Testing Flex 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

Contents

Introduction

Introduction

This chapter provides an overview of Adobe Flex support for SilkTest and Silk4J, the limitations for Flex support, and the sample applications provided with SilkTest.

What you will learn

This book contains the following chapters:

Overview of Adobe Flex Support

SilkTest 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.

Limitations for Flex

The following limitations exist when using Adobe Flex with SilkTest.

- The following commands are not supported:
  - Record Window Locations
  - Record Class
  - Record Method
  - Record Defined Window

- When you record actions, the actions are not shown immediately. Recorded actions are shown when recording stops.
- The recovery system for browsers is limited in the following ways:
  - URLs must be entered as parameter for the browser in the command line string.
  - SilkTest does not identify the browser window as a browser window. Therefore, the browser class functions are unavailable.
- Functions/classes in the **winclass.inc** file that are marked with the "supported_ca" keyword are supported on the Classic Agent only.
- Certain functions and methods run on the Classic Agent only. When these are recorded and replayed, they default to the Classic Agent automatically. You can use these in an environment that uses the Open Agent. SilkTest will automatically use the appropriate Agent. The functions and methods include:
  - SYS functions (except for SYS_GetDir, which is supported on the Open Agent)
  - C data types for use in calling functions in DLLs
  - ClipboardClass Class methods
  - CursorClass Class methods

For details about known Flex issues, refer to the *Release Notes*.

Sample Applications

SilkTest provides several sample Adobe Flex test applications. You can use these sample applications to record tests with SilkTest and Silk4J. To access the samples, choose **`Start/Programs/Borland/SilkTest <version>/ Sample Applications/Adobe Flex/Flex Sample Applications`** and choose the sample application that you want to use.

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

1

Enabling Your Flex Application for Testing

Introduction

This chapter details how to use the Adobe Flex Automation API to prepare your Flex application for automation testing using SilkTest. Flex developers are the target audience for this document.

For information about which version of Flex to use and which browsers and operating environments are supported for testing, refer to the *SilkTest 2008 Release Notes*.

Enabling Your Application for Testing

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

- "Adobe Flex Automation Package"
- "SilkTest Automation Package"

You can load these packages at run time or prior to run time by precompiling 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" on page 8. In contrast, precompiling 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 more details about the differences between the run time and precompiled approaches, refer to the Adobe guideline at http://download.macromedia.com/pub/documentation/en/flex/2/at_api.pdf.

Note If you are using a 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 SilkTest 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 SilkTest Flex Automation Launcher is available in HTML and SWF file formats.

Limitations

- The Flex Automation Launcher Application automatically becomes the root application. If your application must be the root application, you cannot load automation support with the SilkTest Flex Automation Launcher. For other options, see "Compiling Automation Packages Prior to Run Time" on page 9.
- Testing applications that load external libraries – Applications that load 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);
```

To use the Flex Automation Launcher for run-time loading

1. Copy the content of the **Borland\SilkTest\ng\AutomationSDK\Flex\<VERSION>\FlexAutomationLauncher** directory into the directory of the Flex application that you are testing.
2. Open **FlexAutomationLauncher.html** in a browser and add the following parameter to the URL:

   `?automationurl=YourApplication.swf`

   where `YourApplication.swf` is the name of the .swf file for your Flex application.

For details about creating events and custom components to support automated testing, refer to "Instrumenting Flex Events and Components" in the SilkTest online help.

Compiling Automation Packages Prior to Run Time

You can precompile 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 precompile the Flex application for testing, in the `include-libraries compiler` option, reference the following files:

- **automation.swc**
- **automation_agent.swc**
- **FlexTechDomain.swc**
- **automation_charts.swc (include only if your application uses charts and Flex 2.0)**
- **automation_dmv.swc (include only if your application uses charts and Flex 3.0)**
- **automation_flasflexkit.swc (include only if your application uses embedded flash content)**

When you create the final release version of your Flex application, you recompile the application without the references to these SWC files. For more information about using the automation SWC files, see 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 compiler's configuration file or using a command-line option.

Modifying the Compiler's Configuration File to Add Configuration Information

1 Include the **automation.swc**, **automation_agent.swc**, and **FlexTechDomain.swc** libraries in the compiler's configuration file 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 2 SDK	<***flex_installation_directory***>/**frameworks/flex-config.xml**
Adobe Flex Data Services	<***flex_installation_directory***>/**flex/WEB-INF/flex/flex-config.xml**

Table 1: Configuration file locations

The following example adds the automation.swc and automation_agent.swc files 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 SilkTest Flex Automation SDK is based on the Automation API for Flex. The SilkTest 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 runs out of memory eventually. The Flex ControlExplorer 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, see the *Adobe Flex Release Notes*.

For details about creating events and custom components to support automated testing, refer to “Instrumenting Flex Events and Components” in the SilkTest online help.

2

Tutorial for Flex Application Testing

Introduction

This chapter guides you through the steps of testing an Adobe Flex application. Quality Assurance testers are the target audience for this chapter.

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

What you will learn

This chapter contains the following sections:

Prerequisites for Testing Our Adobe Flex Applications

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 security settings for your local Flash player

1. Open the Flex Security Settings Page by choosing **Start/Programs/Borland/SilkTest <version>/Sample Applications/Adobe Flex/Flex Security Settings**.
2. Click **Always allow**.
3. From the **Edit Locations** drop-down menu, click **Add Location**.

4 From the **Edit Locations** drop-down menu, click **Add Location**.

5 Click **Browse for folder** and navigate to the folder where your local application is installed.

The Control Explorer sample application is located in the **<*SilkTest_Install_Directory*>\ng\samples\flex** folder on your local drive. By default, the **<*SilkTest_Install_Directory*>** is located at **Program Files\Borland\SilkTest**.

6 Click **Confirm** and then close the browser.

SilkTest Tutorial

This tutorial explains each of the steps involved in testing an Adobe Flex application with SilkTest. These procedures use the Control Explorer sample application. However, if you have another Flex application that you prefer to use, make sure that the steps in the previous chapter have been completed and then follow these steps.

The steps include:

- "Launching the Control Explorer"
- "Creating a New Project"
- "Enabling Extensions"
- "Setting the Recovery System"
- "Recording a Testcase"
- "Replaying a Testcase"
- "Verifying Object Properties"

After the tutorial, the "Testing a Flex Sample Application Using a Dual Agent Approach" on page 23 details a more complex series of steps for testing an Adobe Flex application that is hosted as part of a web page. This procedure uses

the Flex sample application web page and the Control Explorer sample application.

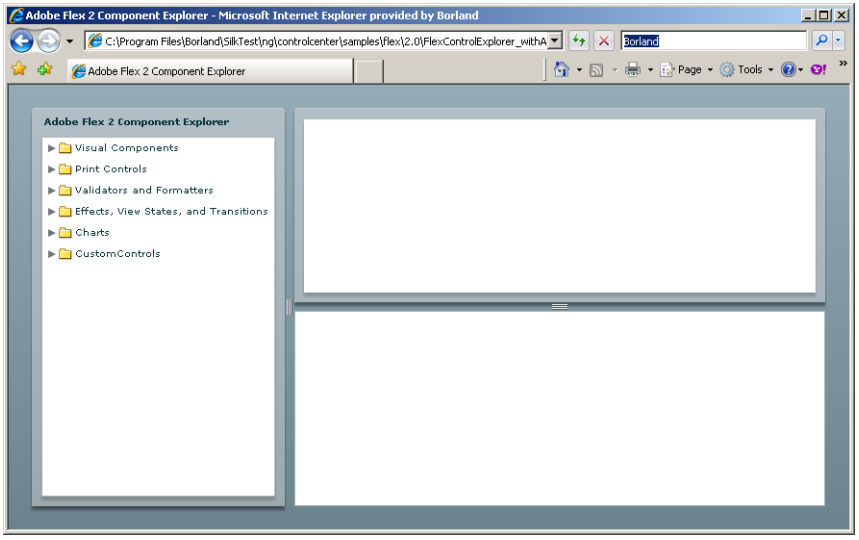
Launching the Control Explorer

Compiled with the Adobe Automation SDK and the SilkTest specific automation implementation, the Control Explorer is preconfigured for testing.

To launch the Control Explorer

- Choose **Start>Programs>Borland>SilkTest <version>>Sample Applications>Adobe Flex>Flex <version>>Flex Control Explorer>Control Explorer with Automation**.

The application launches in your default browser and looks similar to the following image:



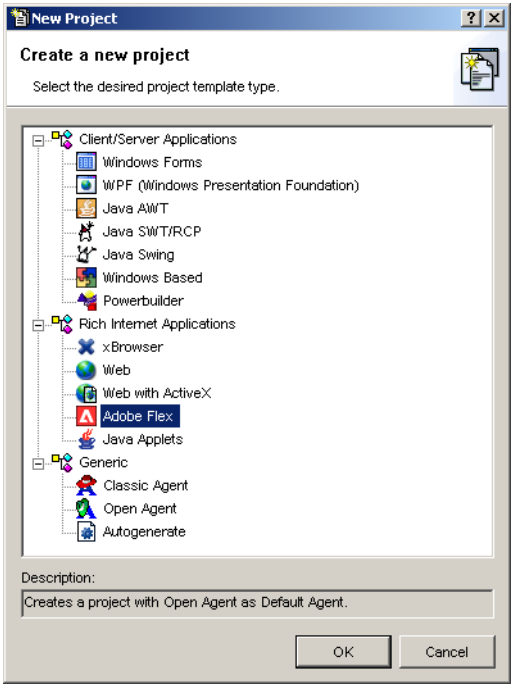
Creating a New Project

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

To create a project

1 In SilkTest, choose **File/New Project**, or click **Open Project/New Project** on the Basic workflow bar.

2 On the New Project dialog, under Rich Internet Applications, click **Adobe Flex**.



3 Click **OK**.

4 On the Create Project dialog, type the **Projectname** and **Description**.

5 Click **OK** to save your project in the default location, ***<SilkTest_Install_Directory>*\Projects**.

SilkTest will create a projectname folder within this directory, save the projectname.vtp and projectname.ini to this location and copy the extension .ini files (appexpex.ini, axext.ini, domex.ini, and javaex.ini) to a **projectname\extend** subdirectory. If you do not want to save your project in the default location, click Browse and specify the folder in which you want to save your project.

6 SilkTest creates your project and displays nodes on the Files and Global tabs for the files and resources associated with this project.

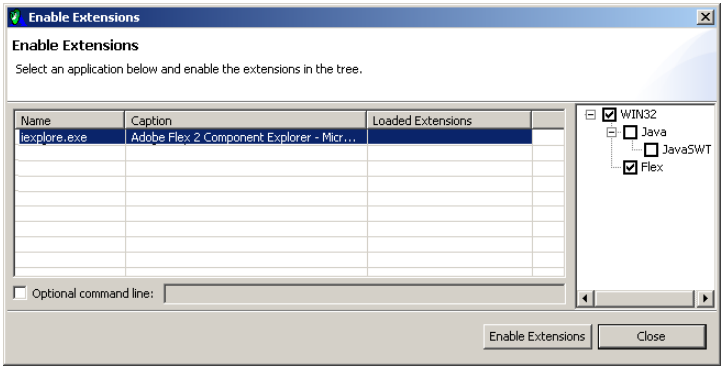
Enabling Extensions

An extension is a file that serves to extend the capabilities of, or data available to, a more basic program. SilkTest provides extensions for testing applications that use non-standard controls in specific development and browser environments.

Launch the Control Explorer before you perform this step.

To enable the extensions for the browser

1. In the SilkTest Basic Workflow bar, click **Enable Extensions**.
2. Select your browser from the list of running processes and ensure that SilkTest automatically checked the Flex check box.



3. Click **Enable Extensions**.

   After a few moments, Win32 and Flex display in the Loaded Extensions column in the dialog.
4. Click **Close**.

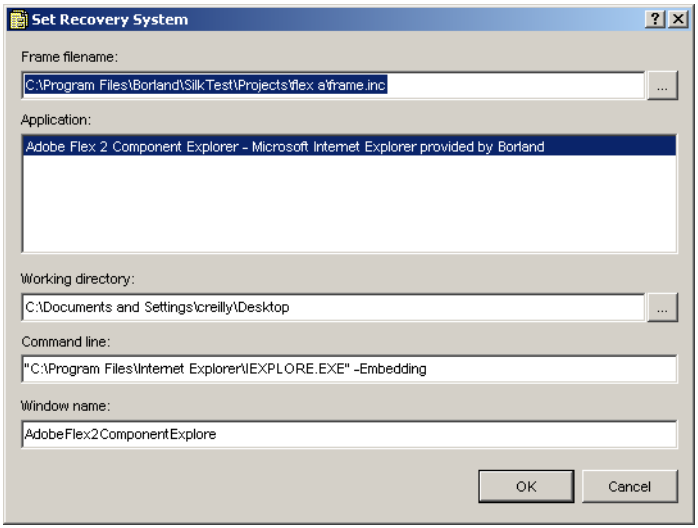
Setting the Recovery System

The recovery system ensures that each testcase begins and ends with the application in its intended state. SilkTest refers to this intended application state as the *base state*.

To set the recovery system

1. Make sure the sample Control Explorer application is running.

2 Click **Set Recovery System** on the Basic Workflow bar.



3 From the Application list, click the name of the Adobe Flex application that you are testing. Only applications that have extensions enabled are displayed in the Application list.

Notice that:

- the Command line field displays the path to the executable (.exe) for the Flex sample application.
- the Working directory field displays the path of the Flex sample application you selected.

4 Optionally, in the Frame filename field, modify the frame file name and click Browse to specify the location in which you want to save this file. Frame files must have a .inc extension. By default, this field displays the default name and path of the frame file you are creating. The default is frame.inc. If frame.inc already exists, SilkTest appends the next logical number to the new frame file name; for example, frame1.inc.

5 Optionally, in the Window name field, change the window name to use a short name to identify your application.

6 Click **OK**.

7 Click **OK** when the message indicating that the recovery system is configured appears.

8 A new 4Test include file, frame.inc, opens in the SilkTest Editor. The frame.inc file is automatically added to your project and includes the information about the selected application.

Recording a Testcase

A testcase:

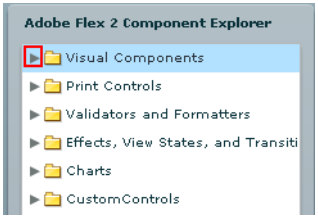
- Drives the application from the initial state to the state you want to test.
- Verifies that the actual state matches the expected (correct) state.
- Cleans up the application, in preparation for the next testcase, by undoing the steps performed in the first stage.

To record a testcase

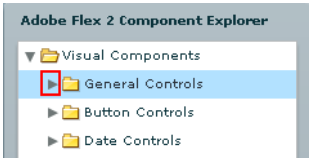
1. Click **Record Testcase** on the Basic Workflow bar.
2. In the Record Testcase dialog, type the name of your testcase in the Testcase name field. Testcase names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
3. Select **DefaultBaseState** in the **Application state** field to have the built-in recovery system restore the default BaseState before the testcase begins executing.
4. Click **Start Recording**. SilkTest closes the Record Testcase dialog and displays the Flex Control Explorer application.
5. When the Record Status window opens, record the following scenario using the Flex Control Explorer application.

   Note It is essential that you perform these steps exactly how they are documented. Otherwise, your testcase script may not match the sample provided later in this document.

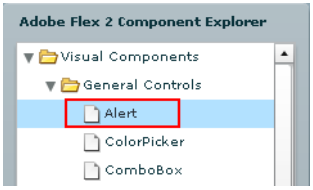
   a. Click the ▶ arrow next to the Visual Components tree element to expand the list.



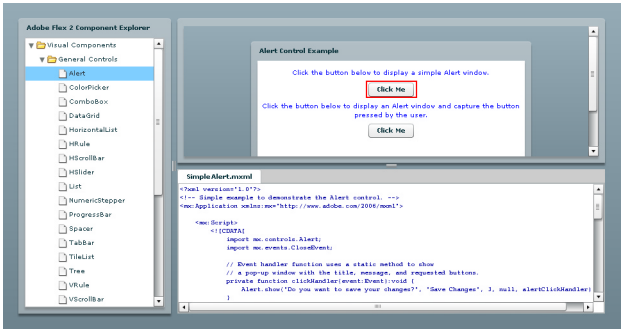
b Click the ▶ arrow next to the General Controls tree element to expand the list.



c Click the Alert tree element.



d In the Alert Control Example section, click the **Click Me** button and then click **OK** in the Hello World message box.



e Click the ▼ arrow next to the General Controls tree element to hide the list.

f Click the ▼ arrow next to the Visual Components tree element to hide the list.

6 In the Record Status window, click **Done**.

7 SilkTest opens the Record Testcase dialog, which contains the 4Test code that has been recorded for you.

8 Click **Paste to Editor**.

9 On the Update Files dialog, select **Paste testcase and update window declaration(s)** and then click **OK**.

SilkTest will create window declarations for new objects and use the new identifiers in the resulting testcase.

10 Choose **File/Save**.

11 Specify the file name and location.

12 When SilkTest prompts you to add the file to the project, click **Yes**.

Your testcase should include the following calls:

```
testcase Test1 ()
    recording
        AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Index0.AdobeFlex2ComponentExplore.CompLibTree.Select("Visual Components>General Controls>Alert")
        AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Index0.Index1.SwfLoader.ControlsSimpleAlertSwf.AlertControlExample.ClickMe.Click()
        AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Message.OK.Click()
        AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Index0.AdobeFlex2ComponentExplore.CompLibTree.Close("Visual Components>General Controls")
        AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Index0.AdobeFlex2ComponentExplore.CompLibTree.Close("Visual Components")
```

Replaying a Testcase

When you run a testcase, SilkTest interacts with the application by executing all the actions you specified in the testcase and testing whether all the features of the application performed as expected.

To run a testcase

1 Make sure that the testcase you want to run is in the active window.

2 Click **Run Testcase** on the Basic Workflow bar.

3 SilkTest displays the Run Testcase dialog, which lists all the testcases contained in the current script.

4 Select a testcase and specify arguments, if necessary, in the Arguments field. Remember to separate multiple arguments with commas.

5 Click **Run**. SilkTest runs the testcase and generates a results file.

Verifying Object Properties

Using the Verification wizard you can add verification statements to your scripts using the user interface. Or, you can manually customize your scripts using the Verify function on Flex object properties. Each Flex object has a list of properties that you can verify. For a list of the properties available for verification, review the **extend\Flex\Flex.inc** file.

Perform these steps after you record a testcase.

Using the Verification Wizard

To use the Verification wizard

1 Choose **Record/Actions**.

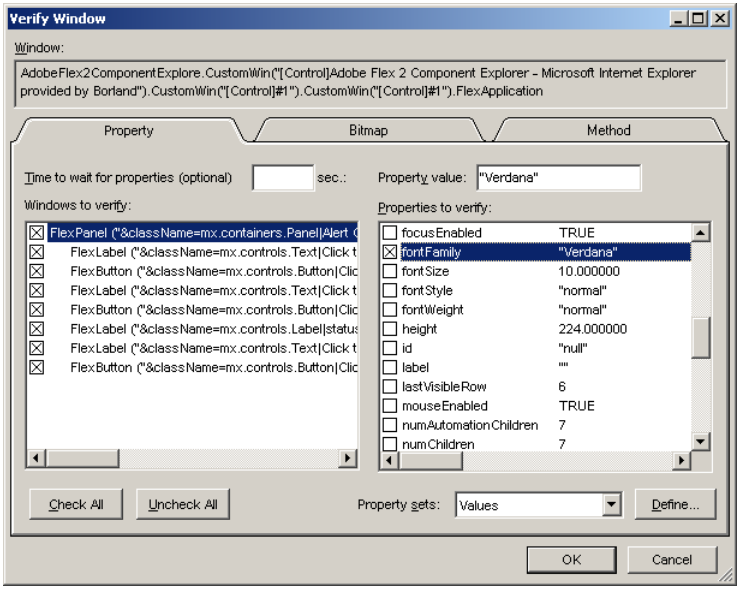
2 Drive your application to the state that you want to verify and position the mouse cursor over the object. For this example, perform step 12 a) - d) from “Recording a Testcase” on page 19.

3 Position your cursor in the Alert Control Example pane and press Ctrl+Alt.

The Verify Window dialog appears over your application window. The Window field, in the top-left corner of the dialog, displays the name of the object you were pointing at when you pressed Ctrl+Alt.

> **Note** If the name in the Window field is incorrect, click Cancel to close the dialog and return to the application. Point to the object you want to verify and press Ctrl+Alt again.

4 On the Property tab, all objects in the Windows to Verify column are checked by default. In the **Properties to Verify** column, check the **fontFamily** check box.

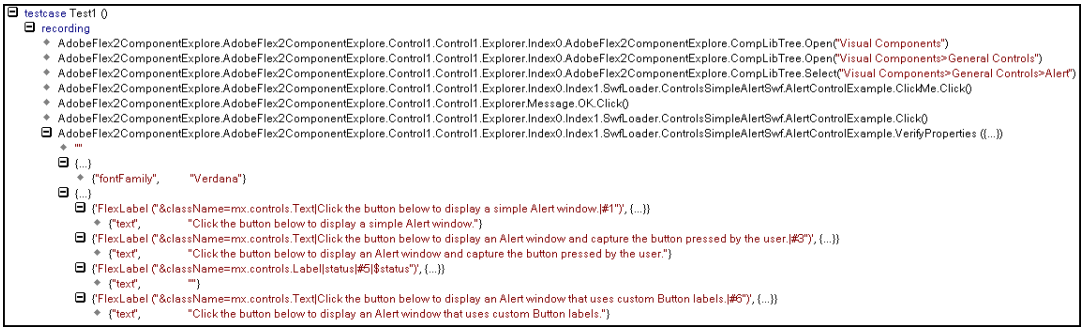


5 Click **OK**.

6 In the Record Actions on Open Agent dialog, click **Copy to Clipboard** and then click **Close**.

7 In the testcase file, navigate to the place in the file where you want to add the verification step and press Ctrl+V.

This is what your script should look like:



8 Save the file.

9 Run the testcase.

Manually Customizing Your Script

To manually customize your script

- Open the script file and insert the code that you want to add.

  For example, you can add the following call to your script:

```
AdobeFlex2ComponentExplore.AdobeFlex2ComponentExplore.Control1.Control1.Explorer.Index0.Index1.SwfLoader.ControlsSimpleAlertSwf.AlertControlExample.fontFamily("Verdana")
```

Testing a Flex Sample Application Using a Dual Agent Approach

To test a Flex application that is hosted as part of a web page or that is only accessible by browsing through individual web pages first, configure SilkTest to use the Classic and Open Agent. Both Agents must be configured to test the same browser. Therefore, you must enable extensions twice – once for the Classic Agent and once for the Open Agent. Use the Classic Agent for the recovery system and to interact with the web elements of the application that you want to test, for example, clicking links, entering web-form data, and so on. Then, test the Flex application with the Open Agent, which provides built-in support for the Adobe Flex technology.

This procedure details the steps required to set up SilkTest to use a dual Agent approach to test a single application.

To test a Flex Sample Application that uses the SilkTest Classic and Open Agents

1. Open your web application or choose **Start/Programs/Borland/SilkTest <version>/Sample Applications/Adobe Flex/Flex Sample Applications** to use the Flex sample application web page.
2. In SilkTest, create a new project to test the sample web application using the SilkTest Classic Agent. By default, web applications use the Classic Agent. You want to use the Classic Agent as the default Agent because the default base state for your application will launch a web application.
3. Enable extensions for the web application automatically using the Basic Workflow.
4. Set the recovery system.
5. Choose **Record/Application State** to record an application state that is based on the DefaultBaseState. For this scenario, click a link on the Flex Sample Overview page for the Flex sample application that you want to test. For example, click the Control Explorer Automation Ready link.

   Note Make sure that you click **Paste appstate and update window declaration(s)** when you paste the application state into the Editor. SilkTest includes the application state in the frame.inc file. By default, the application state is called State1 (unless you have created other application states for the project.)
6. From the main toolbar, click  to use the Open Agent.
7. Enable extensions for the Flex application automatically using the Basic Workflow.
8. Choose **Options/Agent** and increase the Window timeout value on the Timing tab.
9. Click **Record Testcase** in the Basic Workflow bar.
10. From the **Application state** list, select **State1** or the name that the application state from step 5 was saved as.
11. Click **Start Recording** to record the testcase to test the Flex controls on the web page.
12. Click **Done** when you are finished.
13. When prompted, click **Paste to Editor** and then click **Paste testcase and update window declaration(s)**.

    Note SilkTest automatically detects which Agent is required for each test based on the window declaration and changes the Agent accordingly.

14 Choose **File/Save** to save the testcase.

15 Click **Run Testcase** in the Basic Workflow bar to replay the testcase.

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 testcases.

This quick tour explains 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 testcase execution.

This quick tour includes:

- "Importing Silk4J Sample Scripts"
- "Running a Sample Testcase"

For a Getting Started tutorial for Silk4J, refer to the Silk4J Eclipse help.

Importing Silk4J Sample Scripts

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

1 In the Eclipse workspace, choose **File > Import**. The Import wizard opens.

2 In the menu tree, click the plus sign (+) to expand the **General** folder and select **Existing Projects into Workspace**.

3 Click **Next**. The Import Projects dialog opens.

4 In the **Select root directory** field, click **Browse**. The Browse For Folder dialog opens.

5 Navigate to the <silktest_install_directory>\ng\samples\flex\Silk4j_AutomatedTests_Flex folder. By default, the sample files are installed in: C:\Program Files\Borland\SilkTest\ng\samples\flex\Silk4j_AutomatedTests_Flex.

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 Testcase

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

Before you run a testcase, start the SilkTest Open Agent.

1. Navigate to the Flex sample project.
2. Choose one of the following:
   - **a** Right-click the test class package name to run all tests in the project.

     For example, right-click com.borland.flextestapp.test.
   - **b** Right-click the test class class name to run a test for only that class.

     For example, right-click FlexSampleApplicationTest.java in the com.borland.flextestapp.test package.
3. Choose **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.

For detailed procedures about creating Silk4J scripts, refer to the Silk4J Eclipse help.