Borland
A MICRO FOCUS COMPANY

Silk Test 13.5

Testing Flex Applications

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

2012-09-19

Contents

Overview of Adobe Flex Support

The tasks described in this tutorial are supported only for the Open Agent.

Silk Test Classic provides built-in support for testing Adobe Flex applications using the 4Test scripting language. You can also test Adobe Flex applications with the Silk4J Eclipse plug-in using the Java programming language. Silk4J is an optional program. During installation, you must specify that you want to install Silk4J in order to use it.

Silk Test Classic supports Adobe Flex applications using Windows Internet Explorer, Mozilla Firefox, or the standalone Flash Player, and Adobe AIR applications built with Adobe Flex 4 or later.

Silk Test Classic also supports multiple application domains in Adobe Flex 3.x and 4.x applications, which enables you to test sub-applications. Silk Test Classic recognizes each sub-application in the locator hierarchy tree as an application tree with the relevant application domain context. At the root level in the locator attribute table, Adobe Flex 4.x sub-applications use the `SparkApplication` class. Adobe Flex 3.x sub-applications use the `FlexApplication` class.

For additional information about the supported versions for Adobe Flex and potential known issues, refer to the *Release Notes*.

Sample Applications

Silk Test Classic provides several sample Adobe Flex test applications. You can use these sample applications to record tests with Silk Test Classic and Silk4J.

You can access the Adobe Flex sample applications at *http://demo.borland.com/flex/SilkTest13.5/index.html*.

The chapter *Silk4J Quick Tour* uses the Component Explorer sample application to walk you through testing an Adobe Flex application. However, if you prefer, you can use your own Adobe Flex application to perform these steps. If you use your own application, follow the steps in the *Enabling Your Adobe Flex Application for Testing* chapter before you begin the *Silk4J Quick Tour*.

Enabling Your Adobe Flex Application for Testing

This section describes how you can use the Adobe Flex Automation API to prepare your Adobe Flex application for automation testing using Silk Test Classic. Adobe Flex developers are the target audience for this document.

For additional information about which version of Adobe Flex to use and which browsers and operating environments are supported for testing, refer to the *Release Notes*.

To enable your Adobe Flex application for testing, you must include the following components in your application:

- Adobe Flex Automation Package
- Silk Test Automation Package

You can load these packages at run time or prior to run time by pre-compiling your application. When you load the automation packages at run time, your application is not modified. However, this method is difficult to use in applications that are tested in a Web browser. For details about these limitations, see *Limitations*. In contrast, pre-compiling your application modifies your application and increases the file size, which means that you must create two builds, one for testing and one for release. However, this method works for all applications.

For additional information about the differences between the run time and pre-compiled approaches, refer to the Adobe guideline at *http://download.macromedia.com/pub/documentation/en/flex/2/at_api.pdf*.

Note: If you are using an Adobe Flex sample application, you do not need to perform these steps. The sample applications have already been enabled for testing.

Loading Automation Packages at Run Time

You can load automation support at run time using the Silk Test Flex Automation Launcher. This application is compiled with the automation libraries and loads your application with the `SWFLoader` class. This automatically enables your application for testing without compiling automation libraries into your SWF file. The Silk Test Flex Automation Launcher is available in HTML and SWF file formats.

Limitations

When you load automation support at run time, the following limitations apply:

- The Silk Test Flex Automation Launcher automatically becomes the root application. If your application must be the root application, you cannot load automation support with the Silk Test Flex Automation Launcher. For other options, see *Compiling Automation Packages Prior to Run Time*.
- Applications that load external libraries, which means other SWF file libraries, require a special setting for automated testing. A library that is loaded at run time, including run-time shared libraries (RSLs), must be loaded into the ApplicationDomain of the loading application. If the SWF file used in the application is loaded in a different application domain, automated testing record and playback will not function properly. The following example shows a library that is loaded into the same ApplicationDomain:

```
import flash.display.*;
import flash.net.URLRequest;
import flash.system.ApplicationDomain;
import flash.system.LoaderContext;
```

```
var ldr:Loader = new Loader();
var urlReq:URLRequest = new
URLRequest("RuntimeClasses.swf");
var context:LoaderContext = new LoaderContext();
context.applicationDomain =
ApplicationDomain.currentDomain;
loader.load(request, context);
```

Using the Flex Automation Launcher for Run-Time Loading

For additional information about creating events and custom controls to support automated testing, refer to *Testing Flex Custom Controls* in the *Silk Test Classic Help*.

1. Copy the content of the directory `Silk\Silk Test\ng\AutomationSDK\Flex\<VERSION>\FlexAutomationLauncher` into the directory of the Adobe Flex application that you are testing.
2. Open `FlexAutomationLauncher.html` in Windows Explorer and add the following parameter as a suffix to the file path:

   ```
   ?automationurl=YourApplication.swf
   ```

   where `YourApplication.swf` is the name of the `.swf` file for your Adobe Flex application.
3. Add `file:///` as a prefix to the file path.

   For example, if your file URL includes a parameter, such as `?automationurl=explorer.swf`, type `file:///C:/Program%20Files/Silk/SilkTest/ng/samples/Flex/3.2/FlexControlExplorer32/FlexAutomationLauncher.html?automationurl=explorer.swf`.

Compiling Automation Packages Prior to Run Time

You can pre-compile applications that you plan to test. The functional testing classes are embedded in the application at compile time, and the application has no external dependencies for automated testing at run time.

When you embed functional testing classes in your application SWF file at compile time, the size of the SWF file increases. If the size of the SWF file is not important, use the same SWF file for functional testing and deployment. If the size of the SWF file is important, generate two SWF files, one with functional testing classes embedded and one without. Use the SWF file that does not include the embedded testing classes for deployment.

When you pre-compile the Adobe Flex application for testing, in the include-libraries compiler option, reference the following files:

- `automation.swc`
- `automation_agent.swc`
- `FlexTechDomain.swc`
- `automation_dmv.swc`. Include if your application uses charts.
- `automation_flasflexkit.swc`. Include if your application uses embedded flash content.
- `automation_spark.swc`. Include if your application uses Adobe Flex 4 controls.
- `automation_air.swc`. Include if your application is an AIR application.
- `automation_airspark.swc`. Include if your application is an AIR application and uses Adobe Flex 4 controls.

When you create the final release version of your Adobe Flex application, you recompile the application without the references to these SWC files. For more information about using the automation SWC files, refer to the *Adobe Flex Release Notes*.

If you do not deploy your application to a server, but instead request it by using the file protocol or run it from within Adobe Flex Builder, you must include each SWF file in the local-trusted sandbox. This requires additional configuration information. Add the additional configuration information by modifying the configuration file of the compiler or by using a command-line option.

Modifying the Configuration File of the Compiler

For additional information about creating events and custom controls to support automated testing, refer to *Testing Flex Custom Controls* in the *Silk Test Classic Help*.

1. Include the libraries `automation.swc`, `automation_agent.swc`, and `FlexTechDomain.swc` in the configuration file of the compiler by adding the following code to the configuration file:

```
<include-libraries>
...
<library>/libs/automation.swc</library>
<library>/libs/automation_agent.swc</library>
<library>pathinfo/FlexTechDomain.swc</library>
</include-libraries>
```

   Note: If your application uses charts, you must also add the `automation_charts.swc` file to the include-libraries compiler option.

2. Specify the location of the `automation.swc`, `automation_agent.swc`, and `FlexTechDomain.swc` libraries using the include-libraries compiler option with the command-line compiler

   The configuration files are located at:

Product	File Location
Adobe Flex SDK	`<flex_installation_directory>/frameworks/flex-config.xml`
Adobe Flex Data Services	`<flex_installation_directory>/flex/WEB-INF/flex/flex-config.xml`

   The following example adds the files `automation.swc` and `automation_ agent.swc` to the application:

```
mxmlc -include-libraries+=../frameworks/libs/
automation.swc;../frameworks/libs/
automation_agent.swc;pathinfo/FlexTechDomain.swc
MyApp.mxml
```

   Note: Explicitly setting the include-libraries option on the command line overwrites, rather than appends, the existing libraries. If you add the `automation.swc` and `automation_agent.swc` files using the include-libraries option on the command line, ensure that you use the += operator. This appends rather than overwrites the existing libraries that are included.

   Note: The Silk Test Flex Automation SDK is based on the Automation API for Flex. The Silk Test Automation SDK supports the same components in the same manner that the Automation API for Flex supports them. For instance, when an application is compiled with automation code and successive .swf files are loaded, a memory leak occurs and the application eventually runs out of memory. The Flex Component Explorer sample application is affected by this issue. The workaround is to not compile the application .swf files that Explorer loads with automation libraries. For example, compile only the Explorer main application with automation libraries. Another alternative is to use the module loader instead of swfloader. For more information about using the Flex Automation API, refer to the *Adobe Flex Release Notes*.

Testing Flex Applications

This section guides you through the steps of testing an Adobe Flex application. The procedures in this section use the Component Explorer sample application. However, if you have another Flex application that you prefer to use, enable your Adobe Flex application for testing and then follow the steps in this section. The target audience for this chapter are Quality Assurance testers.

For information about supported versions of Adobe Flex and the supported browsers and operating environments for testing, refer to the *Release Notes*.

Configuring the Security Settings for Your Local Flash Player

Before you launch an Adobe Flex application that runs as a local application for the first time, you must configure security settings for your local Flash Player. You must modify the Adobe specific security settings to enable the local application access to the file system.

To configure the security settings for your local Flash player:

1. Open the **Flex Security Settings** page by clicking **Flash Player Security Manager** on *http://demo.borland.com/flex/SilkTest13.5/index.html*.
2. Click **Always allow**.
3. From the **Edit Locations** list box, select **Add Location**.
4. Click **Browse for folder** and navigate to the folder where your local application is installed.
5. Click **Confirm** and then close the browser.

Starting the Component Explorer

Compiled with the Adobe Automation SDK and the Silk Test specific automation implementation, the Component Explorer is pre-configured for testing.

To start the Component Explorer in Windows Internet Explorer, open *http://demo.borland.com/flex/SilkTest2011/3.5/Flex3TestApp_withAutomation/Flex3TestApp.html*.

Creating a New Project

To test the sample Adobe Flex application, begin by creating a new project.

To create a project:

1. In Silk Test Classic, click **File** > **New Project**, or click **Open Project** > **New Project** on the Basic Workflow bar. The **New Project** dialog box opens.
2. Under **Rich Internet Applications**, click **Adobe Flex**.
3. Click **OK**. The **Create Project** dialog box opens.
4. Type the **Projectname** and **Description**.
5. *Optional:* Your new project is saved in the default location, `<SilkTest Installation Directory>\Projects`. If you do not want to save your project in the default location, click **Browse** and navigate to the folder in which you want to save your project.

6. Click **OK**.

Silk Test Classic creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources that are associated with this project.

Configuring an Adobe Flex Application

Launch the Component Explorer before you perform this step.

Configure the Adobe Flex application to set up the environment that Silk Test Classic will create each time you record or replay a test case.

When you configure an application, Silk Test Classic automatically creates a base state for the application. The base state of an application is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution.

1. In Silk Test Classic, click **Configure Applications** on the Basic Workflow bar.

   This functionality is available only with the Open Agent.

   The **New Test Frame** dialog box opens.
2. Double-click **Web Site Test Configuration**. The **New Web Site Configuration** page opens.
3. From the **Browser Type** list, select **Internet Explorer**.

   You can use one of the other supported browsers to replay tests but not to record them.
4. Perform one of the following steps:

   - **Use existing browser** – Click this option button to use a browser window that is already open to configure the test. For example, if the Web page that you want to test is already displayed in the browser window, you might want to use this option.
   - **Start new browser** – Click this option button to start a new browser instance to configure the test. Then, in the **Browse to URL** text box, specify the Web page to open.
5. Click **Finish**. The **Choose name and folder of the new frame file** page opens. Silk Test Classic configures the recovery system and names the corresponding file `frame.inc` by default.
6. Navigate to the location in which you want to save the frame file.
7. In the **File name** field, type the name for the frame file that contains the default base state and recovery system.
8. Click **Save**.

   When you configure an application, Silk Test Classic adds an include file based on the technology or browser type that you enable to the **Use files** location in the **Runtime Options** dialog box. For instance, if you configure an Adobe Flex application, a file named `flex.inc` is added.

   Silk Test Classic opens the include file.

Recording a Test Case

A test case:

- Drives the application from the initial state to the state you want to test.
- Verifies that the actual state matches the expected (correct) state.
- Cleans up the application, in preparation for the next test case, by undoing the steps performed in the first stage.

To record a test case:

1. Click **Record Testcase** on the Basic Workflow bar. The **Record Testcase** dialog box opens.

2. Type the name of your test case into the **Testcase name** field.

   Test case names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
3. From the **Application State** list box, select **DefaultBaseState** to have the built-in recovery system restore the default base state before the test case begins executing.

   The test case is recorded in the script file as `testcase testcase_name ()`.
4. Click **Start Recording**. Silk Test Classic closes the **Record Testcase** dialog box and displays the Flex Component Explorer application.
5. When the **Record Status** window opens, record the following scenario using the Flex Component Explorer application.
   a) Click the arrow next to the **Visual Components** tree element to expand the list.
   b) Click the arrow next to the **General Controls** tree element to expand the list.
   c) Click the **SimpleAlert** tree element.
   d) Point to the **SimpleAlert** tree and press **Ctrl+Alt** to add a verification to the script.

      You can add a verification for any of the information that displays.

      The **Verify Properties** dialog box opens.
   e) Check the **Visible** check box.
   f) Click **OK**. A verification action is added to the script for the tree.
   g) In the **Alert Control Example** section, click **Click Me** and then click **OK** in the **Hello World** message box.
   h) Click the arrow next to the **General Controls** tree element to hide the list.
   i) Click the arrow next to the **Visual Components** tree element to hide the list.
6. In the **Recording status** window, click **Stop Recording**.

   Silk Test Classic opens the **Record Testcase** dialog box, which contains the recorded 4Test code.
7. Click **Paste to Editor**. The **Update Files** dialog box opens.
8. Choose **Paste testcase and update window declaration(s)** and then click OK.
9. Click **File** > **Save**.
10. Specify the file name and location.
11. When Silk Test Classic prompts you to add the file to the project, click **Yes**.

Replaying a Test Case

When you run a test case, Silk Test Classic interacts with the application by executing all the actions that you have specified in the test case and testing whether all the features of the application performed as expected.

To run a test case:

1. Set the test case that you want to run as the active window in Silk Test Classic.
2. Click **Run Testcase** on the Basic Workflow bar. Silk Test Classic displays the **Run Testcase** dialog box, which lists all the test cases contained in the current script.
3. Select the test case that you have created.
4. Check the **Animated Run Mode (Slow-Motion)** check box, to wait one second after each script line is executed.

   Typically, you will only use this check box if you want to watch the test case run. For instance, if you want to demonstrate a test case to someone else, you might want to check this check box.
5. Click **Run**. Silk Test Classic runs the test case and generates a results file. The results file describes whether the test passed or failed, and provides summary information.

Silk4J Quick Tour

Silk4J provides a Java run time library that includes test classes for all the classes that Silk4J supports for testing. This run time library is compatible with JUnit, which means you can leverage the JUnit infrastructure and run and create tests using JUnit. You can also use all available Java libraries in your test cases.

This quick tour describes how to use the Adobe Flex sample scripts provided with Silk4J. Use the sample script files in the sample application to view typical script configurations and test case execution.

For a Getting Started tutorial for Silk4J, refer to the *Silk4J Quick Start Tutorial*.

Importing Silk4J Sample Scripts

Silk4J can use the sample Flex applications and provides several Adobe Flex scripts that work with Silk4J. Import the Adobe Flex scripts to view how Silk4J works with Adobe Flex.

1. In the Eclipse workspace, click **File** > **Import**. The **Import wizard** opens.
2. In the menu tree, click the plus sign (+) to expand the **General** folder and click **Existing Projects into Workspace**.
3. Click **Next**. The **Import Projects** dialog box opens.
4. In the **Select root directory** field, click **Browse**. The **Browse For Folder** dialog box opens.
5. Navigate to the folder `Documents and Settings\All Users\Shared Documents\Silk Test \samples\Silk4J`.

   For Microsoft Windows Vista and Microsoft Windows 7 operating systems, the file location is `Users \Public\Documents\Silk Test\samples\Silk4J\`.
6. Select the root directory for the sample and then click **OK**.
7. In the **Projects** section, select the project that you want to import and then click **Finish**. The sample project shows in the **Package Explorer** view.

Running a Sample Silk4J Test Case

Before you run a test case, start the Open Agent.

Use the sample test cases that Silk4J provides to view script contents and test execution results.

For detailed procedures about creating Silk4J scripts, refer to the *Silk4J User Guide*.

1. Navigate to the Adobe Flex sample project.
2. Choose one of the following:

   - Right-click the package name to run all tests in the project. For example, right-click **com.borland.flex.store**.
   - Right-click the class name to run all tests for only that class. For example, right-click **FlexStoreTest.java** in the **com.borland.flex.store** package.
3. Click **Run As** > **JUnit Test**. The test results display in the JUnit view as the test runs. If all tests pass, the status bar is green. If one or more tests fail, the status bar is red. You can click a failed test to display the stack trace in the **Failure Trace** area.

Index