Borland®
(A MICRO FOCUS COMPANY)

MICRO FOCUS®

SilkTest 13.0

SilkTest Classic: Testing Java Applications with the Classic Agent

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa, CA 92626

2012-05-07

Contents

Introduction

The tutorials included in this document are designed to introduce you to using SilkTest Classic and the Classic Agent to test stand-alone Java applications developed with the Abstract Window Toolkit (AWT) and the Java Foundation Class (JFC). The tutorials use the following sample Java applications:

- AWT Test Application
- JFC Test Application

Download and install the sample applications from *http://supportline.microfocus.com/websync/SilkTest.aspx*. After you have installed the sample applications, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** and select the Java application that you want to use.

Updating the Java Reference of the Sample Java Applications

SilkTest Classic no longer supports JRE 1.2. In order to run theSilkTest Classic 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 *http://www.oracle.com/technetwork/java/index.html*.

To update the Java references:

1. Use Windows Explorer to navigate to `<SilkTest installation directory\JavaEx`.
2. Open `AWT_TestApplication.bat` with a text editor, for example Notepad.
3. Set `JavaRun=` to the directory in which `java.exe` is installed. For example `JavaRun=C:\jdk1.7\bin`.
4. Save your changes.

   You can start the sample application by double-clicking `AWT_TestApplication.bat`, or by clicking **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java AWT** > **Classic Agent** > **AWT Test Application**.
5. Use Windows Explorer to navigate to `<SilkTest installation directory\JavaEx\JFC`.
6. Open `JFC_TestApplication.bat` with a text editor, for example Notepad.
7. Set `JavaRun=` to the directory in which `java.exe` is installed. For example `JavaRun=C:\jdk1.7\bin`.
8. Save your changes.

   You can start the sample application by double-clicking `JFC_TestApplication.bat`, or by clicking **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java AWT** > **Classic Agent** > **JFC Test Application**.

Java AWT Tutorial

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

While working through this tutorial, you will perform the following tasks:

1. Set up for testing the application.
2. Become familiar with the sample Java AWT application.
3. Focus on a part of the application under test (AUT).
4. Identify custom controls and learn when to record classes.
5. Record classes for custom Java controls.
6. Record window declarations for all controls that 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, learn the basics of recording test cases, running test cases, and using the recovery system. You can access the SilkTest Classic tutorials by clicking **Start** > **Programs** > **Silk** > **SilkTest** > **Documentation** > **SilkTest Classic** > **Tutorials** or by clicking **HelpTutorials** in SilkTest Classic.

Step 1: Enabling Java Support in the Basic Workflow Bar

When you enable extensions in the **Basic Workflow** bar, Java support is configured automatically.

To run the AWT test application:

1. Install the applicable version JDK, if necessary.
2. If you have not done so already, update the Java reference.
   For additional information, see *Updating the Java Reference of the Sample Java Applications*.
3. Click **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** and select the Java AWT test application.
4. Click **Enable Extensions** on the **Basic Workflow** bar, then select the sample application. SilkTest Classic will prompt you to close and restart the sample application.

Step 2: Becoming Familiar with the Sample Java AWT Application

The sample Java AWT application used in this tutorial is a simple Java application that provides a mix of predefined AWT controls along with custom Java objects. For additional information about the AWT, refer to the *SilkTest Classic Help*. Explore the sample Java application to become familiar with it.

1. Click **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Java AWT** > **AWT Test Application**. The **Test Application** dialog box 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 inside the canvas and drag the mouse to draw points and lines. The mouse actions are recorded in the event log.
4. Click **Exit** to close the **Drawing Area** window.
5. Experiment with other controls accessible from the **Control** menu.
6. Explore other menus in the menu bar of the application.

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

Step 3: Focusing On a Part Of the Sample Java AWT Application

Now that you have become familiar with the **Drawing area** in the sample Java AWT application, 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 AWT controls and a custom Java object. SilkTest Classic 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.

Step 4: Identifying Custom Controls in the Drawing Area Window

SilkTest Classic provides 4Test class definitions for commonly used Java objects, such as AWT controls. With the Java support of SilkTest Classic, 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 additional information about the AWT or predefined Java classes, refer to the *SilkTest Classic Help*.

The SilkTest Classic Java support also enables 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.

To identify the custom controls in the sample Java AWT application:

1. Click **Control** > **Drawing area** in the sample Java AWT application. The **Drawing Area** window opens.
2. In SilkTest Classic, click **Record** > **Window Declarations**. The **Record Window Declarations** dialog box opens.
3. Move the mouse pointer inside the **Event Log**, which is the scrollable white rectangular area.

   SilkTest Classic recognizes the **Event Log** as a `JavaAwtTextField` object, which is a predefined AWT control.
4. Move the mouse pointer over the **Reset** button, then over the **Exit** button.

   SilkTest Classic recognizes these buttons as `JavaAwtPushButton` objects, which are predefined AWT controls.
5. Move the mouse pointer inside the drawing canvas, which is the gray rectangular area. SilkTest Classic recognizes the drawing canvas as a `CustomWin` object, which is a custom Java control.
6. Click **Close** to close the **Record Window Declarations** dialog box.

Step 5: Recording a Class for the Drawing Area Canvas

Before you can record a class for the canvas, the **Drawing Area** window needs to be open in the sample Java AWT application.

To record a class for the canvas in the **Drawing Area** window of the sample Java AWT application:

1. Create a new include file. In SilkTest Classic, click **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. Click **Options** > **Recorder**, uncheck **Record multiple tags**, and then click **OK**.
4. With **canvas.inc** as the active window, click **Record** > **Class** > **Scripted**. The **Record Class** dialog box opens.
5. In the **Record Class** dialog box, uncheck **Show all classes**.
6. Move your mouse pointer over the drawing canvas in the sample Java application.
7. Ctrl+Alt.When `DrawingCanvas` displays in the **Class Name** field, press **Ctrl+Alt**. Native methods for the drawing canvas display in the **Record Class** dialog box.
8. Search the **Derived From** 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 box.

Step 6: Recording Window Declarations for the Drawing Area

Before you can record window declarations, you need to have recorded a class for the canvas, and the **Drawing Area** window needs to be open in the sample Java AWT application.

To record window declarations:

1. Create a new test frame file. In SilkTest Classic, click **File** > **New** > **Test frame**, and then click **OK**. The **New Test Frame** dialog box opens.
2. In the **Application** field, select `Test Application`.
3. In the **Frame filename** field, type `<SilkTest install directory>\extend\draw.inc`.

   For example, if SilkTest Classic is installed in `C:\Program Files (x86)\Silk\SilkTest`, type `C:\Program Files (x86)\Silk\SilkTest\extend\draw.inc`.
4. Click **OK**. The 4Test include file `draw.inc` opens and is automatically loaded in SilkTest Classic.
5. Manually load the class include file `canvas.inc`, which you have previously created.
   a) In SilkTest Classic, click **Options** > **Runtime**. The **Runtime Options** dialog box opens.
   b) In the **Use files** field, click **Browse** and select `canvas.inc` from the `extend` directory.
   c) Click **Open**. The `canvas.inc` file is added to the **Use Files** field.
6. Click **OK** to close the **Runtime Options** dialog box.
7. With `draw.inc` as the active window, click **Record** > **Window Declarations**.
8. Move your mouse pointer over the title bar of the **Drawing Area** window. When *DrawingArea* displays 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 box.
12. Click **Exit** to close the **Drawing Area** window.
13. Save and close `draw.inc`.

Step 7: Preparing the Test Script File

Before you can prepare the test script file, you need to have recorded window declarations for **Drawing Area** controls, have loaded `draw.inc` and `canvas.inc`, and have 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 Classic menu bar, click **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, click **Record** > **Testcase** in the SilkTest Classic menu bar. The **Record Testcase** dialog box opens.
5. Change the **Testcase name** to `LogMouseMoves`.
6. In the **Application state** list box, select **DefaultBaseState**.

Step 8: Recording a Test Against Drawing Area Controls

Before you can record actions, you need to have created your 4Test script file and have saved it as `draw.t`. You also need to have the **Record Testcase** dialog box open, showing the test case name as *LogMouseMoves* and the application state as *DefaultBaseState*.

Now that you have prepared your test script, you are ready to record actions for your 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.

To record actions for verifying the **Event Log** function:

1. In the **Record Testcase** dialog box, 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 box opens and you can start recording your test case.
2. Move your cursor into the main window of the sample application. When you see `Test Application` at the bottom of the **Record Status** dialog box, click **Control** > **Drawing area**.
3. Click once inside the drawing canvas, which is 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 box opens. Make sure that `JavaAwtTextField DrawingArea.EventLog` displays 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`.
8. In the **The string or list of string you expect selected** field, type `"Button down"`, including the quotation marks.
9. Click **OK**.
10. Repeat steps 4 through 9 for the second mouse action, `Button up`.

11. Move your cursor over **Exit** in the **Drawing Area** window. When `Exit` displays in the **Record Status** window, click **Exit**, and then click **Done** in the **Record Status** window. The **Record Testcase** window displays, displaying your actions as translated into 4Test commands.
12. In the **Record Testcase** window, click **Paste to Editor**. The test case *LogMouseMoves* is pasted into `draw.t`.
13. Expand the test case by clicking in `LogMouseMoves`, and then click **Outline** > **Expand All**.
14. Save `draw.t`, but do not close it.

Step 9: Running the Recorded Test Against the Sample Java AWT Application

To run the test case *LogMouseMoves*:

1. If the test script file `draw.t` is not already open, open the file.
2. Set keyboard and mouse delays.
   a) In the SilkTest Classic menu bar, click **Options** > **Agent**. The **Agent Options** dialog box opens.
   b) In the **Agent Options** dialog box, set keyboard event delay and mouse event delay both to 0.01.
   c) Click **OK**.
3. With **draw.t** as the active window, click **Run** > **Run**. SilkTest Classic runs the test, restoring the sample AWT application to its base state and interacting with the application. The test case passes.

Step 10: Extending the Test Programmatically

The sample Java application must be running and not minimized, and the test script file `draw.t` must be open.

So far, you should have verified that the **Event Log** records the `Button down` and `Button up` mouse events correctly when you click once to draw a single point in the drawing canvas. To extend this test, you 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, you will now create two integer variables to store the X and Y coordinates of a point. In addition, you will set each variable to a two-digit random number within a range of values that falls inside the drawing canvas.

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.
   a) In the sample Java AWT application, click **Control** > **Drawing area**.
   b) In SilkTest Classic, click **Record** > **Actions**.
   c) Click the title bar of the **Drawing area** window and move your mouse pointer into the drawing canvas.
   d) When you see `Press <Ctrl+Alt> to verify window Canvas` display in the **Record Actions** dialog box, press **Ctrl+Alt** to bring up the **Verify Window**. The values for the `Rect` property display in the **Properties to Verify** window. For example, 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.
   e) Click **Cancel** to close the **Verify Window**.
   f) Click **Close** to close the **Record Actions** window.
   g) Click **Exit** to close the **Drawing Area** window.

2. Expand the test case `LogMouseMoves` in your `draw.t` file.
   a) 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. For additional details about the `RandInt` function, refer to the *SilkTest Classic Help*.

      For example, set iX to a random number between 10 and 99 and iY to a random number between 10 and 60. You can copy the following sample code into your test script. Substitute different numbers in the `RandInt` function calls, if necessary.

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

   b) 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. You can copy the following sample code into your test script.

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

   c) After the `VerifySelText ("Button up")` command, add 4Test code to select and verify the two-digit X coordinate and the two-digit Y coordinate of the mouse event recorded as the `Button Down…` text string in the **Event Log**.

      You can copy the following sample code into your test script.

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

      In this code, the selection range of the X and Y coordinates in SetSelRange is determined by counting text characters from left to right in the string `Button Down at (xx,yy)`. Random integers are forced to be two-digit numbers so the selection range will have a constant start and end value for each coordinate.

      

      Note: The `str` function is used to convert integer variables to string arguments that can be passed to the `VerifySelText` commands.
3. Save `draw.t`, but do not close it.

Step 11: Running the Extended Test

To run the test case `LogMouseMoves`:

1. Open the script file `draw.t` and set it as the active window.
2. Click **Run** > **Run**.

   SilkTest Classic runs the test, restoring the sample Java application to its base state and interacting with the application. The test case passes.

Congratulations! You have completed the Java tutorial. For additional information about Java support in SilkTest Classic, refer to the *SilkTest Classic Help*.

Java JFC/Swing Tutorial

This tutorial is designed to help you get comfortable with testing a sample stand-alone Java application developed with JFC and Swing controls, using Java support in SilkTest Classic. For additional information on JFC and Swing controls, refer to the *SilkTest Classic Help*.

While working through this tutorial, you will perform the following tasks:

1. Set up for testing the application.
2. Become familiar with the JFC test application.
3. Focus on a part of the application under test (AUT).
4. Record window declarations for all controls that 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.

Before you begin this tutorial, learn the basics of recording test cases, running test cases, and using the recovery system. You can access the SilkTest Classic tutorials by clicking **Start** > **Programs** > **Silk** > **SilkTest** > **Documentation** > **SilkTest Classic** > **Tutorials** or by clicking **HelpTutorials** in SilkTest Classic.

Step 1: Enabling Java Support in the Basic Workflow Bar

When you enable extensions in the **Basic Workflow** bar, Java support is configured automatically.

To run the JFC test application:

1. Install the applicable version JDK, if necessary.
2. If you have not done so already, update the Java reference.

   For additional information, see *Updating the Java Reference of the Sample Java Applications*.
3. Click **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** and select the Java JFC test application.
4. Click **Enable Extensions** on the **Basic Workflow** bar, then select the sample application. SilkTest Classic will prompt you to close and restart the sample application.

Step 2: Becoming Familiar with the Sample JFC Test Application

The JFC test application used in this tutorial is a simple Java application that provides JFC and Swing controls. For additional information about the JFC and Swing, refer to the *SilkTest Classic Help*. Explore the sample Java application to become familiar with it.

1. Click **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Java JFC** > **JFC Test Application**. The **Test Application** dialog box 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:
   - An area for adding pages and indicating which page is selected.
   - An **Item Text** field where you specify a label for each page that you add.
   - A list box from which you can select where to display the page list. Top is the default.
   - **Add Item**, **Reset**, and **Exit** buttons.
   - **Add Image too** and **Enabled** check boxes.
3. Click in the **Item Text** field and add a label for a new page.
4. In the list box, select **Right** and then click **Add Item**.
5. Type another label in the **Item Text** field.
6. Check the **Add Image too** check box.
7. Click **Add Item**.
8. Click **Exit** to close the **Page List** window.
9. Experiment with other controls accessible from the **Control** menu.
10. Explore other menus in the menu bar of the application.

Step 3: Focusing On a Part of the Sample Java JFC Application

Now that you have become familiar with the **Page List** window in the JFC test application, develop a test 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 Classic 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 the native method `removeTabAt`.

Step 4: Recording Window Declarations for the Page List Window

The JFC test application needs to be running before you can record window declarations.

To record window declarations:

1. Create a new test frame file. In SilkTest Classic, click **File** > **New** > **Test frame**, and then click **OK**. The **New Test Frame** dialog box opens.
2. In the **Application** field, select `Test Application`.
3. In the **Frame filename** field, type `<SilkTest install directory>\extend\pages.inc`.

   For example, if SilkTest Classic is installed in `C:\Program Files (x86)\Silk\SilkTest`, type `C:\Program Files (x86)\Silk\SilkTest\extend\pages.inc`.
4. Click **OK**. The 4Test include file `pages.inc` opens and is automatically loaded in SilkTest Classic.
5. With `draw.inc` as the active window, click **Record** > **Window Declarations**.
6. In the JFC test application, click **Control** > **Page list**.
7. Move your mouse pointer over the title bar of the **Page List** window. When *xPageList* displays in the **Identifier** field, press **Ctrl+Alt**. Declarations are captured for the **Page List** window and all of its controls.
8. Change the name in the **Identifier** field to `PageListWindow`.
9. Click **Paste to Editor** to record the declarations in `pages.inc`.

10. Save `pages.inc`, but leave the file open.
11. Click **Close** to close the **Record Window Declarations** dialog box.
12. Click **Exit** to close the **Page List** window.

Step 5: Preparing the Test Script File (JFC)

Before you can prepare the test script file, you need to have recorded window declarations for **Page List** controls.

To prepare the test script file:

1. Close all secondary windows in the JFC test application so that only the main **Test Application** window is open.
2. Open a new test script. In the SilkTest Classic menu bar, click **File** > **New** > **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 have installed SilkTest Classic.
4. With `pages.t` as the active window, click **Record** > **Testcase** in the SilkTest Classic menu bar. The **Record Testcase** dialog box opens.
5. Change the **Testcase name** to `CheckAddPages`.
6. In the **Application state** list box, select **DefaultBaseState**.

Step 6: Recording a Test Against Page List Controls

Before you can record actions, you need to have created your 4Test script file and have saved it as `pages.t`. You also need to have the **Record Testcase** dialog box open, showing the test case name as *CheckAddPages* and the application state as *DefaultBaseState*.

Now that you have prepared your test script, you are ready to record actions for your 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.

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

1. In the **Record Testcase** dialog box, 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 box opens and you can start recording your test case.
2. Move your cursor into the main window of the sample application. When you see `Test Application` at the bottom of the **Record Status** dialog box, click **Control** and move your cursor over **Page list** in the **Control** menu.
3. When you see `PageList` in the **Record Status** dialog box, click the **Page list** menu item. The **Page List** window opens, displaying one page called *Tab 1*.
4. Move your cursor over the **Item Text** field. When you see `ItemText1` in the **Record Status** dialog box, add a page by clicking in the **Item Text** field, typing 2, and then clicking **Add Item**. A second page called *2* is added to the page list.
5. Move your cursor back over the **Item Text** field. When `ItemText1` displays in the **Record Status** dialog box, add another page by clicking just before the text 2 in the field. Press **Delete**, type 3, and then click **Add Item** again. A third page called *3* is added to the page list.
6. Click **Tab 1** in the list of pages, wait until you see `ThePageList` in the **Record Status** dialog box, and then press **Ctrl+Alt**. The **Verify Window** dialog box opens; make sure that `JavaJFCPageList` displays as the window to verify.

7. Click the **Method** tab, and then check **Include inherited** to display the 4Test methods available for the page list.
8. Scroll down and select **GetPageCount**.

   You 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.
9. Click **OK** to close the **Verify Window** dialog box.
10. In the **Page List** window, move your cursor over the **Exit** button. When `Exit` displays in the **Record Status** dialog box, click the **Exit** button, and then click **Done** in the **Record Status** dialog box. The **Record Testcase** window opens, displaying your actions as translated into 4Test commands.
11. In the **Record Testcase** window, click **Paste to Editor**. The test case *CheckAddPages* is pasted into `pages.t`.
12. Click in `CheckAddPages`, and then click **Outline** > **Expand All** to expand the test case.
13. Edit `pages.t` to verify the page count by wrapping the `Verify` function around the call to `GetPageCount`.

    ```
    Verify ( PageListWindow.ThePageList.GetPageCount(), 3)
    ```
14. Save `pages.t`, but do not close it.

Step 7: Running the Recorded Test Against the JFC Test Application

To run the test case *CheckAddPages*:

1. If the test script file `pages.t` is not already open, open the file.
2. With **pages.t** as the active window, click **Run** > **Run**. SilkTest Classic runs the test, restoring the JFC test application to its base state and interacting with the application. The test case passes.
3. Close the results file, but leave **pages.t** open.

Step 8: Getting Native Methods for a Predefined Swing Class

So far, you have verified that the **Add Item** button adds pages correctly to the page list. To extend this test, you should verify that you 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: `JavaEx.inc` is a built-in SilkTest Classic file. If you do not complete this tutorial, you must un-comment the `JavaJFCPageList` class definition otherwise SilkTest Classic will not recognize page lists in your application.

To get native methods for the `JavaJFCPageList` class:

1. With the JFC test application running and not minimized, open the file `javaex.inc` in SilkTest Classic. `javaex,inc` is located in `<SilkTest install directory>\extend`. Find the line that reads:

   ```
   winclass JavaJFCPageList : PageList
   ```
2. Click anywhere in the line. If the declaration is expanded, click **Outline** > **Collapse** to collapse it.

3. Click **Outline** > **Comment Block** to comment out the collapsed line.

   Remember to un-comment these lines when you finish this tutorial.
4. Save and close **javaex.inc**.
5. Open your test frame file `pages.inc` and click in the existing page list declaration, which is the following line:

   ```
   window JavaDialogBox PageListWindow
   ```
6. Collapse this declaration if expanded and then click **Outline** > **Comment Block** to comment out the collapsed line.
7. Scroll to the bottom of `pages.inc` and create a new section by adding the following comment:

   ```
   //Native Page List Declarations
   ```
8. With **pages.inc** as the active window, click **Record** > **Class**. The **Record Class** dialog box opens.
9. In the **Record Class** dialog box, uncheck **Show all classes**.
10. In the JFC test application, click **Control** > **Page list** to open the **Page List** window.
11. Move your mouse cursor over **Tab 1** of the page list. When `JavaJFCPageList` displays in the **Class Name** field, press **Ctrl+Alt**. Native methods and properties for the page list display in the **Record Class** dialog box.
12. Scroll in the **Methods** pane until you find the method `removeTabAt`. You will use this method later when you create a script that tests whether pages are removed correctly from the **Page List** window.
13. From the list of available 4Test classes in the **Derived From** list box, select `PageList`.
14. Click **Paste to Editor** to record the class declaration in the **Native Page List Declarations** section that you just created in `pages.inc`.
15. Click **Close** to close the **Record Class** dialog box.
16. Save `pages.inc`.

Step 9: Recording New Window Declarations for the Page List Window

The include file `pages.inc` must be open in SilkTest Classic and the **Page List** window must be open in the JFC test application.

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

1. Set **pages.inc** as the active window in SilkTest Classic.
2. Click **Record** > **Window Declarations**.
3. Move your mouse pointer over the title bar of the **Page List** window in the JFC test application. When `xPageList` displays in the **Identifier** field, press **Ctrl+Alt**. Declarations are captured for the **Page List** window and all of its controls.
4. Change the name in the **Identifier** field from `xPageList` to `PageListNative`.
5. Click **Paste to Editor** to record the declarations in `pages.inc`.
6. Click **Close** to close the **Record Window Declarations** dialog box.
7. Click **Exit** to close the **Page List** window.
8. Save and close `pages.inc`.

Step 10: Creating a New Test Using a Native Method

The include file `pages.inc` must be open in SilkTest Classic and the JFC test application must be running.

You are going to create a new script for testing whether you can delete pages from the page list in the JFC test 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. This script includes one way to protect against incompatible indexing.

To develop a new test using native methods:

1. In `pages.t`, collapse the `CheckAddPages` test case.
2. Select the test case by clicking in the margin to the left of the test case, and click **Edit** > **Copy** in the menu bar.
3. Click **FileNew4Test script** to open a new untitled test script file and then click **OK**.
4. Click in the new script file, and then click **Edit** > **Paste** to paste a copy of the `CheckAddPages` test case.
5. Change the name of the test case to `CheckDeletePages` and expand the test case.
6. Replace all instances of `PageListWindow` with `PageListNative` to match the identifier in the new page list window declaration in `pages.inc`.
7. Before the `Verify` function, insert a line that calls the native method `removeTabAt` to delete the second page in the list.

   You can copy and paste the following 4Test code into your test script:

   ```
   PageListNative.ThePageList.removeTabAt(PageListNative.ThePageList.indexOfTa
   b("2"))
   ```

   Note: The native method `indexOfTab` is passed as an argument to `removeTabAt` to ensure that the correct zero-based index value for page 2 is provided.
8. 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. You can copy and paste the following 4Test code into your test script:

   ```
   PageListNative.ThePageList.Select("3")
   ```
9. Change the second argument in the `Verify` function from 3 to 2, since you have removed one page from the list.
10. Save the test script as `pages2.t` in the `extend` subdirectory of the directory where you have installed SilkTest Classic.

Step 11: Running the Test that Uses Native Methods

To run the test case `CheckDeletePages`:

1. Open the script file `pages2.t` and set it as the active window.
2. Click **Run** > **Run**. SilkTest Classic runs the test, restoring the JFC test application to its base state and interacting with the application. The test case passes.

Congratulations! You have completed the Java JFC/Swing tutorial. For additional information about Java support in SilkTest Classic, refer to the *SilkTest Classic Help*.

Index

A

B

C

E

F

J

R

S

T