Rational Functional Tester. These might be old legacy programs or new, never-before-seen technologies. Even if Rational Functional Tester cannot automatically recognize and understand an application’s objects, you can still develop automated test scripts to work with them. The strategy to do this is different from using Rational Functional Tester’s recorder and object maps. With some knowledge of the application you are working with and the methods demonstrated in this chapter, you can create Rational Functional Tester scripts that interact with and test these unsupported applications.

[Prev](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html)

[Chapter 7. Managing Script Execution](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch07.html)

[Next](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html)

[Chapter 9. Advanced Rational Functional Tester Object Map Topics](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch09.html)

**Welcome to Safari**. Remember, your free trial will end on March 7, 2015, but you can [subscribe at any time](https://www.safaribooksonline.com/subscribe/)

* [Recommended](https://www.safaribooksonline.com/r/)
* [Queue](https://www.safaribooksonline.com/s/)
* [Recent](https://www.safaribooksonline.com/recent/)
* [Topics](https://www.safaribooksonline.com/t/)
* [Settings](https://www.safaribooksonline.com/u/)
* [Blog](http://blog.safaribooksonline.com/)
* [Support](http://support.safaribooksonline.com/?prod=flow)
* [Support](http://msupport.safaribooksonline.com/?prod=flow)
* [Feedback](mailto:feedback@safaribooksonline.com)
* [Sign Out](https://www.safaribooksonline.com/accounts/logout/)

© 2015 [Safari](http://www.safaribooksonline.com/).   
[Terms of Service](https://www.safaribooksonline.com/terms/) / [Membership Agreement](https://www.safaribooksonline.com/membership-agreement/) / [Privacy Policy](https://www.safaribooksonline.com/privacy/)

[**Make font larger Make font smaller**](https://www.safaribooksonline.com/library/view/software-test-engineering/9780137036455/ch08.html)