Working with Java Applications

SilkTest®
2008 R2

Borland®

Borland Software Corporation
8310 N. Capital of Texas Hwy
Building 2, Suite 100
Austin, TX 78731 USA
http://www.borland.com

December 2008
PDF

Table of Contents

Introduction to Java Tutorials

These tutorials are designed to introduce you to using SilkTest to test stand-alone Java applications developed with AWT and Java Foundation Class (JFC). These tutorials use the following sample Java applications that are included with SilkTest:

- AWT Test Application
- JFC Test Application

Borland no longer ships JRE 1.2 with SilkTest. In order to run the SilkTest sample Java applications, you may need to update the Java reference in the *.bat file that launches each sample application. If you do not have a local java.exe, you can download one from java.sun.com.

SilkTest also provides sample test applications for Microsoft .NET, Java SWT, Win32, and Adobe Flex. For more information, see the online Help

Update the sample Java application BAT file paths

In order to run the SilkTest sample Java applications, you may need to update the Java reference in the *.bat file that launches each sample application.

To update AWT_TestApplication.bat

This is the batch file that launches the Java AWT Test Application.

1. Use Windows Explorer to navigate to <*SilkTest installation directory*\JavaEx
2. Open AWT_TestApplication.bat with a text editor (such as Notepad).
3. Set JavaRun= to the directory in which java.exe is installed. For example:

   set JavaRun=C:\jdk1.5\bin
4. Save your changes. You can start the sample application by double-clicking the *.bat file, or by choosing Start/Programs/Borland/SilkTest 2008/Sample Applications/Java AWT/AWT Test Application.

To update JFC_TestApplication.bat

This is the batch file that launches the JFC Test Application.

1. Use Windows Explorer to navigate to <*SilkTest installation directory*\JavaEx\JFC
2. Open JFC_TestApplication.bat with a text editor (such as Notepad).
3. Set JavaRun= to the directory in which java.exe is installed. For example:

   set JavaRun=C:\Java\jdk1.5\bin
4. Save your changes. You can start the sample application by double-clicking the *.bat file, or by choosing Start/Programs/Borland/SilkTest 2008/Sample Applications/Java JFC/JFC Test Application.

Java AWT Tutorial

Objectives

This hands-on tutorial is designed to help you get comfortable testing a sample standalone Java application developed with AWT controls, using Java support in SilkTest.

In this tutorial, you will:

1 Set up for testing the application

2 Become familiar with the sample Java AWT application

3 Focus on a part of the application to test

4 Identify custom controls and learn when to record classes

5 Record classes for custom Java controls

6 Record window declarations for all controls you want to test

7 Prepare the test script file

8 Record a test against the Java application

9 Run the recorded test

10 Extend the test programmatically

11 Run the extended test

Before you begin this tutorial, complete the SilkTest tutorial to learn the basics of recording testcases, running testcases, and using the recovery system. If you installed the documentation when you installed SilkTest, you can access the tutorial by choosing **Start/Programs/Borland/SilkTest 2008/Documentation/SilkTest Tutorials** or **Help/Tutorials**.

Step 1: Use the Basic Workflow Bar to enable Java support

Java Support is configured automatically when you use Enable Extensions in the Basic Workflow bar.

The Java AWT Test Application is included with SilkTest.

Running the Java AWT test application:

> **Note** Borland no longer ships JRE 1.2 with SilkTest. In order to run the SilkTest sample Java applications, you may need to update the Java reference in the *.bat file that launches each sample application. If you do not have a local java.exe, you can download one from java.sun.com.

1. Install the applicable version JDK, if necessary.
2. If you have not done so already, update the sample application batch file (see “Update the sample Java application BAT file path” at the beginning of this document).
3. Select the Java AWT test application from **Start/Programs/Borland/SilkTest 2008/Sample Applications**.
4. Click **Enable Extensions** on the Basic Workflow bar, then select the sample application. SilkTest will prompt you to close and restart the sample application.

You are now ready to begin Step 2: Becoming familiar with the AWT Test Application.

Step 2: Becoming familiar with the AWT application

This tutorial uses a simple Java application that provides a mix of predefined Abstract Window Toolkit controls along with custom Java objects. (For more information about the Abstract Window Toolkit, see the online Help.) Explore the sample Java application now to become familiar with it. We are going to develop and run a test against the Drawing area, so be sure to visit this part of the application, as instructed below.

To explore the sample application:

1. Launch the application from your Windows Start menu by choosing: **Start/Programs/Borland/SilkTest 2008/Sample Applications/Java AWT/AWT Test Application**. The Test Application dialog opens with a menu bar containing four menus.
2. Click the **Control** menu, and then click **Drawing area**. The Drawing area window opens. It contains a canvas for drawing points and lines with the mouse, an Event Log that records mouse actions in the canvas, a Reset button, and an Exit button.
3. Click and drag the mouse inside the canvas to draw points and lines. Note how the mouse actions are recorded in the Event Log. Click the **Exit** button to close the Drawing Area window.
4. Experiment with other controls accessible from the Control menu.
5. Explore other menus in the application's menu bar.

   Note The DisabledMenu menu appears grayed out to illustrate how a disabled menu item is supposed to look. It was not designed to be enabled.

Continue to Step 3: Focusing on a part of the application to test.

Step 3: Focusing on a part of the application to test

Now that you have become familiar with the Drawing area in the sample Java AWT application, we are going to develop a test to verify that mouse events in the canvas are recorded accurately in the Event Log.

We selected the Drawing area window as a test candidate because it contains both predefined Abstract Windowing Toolkit controls and a custom Java object. SilkTest allows you to test both types of controls, but before we can create our test frame, we must first identify the custom Java control in the Drawing area window.

Continue to Step 4: Identifying custom controls in the Drawing Area window.

Step 4: Identifing custom controls in the Drawing Area window

SilkTest provides 4Test class definitions for commonly used Java objects, such as Abstract Window Toolkit (AWT) controls. For more information, see *Abstract Window ToolKit* in the online Help. With our Java support, you will be able to use 4Test methods and properties to test Java objects that belong to these predefined classes, which can be found in javaex.inc. For more information about predefined classes, see *Predefined Java classes* in the online Help.

Java support also allows you to test custom controls, which appear as CustomWin objects in Java applications. You will need to record classes for custom controls to gain access to their native methods and properties. You can then use these native methods and properties to develop scripts for testing custom Java objects in the application.

Let's identify the custom controls we want to test in the Drawing area window.

At this point, we assume you have started the sample Java application, and become familiar with the Drawing area window.

To identify custom controls in the sample Java application:

1. Choose **Control/Drawing area** in the Java application. The Drawing Area window opens.
2. In SilkTest, choose **Record/Window Declarations**.
3. Move the mouse pointer inside the **Event Log** (the scrollable white rectangular area). SilkTest sees the Event Log as a JavaAwtTextField object, a predefined AWT control.
4. Move the mouse pointer over the **Reset** button, then over the **Exit** button. SilkTest sees these buttons as JavaAwtPushButton objects, predefined AWT controls.
5. Move the mouse pointer inside the drawing canvas (the gray rectangular area). SilkTest sees the drawing canvas as a CustomWin object, a custom Java control.
6. Click **Close** to close the Record Window Declarations dialog.

We have determined that the drawing canvas is a custom Java control. Since we will use this custom control in our test, we need to record a class for the drawing canvas; continue to Step 5: Recording a class for the Drawing Area canvas.

Step 5: Recording a class for the Drawing Area canvas

Now we are going to record a class for the canvas in the Drawing Area window.

At this point, we assume the Drawing Area window is still open in the sample Java application.

To record a class for the drawing canvas:

1. Open a new include file. In SilkTest, choose **File/New/4Test include file**, and then click **OK.** An untitled 4Test include file opens.
2. Save the include file as **canvas.inc** in the extend subdirectory of the directory where you installed SilkTest.
3. Disable recording multiple tags. Choose **Options/Recorder**, make sure Record multiple tags is not checked, and then click **OK**.
4. With canvas.inc as the active window, choose **Record/Class/Scripted**. The Record Class dialog opens.
5. Make sure **Show all classes** is not checked in the Record Class dialog.
6. Move your mouse pointer over the drawing canvas in the sample Java application.
7. When DrawingCanvas appears in the Class Name field, press **Ctrl+Alt**. Native methods for the drawing canvas appear in the Record Class dialog.
8. Click the **Derived From** drop-down menu and search the list of available 4Test classes. Since there is no class type in the list that maps directly to drawing canvas, select **AnyWin**, a generic class.
9. Click **Paste to Editor** to record the new class declaration in the include file.
10. Click **Close** to close the Record Class dialog.

Now, you are ready to record window declarations for the predefined Java controls and the custom DrawingCanvas object in the Drawing Area window; continue to Step 6: Recording window declarations for the Drawing Area.

Step 6: Recording window declarations for the Drawing Area

At this point, we assume that you recorded a class for the canvas and that the Drawing Area window is still open in the sample Java application.

To record window declarations:

1. Create a new test frame file. In SilkTest, choose **File/New**, select **Test frame** in the New File dialog, and click **OK**. The New Test Frame dialog opens.
2. Select **Test Application** in the Application field and edit the **Frame filename** field so it reads:

   <*SilkTest install directory*>\extend\draw.inc
3. Click **OK**. The 4Test Include File draw.inc opens and is automatically loaded in SilkTest.
4. Manually load the class include file *canvas.inc*, which you created in Step 4. In SilkTest, choose **Options/Runtime**.
5. In the **Use files** field, click the **Browse** and select **canvas.inc** from the extend directory. Click **Open**. The canvas.inc file is added to the Use Files field.
6. Click **OK** to close the Runtime Options dialog.
7. With draw.inc as the active window, choose **Record/Window Declarations**.
8. Move your mouse pointer over the title bar of the Drawing Area window. When DrawingArea appears in the Identifier field, press **Ctrl+Alt**. Declarations are captured for the Drawing Area window and all of its controls.
9. Click **DrawingCanvas** in the Window Declaration list and change the name in the Identifier field to **Canvas**.
10. Click **Paste to Editor** to record the declarations in draw.inc.
11. Click **Close** to close the Record Window Declarations dialog.
12. Click **Exit** to close the Drawing Area window.
13. Save and close draw.inc.

Now, you are ready to prepare the test script file. Continue to Step 7: Preparing the test script file.

Step 7: Preparing the test script file

Before you can record the test, you must prepare the test script file.

At this point, we assume that you recorded window declarations for Drawing Area controls, loaded draw.inc and canvas.inc, and set up the recovery system.

To prepare the test script file:

1. Close all secondary windows in the sample Java application so that only the main Test Application window is open.
2. Open a new test script. In the SilkTest menu bar, choose **File/New/4Test script**, and then click **OK**. An untitled 4Test script file opens.
3. Save the test script file as **draw.t**.
4. With draw.t as the active window, choose **Record/Testcase** in the SilkTest menu bar.
5. Change the testcase name to **LogMouseMoves,** and then select **DefaultBaseState** from the pulldown menu in the Application state field.

Now you are ready to record a test against the Event Log in the Drawing Area window; continue to Step 8: Recording a test against Drawing Area controls.

Step 8: Recording a test against Drawing Area controls

Now that you have prepared your test script, you are ready to record actions for our test. Before you begin recording, expand this window so you can see all instructions. If you should need to scroll in this window after you start recording, click **Pause Recording** in the Record Status window so your scrolling actions do not get recorded as part of the test case.

At this point, we assume that you have created and saved your 4Test script file as draw.t. We also assume that the Record Testcase dialog is open, showing the testcase name as LogMouseMoves and the application state as DefaultBaseState.

To record actions for verifying the Event Log function:

1. In the Record Testcase dialog, click **Start Recording**. The recovery system sets the DefaultBaseState application state for the sample Java application. Once the DefaultBaseState is established, the Record Status dialog opens and you can start recording your testcase.
2. Move your cursor into the main window of the sample application. When you see Test Application at the bottom of the Record Status window, choose **Control/Drawing area**.
3. Click once inside the drawing canvas (the gray rectangular area in the Drawing Area window). Your mouse click is recorded as two mouse events in the Event Log: **Button down** and **Button up**.
4. In the Event Log, drag your mouse pointer to select the text string **Button down**.
5. With your mouse pointer inside the Event Log, press **Ctrl+Alt**. The Verify Window dialog appears. Make sure that **JavaAwtTextField DrawingArea.EventLog** appears as the window to verify.
6. Click the **Method tab**, and then check **Include inherited** to display the 4Test methods available for the Event Log.
7. Scroll down and select **VerifySelText**. Look at the description of the VerifySelText. We are going to use this method to verify that the Event Log correctly recorded the first mouse action in the drawing canvas, which is Button down, the text you selected in step 4.
8. Click inside the text field labeled, **The string or list of string you expect selected**. We expect that the Error Log recorded Button down, so type this text (including the quotation marks):

   "Button down"

   and then click **OK**.
9. Repeat steps 4 through 8, this time for the second mouse action, **"Button up"**.
10. Move your cursor over the **Exit** in the Drawing Area window. When Exit appears in the Record Status window, click **Exit**, and then click **Done** in the Record Status window. The Record Testcase window reappears, displaying your actions as translated into 4Test commands.
11. In the Record Testcase window, click **Paste to Editor**. The testcase LogMouseMoves is pasted into draw.t
12. Expand the testcase by clicking in LogMouseMoves, and then choose **Outline/Expand All**.

13 Save **draw.t**, but do not close it.

Now you are ready to run the recorded test; continue to Step 9: Running the recorded test against the sample Java AWT application.

Step 9: Running the recorded test against the sample Java AWT application

At this point, we assume that the test script file draw.t is open.

To run the testcase LogMouseMoves:

1. Set keyboard and mouse delays. In the SilkTest menu bar, choose **Options/Agent**. In the Agent Options dialog, set keyboard event delay and mouse event delay both to 0.01. Click **OK**.
2. With draw.t as the active window, choose **Run/Run**. SilkTest runs the test, restoring the sample AWT application to its base state and interacting with the application. The testcase passes.

Now, let's extend the testcase by adding code to verify that the Event Log records the correct mouse click coordinates; continue to Step 10: Extending the test programmatically.

Step 10: Extending the test programmatically

So far, we have verified that the Event Log records the Button Down and Button Up mouse events correctly when we click once to draw a single point in the drawing canvas. To extend this test, we should verify whether the Event Log also records the correct coordinates of the point we draw in the canvas.

To make this a more general test, we will create two integer variables to store the X and Y coordinates of a point. In addition, we will set each variable to a 2-digit random number within a range of values that falls inside the drawing canvas.

At this point, we assume that the sample Java application is running and not minimized, and that your test script draw.t is still open.

To expand the test to verify point coordinates:

1 Determine the range of acceptable values for point coordinates by checking the Rect property of the drawing canvas, as follows:

- In the sample Java application, choose **Control/Drawing area**.
- In SilkTest, choose **Record/Actions**.
- Click the title bar of Drawing area window and move your mouse pointer into the drawing canvas. When you see Press <Ctrl+Alt> to verify window Canvas appear at the bottom of the Record Actions dialog, press **Ctrl+Alt** to bring up the Verify Window.
- The values for the Rect property in the Properties to Verify window. On our system, the values showed that the range of acceptable X coordinates was 0 to 314 and the range of acceptable Y coordinates was 0 to 60. These values might be different on your system, depending on your display settings and other system configuration parameters.
- Click **Cancel** to close the Verify Window, and then click **Close** to close the Record Actions window.
- Click **Exit** to close the Drawing Area window.

2 Expand the testcase LogMouseMoves in your draw.t file and make the following changes:

- Above the recording block, declare two integer variables, **iX** to store an X-coordinate value and **iY** to store a Y-coordinate value. Use the RandInt function to set iX and iY to two-digit random numbers that fall within the range of acceptable coordinates, as determined from the Rect property of the drawing canvas. See *RandInt function* in the online Help for more details. We set iX to between 10 and 99 and iY to a random number between 10 and 60. Here is some sample code which you can copy and paste line by line into your test script. Substitute different numbers in the RandInt function calls, if necessary.

  ```
  int iX = RandInt(10,99)
  int iY = RandInt(25,60)
  ```

- Inside the recording block, substitute iX and iY as the second and third arguments in the command DrawingArea.Canvas.Click. This code forces your test to always draw a point at a random location within the boundaries of the drawing canvas. The following is sample code which you can copy and paste into your test script.

  ```
  DrawingArea.Canvas.Click (1, iX, iY)
  ```

- After the VerifySelText ("Button up") command, add 4Test code to select and verify the two-digit X coordinate and two-digit Y coordinate of the mouse event recorded as the Button Down… text string in the Event Log. Below is some sample code which you can copy and paste line by line into your test script.

  In this code, we determine the selection range of the X and Y coordinates in SetSelRange by counting text characters from left to right in the string **Button Down at (*xx*,*yy*)**. We force our random integers to be two-digit numbers so the selection range will have a constant start and end value for each coordinate.

  Note that we use the **str** function to convert our integer variables to string arguments that can be passed to the VerifySelText commands.

```
DrawingArea.EventLog.SetSelRange (1,17,1,19)
DrawingArea.EventLog.VerifySelText (str(iX))
DrawingArea.EventLog.SetSelRange (1,20,1,22)
DrawingArea.EventLog.VerifySelText (str(iY))
```

3 Save **draw.t**, but do not close it.

You are now ready to run the extended testcase LogMouseMoves; continue to Step 11: Running the extended test.

Step 11: Running the extended test

To run the testcase LogMouseMoves:

With the test script file draw.t as the active window, choose **Run/Run**. SilkTest runs the test, restoring the sample Java application to its base state and interacting with the application. The testcase passes.

Congratulations! You have completed the Java tutorial. Next, you might want to explore *Overview of Java Support* in the online Help.

Java JFC/Swing Tutorial

Objectives

This hands-on tutorial is designed to help you get comfortable testing a sample standalone Java application developed with Java Foundation Class (JFC) Swing controls, using Java support in SilkTest. See the online Help for descriptions of JFC and Swing controls.

In this tutorial, you will:

1 Set up for testing the application

2 Become familiar with the sample JFC Test Application

3 Focus on a part of the application to test

4 Record window declarations for all controls you want to test

5 Set up the recovery system for testing Java applications

6 Prepare the test script file

7 Record a test against the JFC Test Application

8 Run the recorded test

9 Get native methods for a predefined JFC Swing class.

10 Record new window declarations

11 Develop a new test using native methods

12 Run the new test

To get started with this tutorial, make sure you meet all the requirements described in Before you begin, below.

Before you begin

Before you begin this tutorial, make the following preparations:

- Complete the SilkTest tutorial to learn the basics of recording testcases, running testcases, and using the recovery system. The tutorial is located at **Start/Programs/Borland/SilkTest 2008/Documentation/SilkTest Tutorials**

Step 1: Use the Basic Workflow Bar to enable Java support

Java Support is configured automatically when you use Enable Extensions in the Basic Workflow bar.

The JFC Test Application is included with SilkTest.

Running the JFC test application:

> **Note** Borland no longer ships JRE 1.2 with SilkTest. In order to run the SilkTest sample Java applications, you may need to update the Java reference in the *.bat file that launches each sample application. If you do not have a local java.exe, you can download one from java.sun.com.

1. Install the applicable version JDK, if necessary.
2. If you have not done so already, update the sample application batch file (see “Update the sample Java application BAT file path” at the beginning of this document).
3. Select the Java JFC test application from **Start/Programs/Borland/SilkTest 2008/Sample Applications**.
4. Click **Enable Extensions** on the Basic Workflow bar, then select the sample application. SilkTest will prompt you to close and restart the sample application.

You are now ready to begin Step 2: Becoming familiar with the JFC Test Application.

Step 2: Becoming familiar with the JFC Test Application

This tutorial uses a simple Java application that provides JFC Swing controls. Explore the sample JFC Test Application now to become familiar with it. We are going to develop and run a test against the Page List window, so be sure to visit this part of the application, as instructed below.

To explore the sample application:

1. Start the application. See *Running the sample JFC applications* in the online Help if you need directions for this. The Test Application dialog opens with a menu bar containing three menus.
2. Click the **Control menu**, and then click **Page list**. The Page List window opens, displaying one page labeled Tab 1, which is selected. The window contains the following controls:
    - Area for adding pages and indicating which page is selected
    - Item Text text field where you specify a label for each page you add
    - Drop-down list for specifying where to display the page list (Top is the default)
    - Add Item, Reset, and Exit buttons
    - Add Image too and Enabled checkboxes
3. Click the **Item Text** field, enter a label for a new page, select **Right** from the drop-down list, and then click **Add Item**. Enter another label in the Item Text field, and select **Add Image too**, and then click **Add Item**.
4. Click **Exit** to close the Page List window and experiment with other controls accessible from the Control menu.
5. Explore other menus in the application's menu bar.

Continue to Step 3: Focusing on a part of the application to test.

Step 3: Focusing on a part of the application to test

Now that you have become familiar with the Page List window in the sample JFC Test Application, we are going to develop tests to verify that pages are added and deleted correctly.

We selected the Page List window as a test candidate because we will need to access a native method of the page list object to perform one of our tests. SilkTest predefines the JavaJFCPageList class to enable you to manipulate page lists using 4Test methods. However, there is no 4Test method for removing pages from the list, so we must record a new class definition for the page list to access its native method removeTabAt.

Continue to Step 4: Recording window declarations for the Page List window.

Step 4: Recording window declarations for the Page List window

Now we are going to record window declarations for all the controls in the Page List window.

Note At this point, we assume that the sample Java Swing application is still running.

To record window declarations:

1. Create a new test frame file. In SilkTest, choose **File/New/Test frame**, and then click **OK**. The New Test Frame dialog opens.
2. Edit the File name field so it reads: <*SilkTest installation directory*>/extend/pages.inc
3. Select the test application in the Application field, and then click **OK**. The 4Test include file pages.inc opens and is automatically loaded in SilkTest.
4. With pages.inc as the active window, choose **Record/Window Declarations**.
5. Click **Control/Page list** in the JFC Test Application, then move your mouse pointer over the title bar of the Page List window. When xPageList appears in the Identifier field, press Ctrl+Alt. Declarations are captured for the Page List window and all of its controls.
6. Change the name in the Identifier field from xPageList to PageListWindow.
7. Click **Paste to Editor** to record the declarations in pages.inc, then save pages.inc, but leave the file open.
8. Click **Close** to close the Record Window Declarations dialog, and then click **Exit** to close the Page List window.

Continue to Step 5: Preparing the test script file.

Step 5: Preparing the test script file

Before you can record the test, you must prepare the test script file.

> **Note** At this point, we assume that you recorded window declarations for Page List controls (and set up the recovery system if running Windows 98).

To prepare the test script file:

1. Close all secondary windows in the sample JFC Test Application so that only the main Test Application window is open.
2. Open a new test script. In the SilkTest menu bar, choose **File/New**, select **4Test script**, and then click **OK**. An untitled 4Test script file opens.
3. Save the test script file as pages.t in the extend subdirectory of the directory where you installed SilkTest.
4. With pages.t as the active window, choose **Record/Testcase** in the SilkTest menu bar.
5. Change the testcase name to **CheckAddPages** and select **DefaultBaseState** from the Application state list.

Now you are ready to record a test against the Page List window; continue to Step 6: Recording a test against Page List controls.

Step 6: Recording a test against Page List controls

Now that you have prepared your test script, you are ready to record actions for your test. Before you begin recording, expand this help topic window so you can see all instructions. If you need to scroll in this window after you start recording, click **Pause Recording** in the Record Status window so your scrolling actions do not get recorded as part of the test case.

This test will verify that the Add Item button adds the correct number of pages.

> **Note** At this point, we assume that you have created and saved your 4Test script file as pages.t. We also assume that the Record Testcase dialog is open, showing the testcase name as CheckAddPages and the application state as DefaultBaseState.

To record actions for verifying that Add Item adds the correct number of pages:

1. In the Record Testcase dialog, click **Start Recording**. The recovery system sets the DefaultBaseState application state for the sample Java application. Once the DefaultBaseState is established, the Record Status dialog opens and you can start recording your testcase.

2. Move your cursor into the main window of the sample application. When you see Test Application at the bottom of the Record Status window, click **Control** and move your cursor over **Page list** in the Control menu. When you see PageList in the Record Status window, click the **Page list** menu item. The Page List window opens, displaying one page called Tab 1.

3. Move your cursor over the Item Text field. When you see ItemText1 in the Record Status window, add a page by clicking in the **Item Text** field, typing **2**, and then clicking the **Add Item button**. A second page called 2 is added to the page list.

4. Move your cursor back to the Item Text field. When ItemText1 appears in the Record Status window, add another page by clicking just before the text **2** in the text field. Press the **Delete** key once, type **3**, and then click the **Add Item button** again. A third page called 3 is added to the page list.

5. Click **Tab 1** in the list of pages, wait until you see ThePageList in the Record Status window, and then press **Ctrl+Alt**. The Verify Window dialog opens; make sure that JavaJFCPageList appears as the window to verify.

6. In the Verify Window dialog, click the **Method** tab, then check **Include inherited** to display the 4Test methods available for the page list. Scroll down and select the method GetPageCount. Click **OK** to close the Verify Window dialog.

   We are going to use the GetPageCount method to verify that AddItem adds the correct number of pages to the page list. We started out with one page, then added two more, so the page count should equal 3.

7. Back in the Page List window, move your cursor over the **Exit** button. When Exit appears in the Record Status window, click the **Exit** button, and then click **Done** in the Record Status window. The Record Testcase window reappears, displaying your actions as translated into 4Test commands.

8. In the Record Testcase window, click **Paste to Editor**. The testcase CheckAddPages is pasted into pages.t.

9 Expand the testcase in pages.t by clicking in **CheckAddPages** and choosing **Outline/Expand All**.

10 Edit pages.t to verify the page count by wrapping the Verify function around the call to GetPageCount, as follows:

```
Verify ( PageListWindow.ThePageList.GetPageCount(), 3)
```

11 Save pages.t, but do not close it.

Now you are ready to run the recorded test; continue to Step 7: Running the recorded test against the sample JFC Test Application.

Step 7: Running the recorded test against the sample JFC Test Application

Note At this point, we assume that the test script file pages.t is open.

To run the testcase CheckAddPages:

1. With pages.t as the active window, choose **Run/Run**. SilkTest runs the test, restoring the sample JFC Test Application to its base state and interacting with the application. The testcase passes.
2. Close the results file, but leave pages.t open.

Now, let's create another testcase to verify that pages are deleted from the list correctly. Our next step will be to find a native method that allows us to delete pages from the page list; continue to Step 8: Getting native methods for a predefined JFC class.

Step 8: Getting native methods for a predefined Swing class

So far, we have verified that the Add Item button adds pages correctly to the page list. To extend this test, we should verify that we can also delete pages from the list. Unfortunately, the Page List window provides no controls for removing individual pages and there are no 4Test methods for deleting pages from a page list. Our only other option is to check out the native methods for the JavaJFCPageList class.

To get native methods for a predefined Java class, we must comment out the definition for JavaJFCPageList that is provided in javaex.inc and record the class definition for the Page List window.

> **Note** If you do not complete this tutorial, note that JavaEx.inc is a built-in SilkTest file. Therefore, you must uncomment the JavaJFCPageList class definition otherwise SilkTest will not recognize Page Lists in your application.

To get native methods for the JavaJFCPageList class:

1. With the sample JFC Test Application running and not minimized, open the file **javaex.inc** in SilkTest. Javaex,inc is located in <*SilkTest install directory*>\extend. Find the line that reads:

   winclass JavaJFCPageList : PageList

2. Click anywhere on the line. If the declaration is expanded, collapse it by choosing **Outline/Collapse**. Comment out the collapsed line by choosing **Outline/Comment Block**. Then, save and close javaex.inc.

   Remember to uncomment these lines when you finish this tutorial.

3. Open your test frame file **pages.inc** and click in the existing page list declaration—the line that reads

   window JavaDialogBox PageListWindow

4. Collapse this declaration if expanded and then comment it out by choosing **Outline/Comment Block**.

5. Scroll to the bottom of pages.inc and create a new section by adding the following comment:

   //Native Page List Declarations

6. With pages.inc as the active window, choose **Record/Class**. The Record Class dialog opens.

7. Make sure **Show all classes** is not checked in the Record Class dialog.

8. In the JFC Test Application, click **Control/Page list** to open the Page List window.

9. Move your mouse cursor over Tab 1 of the page list. When JavaJFCPageList appears in the Class Name field, press **Ctrl+Alt**. Native methods and properties for the page list appear in the Record Class dialog.

10 Scroll in the Methods pane until you find the method removeTabAt. We will use this method later when we create a script that tests whether pages are removed correctly from the Page List window.

11 Click the **Derived From** drop-down menu, and then select **PageList** from the list of available 4Test classes.

12 Click **Paste to Editor** to record the class declaration in the Native Page List Declarations section that you just created in pages.inc.

13 Click **Close** to close the Record Class dialog, and save pages.inc.

Now you must record new declarations for the Page List window; continue to Step 9: Recording new window declarations for the Page List window.

Step 9: Recording new window declarations for the Page List window

Note At this point, we assume that pages.inc is still open and that the Page List window is still open in the sample Java Swing application.

To re-record window declarations for the Page List window:

1. With pages.inc as the active window, choose **Record/Window Declarations**.
2. Move your mouse pointer over the title bar of the Page List window in the JFC Test Application. When xPageList appears in the Identifier field, press **Ctrl+Alt**. Declarations are captured for the Page List window and all of its controls.
3. Change the name in the Identifier field from xPageList to **PageListNative**.
4. Click **Paste to Editor** to record the declarations in pages.inc.
5. Click **Close** to close the Record Window Declarations dialog.
6. Click **Exit** to close the Page List window.
7. Save and close pages.inc.

Now, you are ready to create a new testcase that uses the native method removeTabAt to verify that we can remove pages from a page list; continue to Step 10: Developing a new test using a native method.

Step 10: Developing a new test using a native method

We are going to create a new script for testing whether we can delete pages from the page list in our sample Swing application.

This script uses native methods—including removeTabAt—along with 4Test methods. Normally, we do not recommend mixing method types because of incompatibilities between Java and 4Test. For example, Java indexing is zero-based while 4Test indexing is one-based. However, in some situations, you must mix native Java methods with 4Test methods to achieve the functionality you require. In this script, we will demonstrate one way to protect against incompatible indexing.

> **Note** At this point, we assume pages.t is still open and the Swing test application is still running.

To develop a new test using native methods:

1. In pages.t, collapse the CheckAddPages testcase, select the testcase by clicking in the margin to the left of the testcase, and choose **Edit/Copy** from the menu bar.
2. Open a new untitled test script file by choosing **File/New/4Test script**, and then click **OK**.
3. Click in the new script file, and then choose **Edit/Paste** to paste a copy of the CheckAddPages testcase.
4. Change the name of the testcase to **CheckDeletePages** and expand the testcase.
5. Replace all instances of PageListWindow with **PageListNative** to match the identifier in our new Page List window declaration in pages.inc.
6. Before the Verify function, insert a line that calls the native method, removeTabAt, to delete the second page in the list. Here is the 4Test code (which you can copy and paste into your test script):

   ```
   PageListNative.ThePageList.removeTabAt(PageListNative.ThePageList.indexOfTab("2
   "))
   ```

   > **Note** We pass the native method indexOfTab as an argument to removeTabAt to ensure that we provide the correct zero-based index value for page 2.

7. To see how the Page List window adjusts the list after one page is removed, add a Select method on the next line to refresh the Page List window. Here is the 4Test code (which you can copy and paste into your test script):

   ```
   PageListNative.ThePageList.Select("3")
   ```

8. Change the second argument in the Verify function from 3 to **2**, since we removed one page from the list.
9. Save the test script as **pages2.t** in the extend subdirectory of the directory where you installed SilkTest.

Now, you are ready to run the test CheckDeletePages; continue to Step 11: Running the test that uses native methods.

Step 11: Running the test that uses native methods

Note At this point, we assume that the test script file pages2.t is open.

To run the testcase CheckDeletePages:

With pages2.t as the active window, choose **Run/Run**. SilkTest runs the test, restoring the sample JFC Test Application to its base state and interacting with the application. You will notice that Tab 2 is deleted and, then, Tab 3 is selected. The testcase passes.

Congratulations! You have completed the Java JFC/Swing tutorial. Next, you might want to explore *Overview of Java Support* in the online Help.