Borland®
A MICRO FOCUS COMPANY

Silk Test 14.0

Silk4J User Guide

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

2013-05-28

Contents

Welcome to Silk4J

Welcome to Silk4J

About Silk4J
Product Suite

What's new

Release Notes

Featured sections

Best Practices for Using Silk4J
Creating Test Methods
Testing Specific Environments

Tutorials and demonstrations

Quick Start Tutorial

Code samples

Enhancing Tests

Online resources

Borland Home Page
Borland Channel on YouTube
Online Documentation
Micro Focus SupportLine
Micro Focus Product Updates
Silk Test Knowledge Base
Silk Test Forum
Micro Focus Training Store

Provide feedback

Contacting Micro Focus on page 145
Email us feedback regarding this Help

Licensing Information

Unless you are using a trial version, Silk Test requires a license.

The licensing model is based on the client that you are using and the applications that you want to able to test. The available licensing modes support the following application types:

Licensing Mode	Application Type
Web	Web applications, including Java-Applets.
Web plus Flex	Web applications, including the following: • Adobe Flex • Java-Applets
Full	• Web applications, including the following: • Adobe Flex • Java-Applets • Adobe Flex • Java AWT/Swing • Java SWT and Eclipse RCP • .NET, including Windows Forms and Windows Presentation Foundation (WPF) • Rumba • Windows API-Based **Note:** To upgrade your license to a Full license, visit *www.borland.com*.
Premium	All application types that are supported with a *Full* license, plus SAP applications. **Note:** To upgrade your license to a Premium license, visit *www.borland.com*.

Silk4J

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

The testing environments that Silk4J supports include:

- Adobe Flex
- Java AWT/Swing
- Java SWT
- Rumba
- SAP
- Microsoft Silverlight
- Windows API-based client/server (Win32)
- Windows Forms
- Windows Presentation Foundation (WPF)
- xBrowser (Web applications)

You can find sample scripts for Web application testing in the public `Documents` folder, under `%PUBLIC%\Documents\SilkTest\samples\Silk4J`.

Silk Test Product Suite

The Silk Test product suite includes the following components:

- Silk Test Workbench – Silk Test Workbench is the native quality testing environment that offers .NET scripting for power users and easy to use visual tests to make testing more accessible to a broader audience.
- Silk4NET – The Silk4NET Visual Studio plug-in enables you to create Visual Basic or C# test scripts directly in Visual Studio.
- Silk4J – The Silk4J Eclipse plug-in enables you to create Java-based test scripts directly in your Eclipse environment.
- Silk Test Classic – Silk Test Classic is the traditional, 4Test Silk Test product.
- Silk Test Agents – The Silk Test Agent is the software process that translates the commands in your tests into GUI-specific commands. In other words, the Agent drives and monitors the application you are testing. One Agent can run locally on the host machine. In a networked environment, any number of Agents can run on remote machines.

The product suite that you install determines which components are available. To install all components, choose the complete install option. To install all components with the exception of Silk Test Classic, choose the standard install option.

Product Notification Service

The product notification service is an application that runs in your system tray and allows you to find out if updates are available for Silk Test. It also provides a link for you to click to navigate to the updates.

Running the Service

In the system tray, click the update notification icon and the Product Notification Service application opens.

Installed Version	Provides the version number of the currently installed Silk Test application.
Update Version	Provides a link and the version number of the next minor update, if one is available.
New Version	Provides a link and the version number of the next full release, if one is available.
Settings	Click the **Settings** button to open the **Settings** window. Select if and how often you want the notification service to check for updates.

Silk4J Quick Start Tutorial

This tutorial provides a step-by-step introduction to using Silk4J to test a Web application using dynamic object recognition. Dynamic object recognition enables you to write test cases that use XPath queries to find and identify objects.

Important: To successfully complete this tutorial you need basic knowledge of Java and JUnit.

For the sake of simplicity, this guide assumes that you have installed Silk4J and are using the sample Insurance Company Web application, available from *http://demo.borland.com/InsuranceWebExtJS/*.

Note: You must have local administrator privileges to run Silk4J.

Creating a Silk4J Project

When you create a Silk4J project using the **New Silk4J Project** wizard, the wizard contains the same options that are available when you create a Java project using the **New Java Project** wizard. Additionally, the Silk4J wizard automatically makes the Java project a Silk4J project.

1. In the Eclipse workspace, perform one of the following steps:
   - Click the drop-down arrow next to the Silk Test toolbar icon and choose **New Silk4J Project**.
   - If you installed or updated Silk4J to an existing Eclipse location, choose **File** > **New** > **Other** . Expand the Silk4J folder and double-click **Silk4J Project**.

   The **New Silk4J Project** wizard opens.
2. In the **Project name** text box, type a name for your project.
   For example, type *Tutorial*.
3. Accept the default settings for the remaining options.
4. Click **Next** and specify any other settings that you require.
5. Click **Finish**. A new Silk4J project is created that includes the JRE system library and the required `.jar` files, `silktest-jtf-nodeps.jar` and the `junit.jar`.

Recording a Test for the Insurance Company Web Application

Record a new test that navigates to the **Agent Lookup** page in the Insurance Company Web application. For a detailed version of how to record a test and how to configure test applications for each technology type, see the *Creating Tests* section of the Silk4J User Guide.

1. In the **Package Explorer**, perform one of the following steps:
   - Right-click the name of your project and choose **New** > **Other** . Expand the Silk4J folder and double-click **Silk4J Test**.
   - Click the drop-down arrow next to the Silk Test toolbar icon and choose **Record Silk4J Test**.
   - If you installed or updated Silk4J to an existing Eclipse location, choose **File** > **New** > **Other** . Expand the Silk4J folder and double-click **Silk4J Test**.

   The **New Silk4J Test** dialog box opens.

2. In the **Source folder** text box, specify the source file location that you want to use.

   The **Source folder** text box is automatically populated with the source file location for the project that you selected.

   To use a different source folder, click **Browse** and navigate to the folder that you want to use.
3. In the **Package** text box, specify the package name.
   For example, type: `com.example`.

   To use an existing package, click **Browse** and select the package that you want to use.
4. In the **Test class** text box, specify the name for the test class.
   For example, type: `AutoQuoteInput`.

   To use an existing class, click **Browse** and select the class that you want to use.
5. In the **Test method** text box, specify a name for the test method.
   For example, type `autoQuote`.
6. Click **Finish**. The **New Application Configuration** wizard opens.
7. Double-click **Web Site Test Configuration**. The **New Web Site Configuration** page opens.
8. From the **Browser Type** group, select **Internet Explorer**.

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

   - **Use existing browser** – Click this option button to use a browser that is already open when you configure the test. For example, if the Web page that you want to test is already displayed in a browser, you might want to use this option.
   - **Start new browser** – Click this option button to start a new browser instance when you configure the test. Then, in the **Browse to URL** text box specify the Web page to open.

     For this tutorial, close all open browsers and then click **Start new browser** and specify *http://demo.borland.com/InsuranceWebExtJS/*.
10. Click **Finish**. The Web site opens. Silk4J creates a base state and starts recording.
11. In the Insurance Company Web site, perform the following steps:
    a) From the **Select a Service or login** list box, select **Auto Quote**. The **Automobile Instant Quote** page opens.
    b) Type a zip code and email address in the appropriate text boxes, click an automobile type, and then click **Next**.
    c) Specify an age, click a gender and driving record type, and then click **Next**.
    d) Specify a year, make, and model, click the financial info type, and then click **Next**. A summary of the information you specified appears.
    e) Point to the **Zip Code** that you specified and press `Ctrl+Alt` to add a verification to the script.

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

       The **Verify Properties** dialog box opens.
    f) Check the **textContents** check box and then click **OK**. A verification action is added to the script for the zip code text.

    An action that corresponds with each step is recorded.
12. Click **Stop Recording**. A base state, test class, test method and package are created. The new class file opens.

Replay the test to ensure that it works as expected. You can modify the test to make changes if necessary.

Replaying the Test for the Insurance Company Web Application

Right-click the **AutoQuoteInput** class in the Package Explorer and choose **Run As** > **Silk4J Test** .

When the test execution is complete, the **Playback Complete** dialog box opens. Click **Explore Results** to review the TrueLog for the completed test. In this example, the verification will fail, because the **Zip Code** field in the test application is not cleaned.

Replaying Tests

Run tests from within Eclipse or using the command line.

Replaying Test Methods from Eclipse

1. Navigate to the test method that you want to test.
2. *Optional:* Click on a browser icon in the toolbar to set the browser on which you want to replay the test.
3. Perform one of the following steps:
   - Right-click a package name in the Package Explorer to replay all test methods in the project.
   - Right-click a class name in the Package Explorer to replay all test methods in the class. Or, alternatively, open the class in the source editor and right-click in the source editor.
   - Right-click a method name in the Package Explorer to replay a test for only that method. Or, alternatively, open the class in the source editor and select a test method by clicking its name.
4. Choose **Run As** > **Silk4J 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.
5. When the test execution is complete, the **Playback Complete** dialog box opens. Click **Explore Results** to review the TrueLog for the completed test.

Replaying a Test Method from the Command Line

You must update the PATH variable to reference your JDK location before performing this task. For details, reference the Sun documentation at: *http://java.sun.com/j2se/1.5.0/install-windows.html*.

1. Set the CLASSPATH to:

   ```
   set CLASSPATH=<eclipse_install_directory>\plugins
   \org.junit4_4.3.1\junit.jar;
   %OPEN_AGENT_HOME%\JTF\silktest-jtf-nodeps.jar;C:\myTests.jar
   ```

2. Run the JUnit test method by typing:

   ```
   java org.junit.runner.JUnitCore <test class name>
   ```

   Note: For troubleshooting information, reference the JUnit documentation at: *http://junit.sourceforge.net/doc/faq/faq.htm#running_1*.

3. To run several test classes with Silk4J and to create a TrueLog, use the `SilkTestSuite` class to run the Silk4J tests.
   For example, to run the two classes *MyTestClass1* and *MyTestClass2* with TrueLog enabled, type the following code into your script:

   ```
   package demo;
   import org.junit.runner.RunWith;
   import org.junit.runners.Suite.SuiteClasses;
   import com.borland.silktest.jtf.SilkTestSuite;

   @RunWith(SilkTestSuite.class)
   @SuiteClasses({ MyTestClass1.class, MyTestClass2.class })
   public class MyTestSuite {}
   ```

   To run these test classes from the command line, type the following:

   ```
   java org.junit.runner.JUnitCore demo.MyTestSuite
   ```

Troubleshooting when Replaying Test Methods from Ant

When using Apache Ant to run Silk4J tests, using the JUnit task with `fork="yes"` causes tests to hang. This is a known issue of Apache Ant (*https://issues.apache.org/bugzilla/show_bug.cgi?id=27614*). Two workarounds exist. Choose one of the following:

- Do not use `fork="yes"`.
- To use `fork="yes"`, ensure that the Open Agent is launched before the tests are executed. This can be done either manually or with the following Ant target:

```
<property environment="env" />
<target name="launchOpenAgent">
<echo message="OpenAgent launch as spawned process" />
<exec spawn="true" executable="${env.OPEN_AGENT_HOME}/agent/
openAgent.exe" />
<!-- give the agent time to start -->
<sleep seconds="30" />
</target>
```

Visual Execution Logs with TrueLog

TrueLog is a powerful technology that simplifies root cause analysis of test case failures through visual verification. The results of test runs can be examined in TrueLog Explorer. When an error occurs during a test run, TrueLog enables you to easily locate the line in your script that generated the error so that the issue can be resolved.

Note: TrueLog is supported only for one local or remote agent in a script. When you use multiple agents, for example when testing an application on one machine, and that application writes data to a database on another machine, a TrueLog is written only for the first agent that was used in the script. When you are using a remote agent, the TrueLog file is also written on the remote machine.

For additional information about TrueLog Explorer, refer to the *Silk TrueLog Explorer User Guide*, located in **Start** > **Programs** > **Silk** > **Silk Test** > **Documentation**.

You can enable TrueLog in Silk4J to create visual execution logs during the execution of Silk4J tests. The TrueLog file is created in the working directory of the process that executed the Silk4J tests.

Note: To create a TrueLog during the execution of a Silk4J test, JUnit version 4.6 or later must be used. If the JUnit version is lower than 4.6 and you try to create a TrueLog, Silk4J writes an error message to the console, stating that the TrueLog could not be written.

The default setting is that screenshots are only created when an error occurs in the script, and only test cases with errors are logged.

Enabling TrueLog

To enable TrueLog:

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **TrueLog** tab.
3. In the **Logging** area, check the **Enable TrueLog** check box.
   - Click **All testcases** to log activity for all test cases, both successful and failed. This is the default setting.
   - Click **Testcases with errors** to log activity for only those test cases with errors.

The TrueLog file is created in the working directory of the process that executed the Silk4J tests. When the Silk4J test execution is complete, the **Playback Complete** dialog box opens, and you can choose to review the TrueLog for the completed test.

Why is TrueLog Not Displaying Non-ASCII Characters Correctly?

TrueLog Explorer is a MBCS-based application, meaning that to be displayed correctly, every string must be encoded in MBCS format. When TrueLog Explorer visualizes and customizes data, many string conversion operations may be involved before the data is displayed.

Sometimes when testing UTF-8 encoded Web sites, data containing characters cannot be converted to the active Windows system code page. In such cases, TrueLog Explorer will replace the non-convertible characters, which are the non-ASCII characters, with a configurable replacement character, which usually is '?'.

To enable TrueLog Explorer to accurately display non-ASCII characters, set the system code page to the appropriate language, for example Japanese.

Setting Script Options

Specify script options for recording, browser and custom attributes, classes to ignore, synchronization, and the replay mode.

Setting TrueLog Options

Enable TrueLogs to capture bitmaps and to log information for Silk4J.

Logging bitmaps and controls in TrueLogs may adversely affect the performance of Silk4J. Because capturing bitmaps and logging information can result in large TrueLog files, you may want to log test cases with errors only and then adjust the TrueLog options for test cases where more information is needed.

The results of test runs can be examined in TrueLog Explorer. For additional information about TrueLog Explorer, refer to the *Silk TrueLog Explorer User Guide*, located in **Start** > **Programs** > **Silk** > **Silk Test** > **Documentation**.

To enable TrueLog and customize the information that the TrueLog collects for Silk4J, perform the following steps:

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **TrueLog** tab.
3. In the **Logging** area, check the **Enable TrueLog** check box.
   - Click **All testcases** to log activity for all test cases, both successful and failed. This is the default setting.
   - Click **Testcases with errors** to log activity for only those test cases with errors.
4. In the **TrueLog file** field, type the path to and name of the TrueLog file, or click **Browse** and select the file.

   This path is relative to the machine on which the agent is running. The default path is the path of the Silk4J project folder, and the default name is the name of the suite class, with a `.xlg` suffix.

   Note: If you provide a local or remote path in this field, the path cannot be validated until script execution time.
5. Select the **Screenshot mode**.

   Default is **None**.
6. *Optional:* Set the **Delay**.

   This delay gives Windows time to draw the application window before a bitmap is taken. You can try to add a delay if your application is not drawn properly in the captured bitmaps.
7. Click **OK**.

Setting Recording Preferences

Set the shortcut key combination to pause recording and specify whether absolute values and mouse move actions are recorded.

Note: All the following settings are optional. Change these settings if they will improve the quality of your test methods.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Recording** tab.
3. To set `Ctrl+Shift` as the shortcut key combination to use to pause recording, check the **OPT_ALTERNATE_RECORD_BREAK** check box.

   By default, `Ctrl+Alt` is the shortcut key combination.

   Note: For SAP applications, you must set `Ctrl+Shift` as the shortcut key combination.
4. To record absolute values for scroll events, check the **OPT_RECORD_SCROLLBAR_ABSOLUT** check box.
5. To record mouse move actions for Web application, Win32 applications, and Windows forms applications, check the **OPT_RECORD_MOUSEMOVES** check box. You cannot record mouse move actions for child technology domains of the xBrowser technology domain, for example Adobe Flex and Swing.
6. If you record mouse move actions, in the **OPT_RECORD_MOUSEMOVE_DELAY** text box, specify how many milliseconds the mouse has to be motionless before a `MouseMove` is recorded

   By default this value is set to 200.
7. To record text clicks instead of `Click` actions on objects where `TextClick` actions usually are preferable to `Click` actions, check the **OPT_RECORD_TEXT_CLICK** check box.
8. To record image clicks instead of `Click` actions on objects where `ImageClick` actions usually are preferable to `Click` actions, check the **OPT_RECORD_IMAGE_CLICK** check box.
9. To record object maps, check the **OPT_RECORD_OBJECTMAPS** check box.
10. Click **OK**.

Setting Browser Recording Options

Specify browser attributes to ignore while recording and whether to record native user input instead of DOM functions.

Note: All the following settings are optional. Change these settings if they will improve the quality of your test methods.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Browser** tab.
3. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording. For example, if you do not want to record attributes named `height`, add the `height` attribute name to the grid.

   Separate attribute names with a comma.
4. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording. For example, if you do not want to record attributes assigned the value of `x-auto`, add `x-auto` to the grid.

   Separate attribute values with a comma.
5. To record native user input instead of DOM functions, check the **OPT_XBROWSER_RECORD_LOWLEVEL** check box.

   For example, to record `Click` instead of `DomClick` and `TypeKeys` instead of `SetText`, check this check box.

   If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.

6. To set the maximum length for locator attribute values, type the length into the field in the **Maximum attribute value length** section.

   If the actual length exceeds that limit the value is truncated and a wild card (*) is appended. By default this value is set to 20 characters.
7. Click **OK**.

Setting Custom Attributes

Silk4J includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results. You can use any property that is available in the respective technology as a custom attribute given that they are either numbers (integers, doubles), strings, item identifiers, or enumeration values.

A well-defined locator relies on attributes that change infrequently and therefore requires less maintenance. Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index might change when another object is added.

For the technology domains listed in the list box on the **Custom Attributes** tab, you can also retrieve arbitrary properties (such as a WPFButton that defines *myCustomProperty*) and then use those properties as custom attributes. To achieve optimal results, add a custom automation ID to the elements that you want to interact with in your test. In Web applications, you can add an attribute to the element that you want to interact with, such as `<div myAutomationId= "my unique element name" />`. Or, in Java SWT, the developer implementing the GUI can define an attribute (for example `testAutomationId`) for a widget that uniquely identifies the widget in the application. A tester can then add that attribute to the list of custom attributes (in this case, `testAutomationId`), and can identify controls by that unique ID. This approach can eliminate the maintenance associated with locator changes.

If multiple objects share the same attribute value, such as a caption, Silk4J tries to make the locator unique by combining multiple available attributes with the "and" operation and thus further narrowing down the list of matching objects to a single object. Should that fail, an index is appended. Meaning the locator looks for the *nth* control with the caption *xyz*.

If more than one object is assigned the same custom attribute value, all the objects with that value will return when you call the custom attribute. For example, if you assign the unique ID, `loginName` to two different text fields, both fields will return when you call the `loginName` attribute.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Custom Attributes** tab.
3. From the **Select a tech domain** list box, select the technology domain for the application that you are testing.

   Note: You cannot set custom attributes for Flex or Windows API-based client/server (Win32) applications.
4. Add the attributes that you want to use to the list.

   If custom attributes are available, the locator generator uses these attributes before any other attribute. The order of the list also represents the priority in which the attributes are used by the locator generator. If the attributes that you specify are not available for the objects that you select, Silk4J uses the default attributes for the application that you are testing.

   Separate attribute names with a comma.

Note: To include custom attributes in a Web application, add them to the html tag. For example type, `<input type='button' bcauid='abc' value='click me' />` to add an attribute called `bcauid`.

Note: To include custom attributes in a Java SWT control, use the `org.swt.widgets.Widget.setData(String key, Object value)` method.

Note: To include custom attributes in a Swing control, use the `SetClientProperty("propertyName", "propertyValue")` method.

5. Click **OK**.

Setting Classes to Ignore

Specify the names of any classes that you want to ignore during recording and playback.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Transparent Classes** tab.
3. In the **Transparent classes** grid, type the name of the class that you want to ignore during recording and playback.

   Separate class names with a comma.
4. Click **OK**.

Setting WPF Classes to Expose During Recording and Playback

Specify the names of any WPF classes that you want to expose during recording and playback. For example, if a custom class called *MyGrid* derives from the WPF `Grid` class, the objects of the *MyGrid* custom class are not available for recording and playback. `Grid` objects are not available for recording and playback because the `Grid` class is not relevant for functional testing since it exists only for layout purposes. As a result, `Grid` objects are not exposed by default. In order to use custom classes that are based on classes that are not relevant to functional testing, add the custom class, in this case *MyGrid*, to the **OPT_WPF_CUSTOM_CLASSES** option. Then you can record, playback, find, verify properties, and perform any other supported actions for the specified classes.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **WPF** tab.
3. In the **Custom WPF class names** grid, type the name of the class that you want to expose during recording and playback.

   Separate class names with a comma.
4. Click **OK**.

Setting Synchronization Options

Specify the synchronization and timeout values for Web applications.

Note: All the following settings are optional. Change these settings if they will improve the quality of your test methods.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Synchronization** tab.
3. To specify the synchronization algorithm for the ready state of a web application, from the **OPT_XBROWSER_SYNC_MODE** list box, choose an option.

   The synchronization algorithm configures the waiting period for the ready state of an invoke call. By default, this value is set to **AJAX**.
4. In the **Synchronization exclude list** text box, type the entire URL or a fragment of the URL for any service or Web page that you want to exclude.

   Some AJAX frameworks or browser applications use special HTTP requests, which are permanently open in order to retrieve asynchronous data from the server. These requests may let the synchronization hang until the specified synchronization timeout expires. To prevent this situation, either use the HTML synchronization mode or specify the URL of the problematic request in the **Synchronization exclude list** setting.

   For example, if your Web application uses a widget that displays the server time by polling data from the client, permanent traffic is sent to the server for this widget. To exclude this service from the synchronization, determine what the service URL is and enter it in the exclusion list.

   For example, you might type:

   - http://example.com/syncsample/timeService
   - timeService
   - UICallBackServiceHandler

   Separate multiple entries with a comma.

   Note: If your application uses only one service, and you want to disable that service for testing, you must use the HTML synchronization mode rather than adding the service URL to the exclusion list.
5. To specify the maximum time, in milliseconds, to wait for an object to be ready, type a value in the **OPT_SYNC_TIMEOUT** text box.

   By default, this value is set to **300000**.
6. To specify the time, in milliseconds, to wait for an object to be resolved during replay, type a value in the **OPT_WAIT_RESOLVE_OBJDEF** text box.

   By default, this value is set to **5000**.
7. To specify the time, in milliseconds, to wait before the agent attempts to resolve an object again, type a value in the **OPT_WAIT_RESOLVE_OBJDEF_RETRY** text box.

   By default, this value is set to **500**.
8. Click **OK**.

Setting Replay Options

Specify whether you want to ensure that the object that you want to test is active and whether to override the default replay mode. The replay mode defines whether controls are replayed with the mouse and keyboard or with the API. Use the default mode to deliver the most reliable results. When you select another mode, all controls use the selected mode.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Replay** tab. The **Replay Options** page displays.
3. From the **OPT_REPLAY_MODE** list box, select one of the following options:

- **Default** – Use this mode for the most reliable results. By default, each control uses either the mouse and keyboard (low level) or API (high level) modes. With the default mode, each control uses the best method for the control type.
- **High level** – Use this mode to replay each control using the API.
- **Low level** – Use this mode to replay each control using the mouse and keyboard.

4. To ensure that the object that you want to test is active, check the **OPT_ENSURE_ACTIVE_OBJDEF** check box.
5. To change the time to wait for an object to become enabled during playback, type the new time into the field in the **Object enabled timeout** section.

   The time is specified in milliseconds. The default value is 1000.
6. To edit the prefix that specifies that an asset is located in the current project, edit the text for the **Asset namespace** option in the **OPT_ASSET_NAMESPACE** text box.
7. Click **OK**.

Setting Advanced Options

Specify whether you want to enable Windows Accessibility, whether the focus should be removed from the window during text capture, and whether locator attribute names should be case sensitive.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Advanced** tab. The **Advanced Options** page displays.
3. Check the **OPT_ENABLE_ACCESSIBILITY** check box to enable Microsoft Accessibility in addition to the normal Win32 control recognition.
4. Check the **OPT_REMOVE_FOCUS_ON_CAPTURE_TEXT** check box to remove the focus from the window before capturing a text.

   A text capture is performed during recording and replay by the following methods:

   - `TextClick`
   - `TextCapture`
   - `TextExists`
   - `TextRect`
5. Check the **OPT_LOCATOR_ATTRIBUTES_CASE_SENSITIVE** check box to set locator attribute names to be case sensitive.
6. Set the default accuracy level for new image assets by selecting a value from 1 (low accuracy) to 10 (high accuracy) from the **OPT_IMAGE_ASSET_DEFAULT_ACCURACY** list box.
7. Set the default accuracy level for new image verification assets by selecting a value from 1 (low accuracy) to 10 (high accuracy) from the **OPT_IMAGE_VERIFICATION_DEFAULT_ACCURACY** list box.
8. Click **OK**.

Setting Silk4J Preferences

Silk4J requires Java Runtime Environment (JRE) version 1.6 or higher.

By default Silk4J checks the JRE version each time you start Silk4J, and displays an error message if the JRE version is incompatible with Silk4J.

1. To turn off the error message, choose **Window** > **Preferences** > **Silk4J** .
2. Select the **Silk4J** branch and uncheck the **Show error message if the JRE version is incompatible** check box.
3. Click **OK**.

Testing Specific Environments

Silk4J supports testing several types of environments.

Active X/Visual Basic Applications

Silk4J provides support for testing ActiveX/Visual Basic applications.

Check the Release Notes for the most up-to-date information about supported versions, any known issues, and workarounds.

Dynamically Invoking ActiveX/Visual Basic Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

  Silk4J types includes primitive types (such as boolean, int, string), lists, and other types (such as Point)

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.

Adobe Flex Support

Silk4J provides built-in support for testing Adobe Flex applications using Internet Explorer, Mozilla Firefox, and the Standalone Flash Player, and Adobe AIR applications built with Adobe Flex 4 or later.

Silk4J also supports multiple application domains in Adobe Flex 3.x and 4.x applications, which enables you to test sub-applications. Silk4J 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.

Supported Controls

For a complete list of the record and playback controls available for Adobe Flex testing, view a list of the supported Flex classes in the `com.borland.silktest.jtf.flex` package in the *API Reference*.

Note: The Silk Test Flex Automation SDK is based on the Automation API for Adobe Flex. The Silk Test Automation SDK supports the same components in the same manner that the Automation API for Adobe Flex supports them. For instance, the `typekey` statement in the Flex Automation API does not support all keys. You can use the input text statement to resolve this issue. For more information about using the Flex Automation API, see the *Adobe Flex Release Notes*.

Configuring Flex Applications to Run in Adobe Flash Player

To run an Adobe Flex application in Flash Player, one or both of the following must be true:

- The developer who creates the Flex application must compile the application as an EXE file. When a user launches the application, it will open in Flash Player. Install Windows Flash Player from *http://www.adobe.com/support/flashplayer/downloads.html*.
- The user must have Windows Flash Player Projector installed. When a user opens a Flex .SWF file, he can configure it to open in Flash Player. Windows Flash Projector is not installed when Flash Player is installed unless you install the Adobe Flex developer suite. Install Windows Flash Projector from *http://www.adobe.com/support/flashplayer/downloads.html*.

1. For Windows 7 and Windows 2008 R2, configure Flash Player to run as administrator. Perform the following steps:
   a) Right-click the Adobe Flash Player program shortcut or the FlashPlayer.exe file, then click **Properties**.
   b) In the **Properties** dialog box, click the **Compatibility** tab.
   c) Check the **Run this program as an administrator** check box and then click **OK**.
2. Start the .SWF file in Flash Player from the command prompt (cmd.exe) by typing:

   `"<Application_Install_Directory>\ApplicationName.swf"`

   By default, the <*SilkTest_Install_Directory*> is located at `Program Files\Silk\Silk Test`.

Launching the Component Explorer

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

In Internet Explorer, open *http://demo.borland.com/flex/SilkTest14.0/index.html*. The application launches in your default browser.

Testing Adobe Flex Applications

Silk Test provides built-in support for testing Adobe Flex applications. Silk Test also provides several sample Adobe Flex applications. You can access the sample applications at *http://demo.borland.com/flex/SilkTest14.0/index.html*.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Before you can test your own Adobe Flex application, your Adobe Flex developers must perform the following steps:

- Enabling your Adobe Flex application for testing
- Creating testable Adobe Flex applications
- Coding Adobe Flex containers
- Implementing automation support for custom controls

To test your own Adobe Flex application, follow these steps:

- Configuring security settings for your local Flash Player
- Recording a test
- Playing back a test
- Customizing Adobe Flex scripts
- Testing a custom Adobe Flex control

Note: Loading an Adobe Flex application and initializing the Flex automation framework may take some time depending on the machine on which you are testing and the complexity of your Adobe Flex application. Set the Window timeout value to a higher value to enable your application to fully load.

Testing Adobe Flex Custom Controls

Silk4J supports testing Flex custom controls. By default, Silk4J provides record and playback support for the individual subcontrols of the custom control.

For testing custom controls, the following options exist:

- Basic support

  With basic support, you use dynamic invoke to interact with the custom control during replay. Use this low-effort approach when you want to access properties and methods of the custom control in the test application that Silk4J does not expose. The developer of the custom control can also add methods and properties to the custom control specifically for making the control easier to test. A user can then call those methods or properties using the dynamic invoke feature.

  The advantages of basic support include:

  - Dynamic invoke requires no code changes in the test application.
  - Using dynamic invoke is sufficient for most testing needs.

  The disadvantages of basic support include:

    - No specific class name is included in the locator (for example, Silk4J records “//FlexBox” rather than “//FlexSpinner”)
    - Only limited recording support
    - Silk4J cannot replay events.

    For more details about dynamic invoke, including an example, see *Dynamically Invoking Adobe Flex Methods*.
- Advanced support

  With advanced support, you create specific automation support for the custom control. This additional automation support provides recording support and more powerful play-back support. The advantages of advanced support include:

    - High-level recording and playback support, including the recording and replaying of events.
    - Silk4J treats the custom control exactly the same as any other built-in Flex control.
    - Seamless integration into Silk4J API
    - Silk4J uses the specific class name in the locator (for example, Silk4J records “//FlexSpinner”)

  The disadvantages of advanced support include:

    - Implementation effort is required. The test application must be modified and the Open Agent must be extended.

Dynamically Invoking Flex Methods

You can call methods, retrieve properties, and set properties on controls that Silk4J does not expose by using the dynamic invoke feature. This feature is useful for working with custom controls and for working with controls that Silk4J supports without customization.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

Note: Typically, most properties are read-only and cannot be set.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods that the Flex API defines
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

  Silk4J types includes primitive types (such as boolean, int, string), lists, and other types (such as Point)

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.

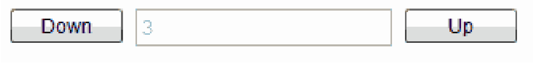
- All methods that have no return value return `null`.

Defining a Custom Control in the Test Application

Typically, the test application already contains custom controls, which were added during development of the application. If your test application already includes custom controls, you can proceed to *Testing a Flex Custom Control Using Dynamic Invoke* or to *Testing a Custom Control Using Automation Support*.

This procedure shows how a Flex application developer can create a spinner custom control in Flex. The spinner custom control that we create in this topic is used in several topics to illustrate the process of implementing and testing a custom control.

The spinner custom control includes two buttons and a textfield, as shown in the following graphic.



The user can click **Down** to decrement the value that is displayed in the textfield and click **Up** to increment the value in the textfield.

The custom control offers a public "CurrentValue" property that can be set and retrieved.

1. In the test application, define the layout of the control.
   For example, for the spinner control type:

```
<?xml version="1.0" encoding="utf-8"?>
<customcontrols:SpinnerClass xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:controls="mx.controls.*" xmlns:customcontrols="customcontrols.*">
    <controls:Button id="downButton" label="Down" />
    <controls:TextInput id="text" enabled="false" />
    <controls:Button id="upButton" label="Up"/>
</customcontrols:SpinnerClass>
```

2. Define the implementation of the custom control.
   For example, for the spinner control type:

```
package customcontrols
{
    import flash.events.MouseEvent;

    import mx.containers.HBox;
    import mx.controls.Button;
    import mx.controls.TextInput;
    import mx.core.UIComponent;
    import mx.events.FlexEvent;

    [Event(name="increment",     type="customcontrols.SpinnerEvent")]
    [Event(name="decrement",     type="customcontrols.SpinnerEvent")]

    public class SpinnerClass extends HBox
    {
        public var downButton : Button;
        public var upButton : Button;
        public var text : TextInput;
        public var ssss: SpinnerAutomationDelegate;
        private var _lowerBound : int = 0;
        private var _upperBound : int = 5;

        private var _value : int = 0;
        private var _stepSize : int = 1;


        public function SpinnerClass() {
            addEventListener(FlexEvent.CREATION_COMPLETE,
creationCompleteHandler);
        }

        private function creationCompleteHandler(event:FlexEvent) : void {
            downButton.addEventListener(MouseEvent.CLICK,
downButtonClickHandler);
            upButton.addEventListener(MouseEvent.CLICK,
upButtonClickHandler);
            updateText();
        }

        private function downButtonClickHandler(event : MouseEvent) : void {
            if(currentValue - stepSize >= lowerBound) {
                currentValue = currentValue - stepSize;
            }
            else {
                currentValue = upperBound - stepSize + currentValue -
lowerBound + 1;
            }

            var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.DECREMENT);
            spinnerEvent.steps = _stepSize;
            dispatchEvent(spinnerEvent);
        }

        private function upButtonClickHandler(event : MouseEvent) : void {
            if(currentValue <= upperBound - stepSize) {
                currentValue = currentValue + stepSize;
            }
            else {
                currentValue = lowerBound + currentValue + stepSize -
upperBound - 1;
            }

            var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.INCREMENT);
            spinnerEvent.steps = _stepSize;
            dispatchEvent(spinnerEvent);
        }

        private function updateText() : void {
            if(text != null) {
                text.text = _value.toString();
            }
        }

        public function get currentValue() : int {
            return _value;
        }

        public function set currentValue(v : int) : void {
            _value = v;
            if(v < lowerBound) {
                _value = lowerBound;
            }
            else if(v > upperBound) {
                _value = upperBound;
            }
            updateText();

        }

        public function get stepSize() : int {
            return _stepSize;
        }

        public function set stepSize(v : int) : void {
            _stepSize = v;
        }

        public function get lowerBound() : int {
            return _lowerBound;
        }

        public function set lowerBound(v : int) : void {
            _lowerBound = v;
            if(currentValue < lowerBound) {
                currentValue = lowerBound;
            }
        }

        public function get upperBound() : int {
            return _upperBound;
        }

        public function set upperBound(v : int) : void {
            _upperBound = v;
            if(currentValue > upperBound) {
                currentValue = upperBound;
            }
        }
    }
}
```

3. Define the events that the control uses.
   For example, for the spinner control type:

```
package customcontrols
{
    import flash.events.Event;

    public class SpinnerEvent extends Event
    {
        public static const INCREMENT : String = "increment";
        public static const DECREMENT : String = "decrement";

        private var _steps : int;

        public function SpinnerEvent(eventName : String) {
            super(eventName);
        }

        public function set steps(value:int) : void {
            _steps = value;
        }

        public function get steps() : int {
            return _steps;
        }

    }
}
```

The next step is to implement automation support for the test application.

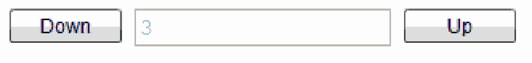
Testing a Flex Custom Control Using Dynamic Invoke

Silk4J provides record and playback support for custom controls using dynamic invoke to interact with the custom control during replay. Use this low-effort approach when you want to access properties and methods of the custom control in the test application that Silk4J does not expose. The developer of the custom control can also add methods and properties to the custom control specifically for making the control easier to test.

1. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.
2. Call dynamic methods on objects with the `invoke` method.
3. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.
4. Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method.

Example

The following example tests a spinner custom control that includes two buttons and a textfield, as shown in the following graphic.



The user can click **Down** to decrement the value that is displayed in the textfield and click **Up** to increment the value in the textfield.

The custom control offers a public "CurrentValue" property that can be set and retrieved.

To set the spinner's value to 4, type the following:

```
FlexBox spinner = _desktop.<FlexBox>find("//
FlexBox[@className=customcontrols.Spinner]");
spinner.setProperty("CurrentValue", 4);
```

Testing a Custom Control Using Automation Support

You can create specific automation support for the custom control. This additional automation support provides recording support and more powerful play-back support. To create automation support, the test application must be modified and the Open Agent must be extended.

Before you can test a custom control in Silk4J, perform the following steps:

- Define the custom control in the test application
- Implement automation support

After the test application has been modified and includes automation support, perform the following steps:

1. Create a Java class for the custom control in order to test the custom control in your tests.

   For example, the spinner control class must have the following content:

```
package customcontrols;

import com.borland.silktest.jtf.Desktop;
import com.borland.silktest.jtf.common.JtfObjectHandle;
import com.borland.silktest.jtf.flex.FlexBox;

/**
 * Implementation of the FlexSpinner Custom Control.
 */
public class FlexSpinner extends FlexBox {

  protected FlexSpinner(JtfObjectHandle handle, Desktop desktop) {
    super(handle, desktop);
  }

  @Override
  protected String getCustomTypeName() {
    return "FlexSpinner";
  }

  public Integer getLowerBound() {
    return (Integer) getProperty("lowerBound");
  }

  public Integer getUpperBound() {
    return (Integer) getProperty("upperBound");
  }

  public Integer getCurrentValue() {
    return (Integer) getProperty("currentValue");
  }

  public void setCurrentValue(Integer currentValue) {
    setProperty("currentValue", currentValue);
  }

  public Integer getStepSize() {
    return (Integer) getProperty("stepSize");
  }

  public void increment(Integer steps) {
    invoke("Increment", steps);
  }

  public void decrement(Integer steps) {
    invoke("Decrement", steps);
  }

}
```

2. Add this Java class to the Silk4J test project that contains your tests.

   Tip: To use the same custom control in multiple Silk4J projects, we recommend that you create a separate project that contains the custom control and reference it from your Silk4J test projects.

3. Add the following line to the `<Silk Test installation directory>\ng\agent\plugins\com.borland.silktest.jtf.agent.customcontrols_<version>\config\classMapping.properties` file:

   ```
   FlexSpinner=customcontrols.FlexSpinner
   ```

   The code to the left of the equals sign must be the name of custom control as defined in the XML file. The code to the right of the equals sign must be the fully qualified name of the Java class for the custom control.

   Now you have full record and playback support when using the custom control in Silk4J.

Examples

The following example shows how increment the spinner's value by 3 by using the "Increment" method that has been implemented in the automation delegate:

```
desktop.<FlexSpinner>find("//FlexSpinner[@caption='index:
1']").increment(3);
```

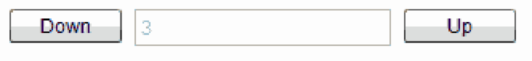
This example shows how to set the value of the spinner to 3.

```
desktop.<FlexSpinner>find("//FlexSpinner[@caption='index:
1']").setCurrentValue(3);
```

Implementing Automation Support for a Custom Control

Before you can test a custom control, implement automation support (the automation delegate) in ActionScript for the custom control and compile that into the test application.

The following procedure uses a custom Flex spinner control to demonstrate how to implement automation support for a custom control. The spinner custom control includes two buttons and a textfield, as shown in the following graphic.



The user can click **Down** to decrement the value that is displayed in the textfield and click **Up** to increment the value in the textfield.

The custom control offers a public "CurrentValue" property that can be set and retrieved.

1. Implement automation support (the automation delegate) in ActionScript for the custom control.

   For further information about implementing an automation delegate, see the Adobe Live Documentation at *http://livedocs.adobe.com/flex/3/html/help.html?content=functest_components2_14.html*.

   In this example, the automation delegate adds support for the methods "increment", "decrement". The example code for the automation delegate looks like this:

```
package customcontrols
{
    import flash.display.DisplayObject;
    import mx.automation.Automation;
    import customcontrols.SpinnerEvent;
    import mx.automation.delegates.containers.BoxAutomationImpl;
    import flash.events.Event;
    import mx.automation.IAutomationObjectHelper;
    import mx.events.FlexEvent;
    import flash.events.IEventDispatcher;
    import mx.preloaders.DownloadProgressBar;
    import flash.events.MouseEvent;
    import mx.core.EventPriority;

    [Mixin]
    public class SpinnerAutomationDelegate extends BoxAutomationImpl
    {

        public static function init(root:DisplayObject) : void {
            // register delegate for the automation
            Automation.registerDelegateClass(Spinner,
SpinnerAutomationDelegate);
        }

        public function SpinnerAutomationDelegate(obj:Spinner) {
            super(obj);
            // listen to the events of interest (for recording)
            obj.addEventListener(SpinnerEvent.DECREMENT, decrementHandler);
            obj.addEventListener(SpinnerEvent.INCREMENT, incrementHandler);
        }

        protected function decrementHandler(event : SpinnerEvent) : void {
            recordAutomatableEvent(event);
        }
        protected function incrementHandler(event : SpinnerEvent) : void {
            recordAutomatableEvent(event);
        }

        protected function get spinner() : Spinner {
            return uiComponent as Spinner;
        }

        //------------------------------------
        //  override functions
        //------------------------------------

        override public function get automationValue():Array {
            return [ spinner.currentValue.toString() ];
        }

        private function replayClicks(button : IEventDispatcher, steps :
int) : Boolean {
            var helper : IAutomationObjectHelper =
Automation.automationObjectHelper;
            var result : Boolean;
            for(var i:int; i < steps; i++) {
                helper.replayClick(button);
            }
            return result;
        }

        override public function
replayAutomatableEvent(event:Event):Boolean {

            if(event is SpinnerEvent) {
                var spinnerEvent : SpinnerEvent = event as SpinnerEvent;
                if(event.type == SpinnerEvent.INCREMENT) {
                    return replayClicks(spinner.upButton,
spinnerEvent.steps);
                }
                else if(event.type == SpinnerEvent.DECREMENT) {
                    return replayClicks(spinner.downButton,
spinnerEvent.steps);
                }
                else {
                    return false;
                }

            }
            else {
                return super.replayAutomatableEvent(event);
            }
        }

        // do not expose the child controls (i.e the buttons and the
textfield) as individual controls
        override public function get numAutomationChildren():int {
            return 0;
        }

    }
}
```

2. To introduce the automation delegate to the Open Agent, create an XML file that describes the custom control.

The class definition file contains information about all instrumented Flex components. This file (or files) provides information about the components that can send events during recording and accept events for replay. The class definition file also includes the definitions for the supported properties.

The XML file for the spinner custom control looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<TypeInformation>
    <ClassInfo Name="FlexSpinner" Extends="FlexBox">
        <Implementation
            Class="customcontrols.Spinner" />
        <Events>
            <Event Name="Decrement">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="decrement" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
                </Property>
            </Event>
            <Event Name="Increment">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="increment" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
                </Property>
            </Event>
        </Events>
        <Properties>
            <Property Name="lowerBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="upperBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <!-- expose read and write access for the currentValue property
-->
            <Property Name="currentValue" accessType="both">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="stepSize" accessType="read">
                <PropertyType Type="integer" />
            </Property>
        </Properties>
    </ClassInfo>
</TypeInformation>
```

3. Include the XML file for the custom control in the folder that includes all the XML files that describe all classes and their methods and properties for the supported Flex controls.

   Silk Test contains several XML files that describe all classes and their methods and properties for the supported Flex controls. Those XML files are located in the <`<Silk Test_install_directory>\ng\agent\plugins\com.borland.fastxd.techdomain.flex.agent_<version>\config\automationEnvironment` folder.

   If you provide your own XML file, you must copy your XML file into this folder. When the Open Agent starts and initializes support for Adobe Flex, it reads the contents of this directory.

   To test the Flex Spinner sample control, you must copy the CustomControls.xml file into this folder. If the Open Agent is currently running, restart it after you copy the file into the folder.

Flex Class Definition File

The class definition file contains information about all instrumented Flex components. This file (or files) provides information about the components that can send events during recording and accept events for replay. The class definition file also includes the definitions for the supported properties.

Silk Test contains several XML files that describe all classes/events/properties for the common Flex common and specialized controls. Those XML files are located in the `<Silk Test_install_directory>\ng\agent\plugins\com.borland.fastxd.techdomain.flex.agent_<version>\config\automationEnvironment` folder.

If you provide your own XML file, you must copy your XML file into this folder. When the Silk Test Agent starts and initializes support for Adobe Flex, it reads the contents of this directory.

The XML file has the following basic structure:

```
<TypeInformation>

<ClassInfo>

<Implementation />

<Events>

<Event />

…

</Events>

<Properties>

<Property />

…

</Properties>

</ClassInfo>

</TypeInformation>
```

Customizing Adobe Flex Scripts

You can manually customize your Flex scripts. You can insert verifications manually 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, view a list of the supported Flex classes in the `com.borland.silktest.jtf.flex` package in the *API Reference*.

1. Record a test for your Flex application.
2. Open the script file that you want to customize.
3. Manually type the code that you want to add.

Testing Multiple Flex Applications on the Same Web Page

When multiple Flex applications exist on the same Web page, Silk4J uses the Flex application ID or the application size property to determine which application to test. If multiple applications exist on the same page, but they are different sizes, Silk4J uses the size property to determine on which application to perform any actions and no additional steps are necessary.

Silk4J uses JavaScript to find the Flex application ID to determine on which application to perform any actions if:

- Multiple Flex applications exist on a single Web page
- Those applications are the same size

Note: In this situation, if JavaScript is not enabled on the browser machine, an error occurs when a script runs.

1. Enable JavaScript.
2. In Internet Explorer, perform the following steps:
   a) Choose **Tools** > **Internet Options**.
   b) Click the **Security** tab.
   c) Click **Custom level**.
   d) In the **Scripting** section, under **Active Scripting**, click **Enable** and click **OK**.
3. Follow the steps in *Testing Adobe Flex Applications*.

   Note: If a frame exists on the Web page and the applications are the same size, this method will not work.

Adobe AIR Support

Silk4J supports testing with Adobe AIR for applications that are compiled with the Flex 4 compiler. For details about supported versions, check the *Release Notes* for the latest information.

Silk Test provides a sample Adobe AIR application. You can access the sample application at *http://demo.borland.com/flex/SilkTest14.0/index.html* and then click the Adobe AIR application that you want to use. You can select the application with or without automation. In order to execute the AIR application, you must install the Adobe AIR Runtime.

Overview of the Flex Select Method Using Name or Index

You can record Flex `Select` methods using the `Name` or `Index` of the control that you select. By default, Silk4J records `Select` methods using the name of the control. However, you can change your environment to record `Select` events using the index for the control, or you can switch between the name and index for recording.

You can record `Select` events using the index for the following controls:

- `FlexList`
- `FlexTree`
- `FlexDataGrid`
- `FlexAdvancedDataGrid`
- `FlexOLAPDataGrid`
- `FlexComboBox`

The default setting is ItemBasedSelection (Select event), which uses the name control. To use the index, you must adapt the AutomationEnvironment to use the IndexBasedSelection (SelectIndex event). To change the behavior for one of these classes, you must modify the FlexCommonControls.xml, AdvancedDataGrid.xml, or OLAPDataGrid.xml file using the following code. Those XML files are located in the `<Silk Test_install_directory>\ng\agent\plugins`

`\com.borland.fastxd.techdomain.flex.agent_< version>\config \automationEnvironment` folder. Make the following adaptations in the corresponding xml file.

```
<ClassInfo Extends="FlexList" Name="FlexControlName"
EnableIndexBasedSelection=”true” >

…

</ClassInfo>
```

With this adaption the IndexBasedSelection is used for recording `FlexList::SelectIndex` events. Setting the `EnableIndexBasedSelection=` to `false` in the code or removing the Boolean returns recording to using the name (`FlexList::Select` events).

Note: You must re-start your application, which automatically re-starts the Silk Test Agent, in order for these changes to become active.

Selecting an Item in the FlexDataGrid Control

Select an item in the FlexDataGrid control using the index value or the content value.

1. To select an item in the FlexDataGrid control using the index value, use the `SelectIndex` method. For example, type `FlexDataGrid.SelectIndex(1)`.
2. To select an item in the FlexDataGrid control using the content value, use the `Select` method.

   Identify the row that you want to select with the required formatted string. Items must be separated by a pipe (" | "). At least one Item must be enclosed by two stars ("*"). This identifies the item where the click will be performed.

   The syntax is: `FlexDataGrid.Select("*Item1* | Item2 | Item3")`

Enabling Your Flex Application for Testing

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

- Adobe Flex Automation Package
- Silk Test Automation Package

Adobe Flex Automation Package

The Flex automation package provides developers with the ability to create Flex applications that use the Automation API. You can download the Flex automation package from Adobe's website, *http://www.adobe.com*. The package includes:

- Automation libraries – the automation.swc and automation_agent.swc libraries are the implementations of the delegates for the Flex framework components. The automation_agent.swc file and its associated resource bundle are the generic agent mechanism. An agent, such as the Silk Test Agent, builds on top of these libraries.
- Samples

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, the `typekey` statement in the Flex Automation API does not support all keys. You can use the input text statement to resolve this issue. For more information about using the Flex Automation API, see the *Adobe Flex Release Notes*.

Silk Test Automation Package

Silk Test's Open Agent uses the Adobe Flex automation agent libraries. The FlexTechDomain.swc file contains the Silk Test specific implementation.

You can enable your application for testing using either of the following methods:

- Linking automation packages to your Flex application
- Run-time loading

Linking Automation Packages to Your Flex Application

You must precompile Flex 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 if your application uses charts and Flex > 3.x)
- automation_flasflexkit.swc (include if your application uses embedded flash content)
- automation_spark.swc (include if your application uses the new Flex 4.x 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 new Flex 4.x controls)

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.

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 runs out of memory eventually. The Flex Control 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, see the *Adobe Flex Release Notes*.

Precompiling the Flex Application for Testing

You can enable your application for testing by precompiling your application for testing or by using run-time loading.

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.

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:

   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

   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 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 runs out of memory eventually. The Flex Control 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, see the *Adobe Flex Release Notes*.

Run-Time Loading

You can load Flex 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

- 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 Silk Test Flex Automation Launcher.
- 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);
```

Run-Time Loading

1. Copy the content of the `Silk\Silk Test\ng\AutomationSDK\Flex\<`*`version`*`>\FlexAutomationLauncher` directory into the directory of the 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 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/Silk Test/ng/sampleapplications/Flex/3.2/
   FlexControlExplorer32/FlexAutomationLauncher.html?automationurl=explorer.swf
   ```

Using the Command Line to Add Configuration Information

To specify the location of the automation.swc, automation_agent.swc, and FlexTechDomain.swc libraries using the command-line compiler, use the include-libraries compiler option.

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

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.

To add automated testing support to a Flex Builder project, you must also add the `automation.swc` and `automation_agent.swc` files to the include-libraries compiler option.

Passing Parameters into a Flex Application

You can pass parameters into a Flex application using the following procedures.

Passing Parameters into a Flex Application Before Runtime

You can pass parameters into a Flex application before runtime using automation libraries.

1. Compile your application with the appropriate automation libraries.
2. Use the standard Flex mechanism for the parameter as you typically would.

Passing Parameters into a Flex Application at Runtime Using the Flex Automation Launcher

Before you begin this task, prepare your application for run-time loading.

1. Open the `FlexAutomationLauncher.html` file or create a file using `FlexAutomationLauncher.html` as an example.
2. Navigate to the following section:

```
<script language="JavaScript" type="text/javascript">

                AC_FL_RunContent(eef

                   "src", "FlexAutomationLauncher",

                   "width", "100%",

                   "height", "100%",

                   "align", "middle",

                   "id", "FlexAutomationLauncher",

                   "quality", "high",

                   "bgcolor", "white",

                   "name", "FlexAutomationLauncher",

                   "allowScriptAccess","sameDomain",

                   "type", "application/x-shockwave-flash",

                   "pluginspage", "http://www.adobe.com/go/getflashplayer",

                   "flashvars", "yourParameter=yourParameterValue"+
"&automationurl=YourApplication.swf"



                );

        </script>
```

Note: Do not change the "`FlexAutomationLauncher`" value for "`src`", "`id`", or "`name`."

3. Add your own parameter to "*yourParameter=yourParameterValue*".
4. Pass the name of the Flex application that you want to test as value for the "*& automationurl=YourApplication.swf*" value.

5. Save the file.

Creating Testable Flex Applications

As a Flex developer, you can employ techniques to make Flex applications as "test friendly" as possible. These include:

- Providing Meaningful Identification of Objects
- Avoiding Duplication of Objects

Providing Meaningful Identification of Objects

To create "test friendly" applications, ensure that objects are identifiable in scripts. You can set the value of the ID property for all controls that are tested, and ensure that you use a meaningful string for that ID property.

To provide meaningful identification of objects:

- Give all testable MXML components an ID to ensure that the test script has a unique identifier to use when referring to that Flex control.
- Make these identifiers as human-readable as possible to make it easier for the user to identify that object in the testing script. For example, set the id property of a Panel container inside a TabNavigator to `submit_panel` rather than `panel1` or `p1`.

When working with Silk4J, an object is automatically given a name depending on certain tags, for instance, id, childIndex. If there is no value for the id property, Silk4J uses other properties, such as the childIndex property. Assigning a value to the id property makes the testing scripts easier to read.

Avoiding Duplication of Objects

Automation agents rely on the fact that some properties of object instances will not be changed during run time. If you change the Flex component property that is used by Silk4J as the object name at run time, unexpected results can occur. For example, if you create a Button control without an `automationName` property, and you do not initially set the value of its label property, and then later set the value of the `label` property, problems might occur. In this case, Silk4J uses the value of the label property of Button controls to identify an object if the automationName property is not set. If you later set the value of the `label` property, or change the value of an existing label, Silk4J identifies the object as a new object and does not reference the existing object.

To avoid duplicating objects:

- Understand what properties are used to identify objects in the agent and avoid changing those properties at run time.
- Set unique, human-readable id or `automationName` properties for all objects that are included in the recorded script.

Flex AutomationName and AutomationIndex Properties

The Flex Automation API introduces the `automationName` and `automationIndex` properties. If you provide the `automationName`, Silk4J uses this value for the recorded window declaration's name. Providing a meaningful name makes it easier for Silk4J to identify that object. As a best practice, set the value of the `automationName` property for all objects that are part of the application's test.

Use the `automationIndex` property to assign a unique index value to an object. For instance, if two objects share the same name, assign an index value to distinguish between the two objects.

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 runs out of memory eventually. The Flex Control 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, see the *Adobe Flex Release Notes*.

Flex Class Definition File

The class definition file contains information about all instrumented Flex components. This file (or files) provides information about the components that can send events during recording and accept events for replay. The class definition file also includes the definitions for the supported properties.

Silk Test contains several XML files that describe all classes/events/properties for the common Flex common and specialized controls. Those XML files are located in the `<Silk Test_install_directory>\ng\agent\plugins\com.borland.fastxd.techdomain.flex.agent_<version>\config\automationEnvironment` folder.

If you provide your own XML file, you must copy your XML file into this folder. When the Silk Test Agent starts and initializes support for Adobe Flex, it reads the contents of this directory.

The XML file has the following basic structure:

```
<TypeInformation>

<ClassInfo>

<Implementation />

<Events>

<Event />

…

</Events>

<Properties>

<Property />

…

</Properties>

</ClassInfo>

</TypeInformation>
```

Setting the Flex automationName Property

The `automationName` property defines the name of a component as it appears in tests. The default value of this property varies depending on the type of component. For example, the `automationName` for a Button control is the label of the Button control. Sometimes, the `automationName` is the same as the `id` property for the control, but this is not always the case.

For some components, Flex sets the value of the `automationName` property to a recognizable attribute of that component. This helps testers recognize the component in their tests. Because testers typically do not have access to the underlying source code of the application, having a control's visible property define that control can be useful. For example, a Button labeled "Process Form Now" appears in the test as `FlexButton("Process Form Now")`.

If you implement a new component, or derive from an existing component, you might want to override the default value of the `automationName` property. For example, UIComponent sets the value of the `automationName` to the component's `id` property by default. However, some components use their own methods for setting the value. For example, in the Flex Store sample application, containers are used to create the product thumbnails. A container's default `automationName` would not be very useful because it is the same as the container's `id` property. So, in Flex Store, the custom component that generates a product thumbnail explicitly sets the `automationName` to the product name to make testing the application easier.

The following example from the CatalogPanel.mxml custom component sets the value of the `automationName` property to the name of the item as it appears in the catalog. This is more recognizable than the default automation name.

```
thumbs[i].automationName = catalog[i].name;
```

The following example sets the `automationName` property of the ComboBox control to "Credit Card List"; rather than using the `id` property, the testing tool typically uses "Credit Card List" to identify the ComboBox in its scripts:

```
<?xml version="1.0"?>

<!-- at/SimpleComboBox.mxml -->

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>

<![CDATA[

[Bindable]

public var cards: Array = [

{label:"Visa", data:1},

{label:"MasterCard", data:2},

{label:"American Express", data:3}

];


[Bindable]

public var selectedItem:Object;

]]>

</mx:Script>

<mx:Panel title="ComboBox Control Example">

<mx:ComboBox id="cb1" dataProvider="{cards}"

width="150"

close="selectedItem=ComboBox(event.target).selectedItem"

automationName="Credit Card List"

/>
```

```
<mx:VBox width="250">

<mx:Text width="200" color="blue" text="Select a type of credit card." />

<mx:Label text="You selected: {selectedItem.label}"/>

<mx:Label text="Data: {selectedItem.data}"/>

</mx:VBox>

</mx:Panel>

</mx:Application>
```

Setting the value of the `automationName` property ensures that the object name will not change at run time. This helps to eliminate unexpected results.

If you set the value of the `automationName` property, tests use that value rather than the default value. For example, by default, Silk4J uses a Button control's label property as the name of the Button in the script. If the label changes, the script can break. You can prevent this from happening by explicitly setting the value of the `automationName` property.

Buttons that have no label, but have an icon, are recorded by their index number. In this case, ensure that you set the `automationName` property to something meaningful so that the tester can recognize the Button in the script. After the value of the `automationName` property is set, do not change the value during the component's life cycle. For item renderers, use the `automationValue` property rather than the `automationName` property. To use the `automationValue` property, override the `createAutomationIDPart()` method and return a new value that you assign to the `automationName` property, as the following example shows:

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml">

<mx:Script>

<![CDATA[

import mx.automation.IAutomationObject;

override public function

createAutomationIDPart(item:IAutomationObject):Object {

var id:Object = super.createAutomationIDPart(item); id["automationName"] =
id["automationIndex"];

return id;

}

]]>

</mx:Script>

</mx:List>
```

Use this technique to add index values to the children of any container or list-like control. There is no method for a child to specify an index for itself.

Setting the Flex Select Method to Use Name or Index

You can record Flex `Select` methods using the `Name` or `Index` of the control that you select. By default, Silk Test records `Select` methods using the name of the control. However, you can change your environment to record `Select` events using the index for the control, or you can switch between the name and index for recording.

1. Determine which class you want to modify to use the Index.

   You can record `Select` events using the index for the following controls:

   - `FlexList`
   - `FlexTree`
   - `FlexDataGrid`
   - `FlexOLAPDataGrid`
   - `FlexComboBox`
   - `FlexAdvancedDataGrid`
2. Determine which XML file is related to the class that you want to modify.

   The XML files related to the preceding controls include: `FlexCommonControls.xml`, `AdvancedDataGrid.xml`, or `OLAPDataGrid.xml`.
3. Navigate to the XML files that are related to the class that you want to modify.

   The XML files are located in the `<Silk Test_install_directory>\ng\agent\plugins\com.borland.fastxd.techdomain.flex.agent_<version>\config\automationEnvironment` folder.
4. Make the following adaptations in the corresponding XML file.

   ```
   <ClassInfo Extends="FlexList" Name="FlexControlName"
   EnableIndexBasedSelection=”true” >

   …

   </ClassInfo>
   ```

   For instance, you might use "`FlexList`" as the " `FlexControlName`" and modify the `FlexCommonControls.xml` file.

   With this adaption the IndexBasedSelection is used for recording `FlexList::SelectIndex` events.

   Note: Setting the `EnableIndexBasedSelection=` to `false` in the code or removing the boolean returns recording to using the name (`FlexList::Select` events).
5. Re-start your Flex application and the Open Agent in order for these changes to become active.

Coding Flex Containers

Containers differ from other kinds of controls because they are used both to record user interactions (such as when a user moves to the next pane in an Accordion container) and to provide unique locations for controls in the testing scripts.

Adding and Removing Containers from the Automation Hierarchy

In general, the automated testing feature reduces the amount of detail about nested containers in its scripts. It removes containers that have no impact on the results of the test or on the identification of the controls from the script. This applies to containers that are used exclusively for layout, such as the HBox, VBox, and Canvas containers, except when they are being used in multiple-view navigator containers, such as the ViewStack, TabNavigator, or Accordion containers. In these cases, they are added to the automation hierarchy to provide navigation.

Many composite components use containers, such as Canvas or VBox, to organize their children. These containers do not have any visible impact on the application. As a result, you typically exclude these containers from testing because there is no user interaction and no visual need for their operations to be recordable. By excluding a container from testing, the related test script is less cluttered and easier to read.

To exclude a container from being recorded (but not exclude its children), set the container's `showInAutomationHierarchy` property to false. This property is defined by the `UIComponent` class, so all containers that are a subclass of `UIComponent` have this property. Children of containers that are not visible in the hierarchy appear as children of the next highest visible parent.

The default value of the `showInAutomationHierarchy` property depends on the type of container. For containers such as Panel, Accordion, Application, DividedBox, and Form, the default value is `true`; for other containers, such as Canvas, HBox, VBox, and FormItem, the default value is `false`.

The following example forces the VBox containers to be included in the test script's hierarchy:

```
<?xml version="1.0"?>
<!-- at/NestedButton.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Panel title="ComboBox Control Example">
<mx:HBox id="hb">
<mx:VBox id="vb1" showInAutomationHierarchy="true">
<mx:Canvas id="c1">
<mx:Button id="b1" automationName="Nested Button 1" label="Click Me" />
</mx:Canvas>
</mx:VBox>
<mx:VBox id="vb2" showInAutomationHierarchy="true">
<mx:Canvas id="c2">
<mx:Button id="b2" automationName="Nested Button 2" label="Click Me 2" />
</mx:Canvas>
</mx:VBox>
</mx:HBox>
</mx:Panel>
</mx:Application>
```

Multiview Containers

Avoid using the same label on multiple tabs in multiview containers, such as the TabNavigator and Accordion containers. Although it is possible to use the same labels, this is generally not an acceptable UI design practice and can cause problems with control identification in your testing environment.

Flex Automation Testing Workflow

The Silk4J workflow for testing Flex applications includes:

- Automated Testing Initialization
- Automated Testing Recording

- Automated Testing Playback

Flex Automated Testing Initialization

When the user launches the Flex application, the following initialization events occur:

1. The automation initialization code associates component delegate classes with component classes.
2. The component delegate classes implement the IAutomationObject interface.
3. An instance for the AutomationManager is created in the mixin `init()` method. (The AutomationManager is a mixin.)
4. The SystemManager initializes the application. Component instances and their corresponding delegate instances are created. Delegate instances add event listeners for events of interest.
5. The Silk4J FlexTechDomain is a mixin. In the FlexTechDomain `init()` method, the FlexTechDomain registers for the `SystemManager.APPLICATION_COMPLETE` event. When the event is received, it creates a FlexTechDomain instance.
6. The FlexTechDomain instance connects via a TCP/IP socket to the Silk Test Agent on the same machine that registers for record/playback functionality.
7. The FlexTechDomain requests information about the automation environment. This information is stored in XML files and is forwarded from the Silk Test Agent to the FlexTechDomain.

Flex Automated Testing Recording

When the user records a new test in Silk4J for a Flex application, the following events occur:

1. Silk4J calls the Silk Test Agent to start recording. The Agent forwards this command to the FlexTechDomain instance.
2. FlexTechDomain notifies AutomationManager to start recording by calling `beginRecording()`. The AutomationManager adds a listener for the AutomationRecordEvent.RECORD event from the SystemManager.
3. The user interacts with the application. For example, suppose the user clicks a Button control.
4. The `ButtonDelegate.clickEventHandler()` method dispatches an AutomationRecordEvent event with the click event and Button instance as properties.
5. The AutomationManager record event handler determines which properties of the click event to store based on the XML environment information. It converts the values into proper type or format. It dispatches the record event.
6. The FlexTechDomain event handler receives the event. It calls the `AutomationManager.createID()` method to create the AutomationID object of the button. This object provides a structure for object identification. The AutomationID structure is an array of AutomationIDParts. An AutomationIDPart is created by using IAutomationObject. (The UIComponent.id, automationName, automationValue, childIndex, and label properties of the Button control are read and stored in the object. The label property is used because the XML information specifies that this property can be used for identification for the Button.)
7. FlexTechDomain uses the `AutomationManager.getParent()` method to get the logical parent of Button. The AutomationIDPart objects of parent controls are collected at each level up to the application level.
8. All the AutomationIDParts are included as part of the AutomationID object.
9. The FlexTechDomain sends the information in a call to Silk4J.
10. When the user stops recording, the `FlexTechDomain.endRecording()` method is called.

Flex Automated Testing Playback

When the user clicks the **Playback** button in Silk4J, the following events occur:

1. For each script call, Silk4J contacts the Silk Test Agent and sends the information for the script call to be executed. This information includes the complete window declaration, the event name, and parameters.

2. The Silk Test Agent forwards that information to the FlexTechDomain.
3. The FlexTechDomain uses `AutomaionManager.resolveIDToSingleObject` with the window declaration information. The AutomationManager returns the resolved object based on the descriptive information (automationName, automationIndex, id, and so on).
4. Once the Flex control is resolved, FlexTechDomain calls `AutomationManager.replayAutomatableEvent()` to replay the event.
5. The `AutomationManager.replayAutomatableEvent()` method invokes the `IAutomationObject.replayAutomatableEvent()` method on the delegate class. The delegate uses the `IAutomationObjectHelper.replayMouseEvent()` method (or one of the other replay methods, such as `replayKeyboardEvent()`) to play back the event.
6. If there are verifications in your script, FlexTechDomain invokes `AutomationManager.getProperties()` to access the values that must be verified.

Styles in Adobe Flex Applications

For applications developed in Adobe Flex 3.x, Silk4J does not distinguish between styles and properties. As a result, styles are exposed as properties. However, with Adobe Flex 4.x, all new Flex controls, which are prefixed with `Spark`, such as `SparkButton`, do not expose styles as properties. As a result, the `GetProperty()` and `GetPropertyList()` methods for Flex 4.x controls do not return styles, such as `color` or `fontSize`, but only properties, such as `text` and `name`.

The `GetStyle(string styleName)` method returns values of styles as a string. To find out which styles exist, refer to the Adobe help located at: *http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/package-detail.html*.

If the style is not set, a `StyleNotSetException` occurs during playback.

For the Flex 3.x controls, such as `FlexTree`, you can use `GetProperty()` to retrieve styles. Or, you can use `GetStyle()`. Both the `GetProperty()` and `GetStyle()` methods work with Flex 3.x controls.

Calculating the Color Style

In Flex, the color is represented as number. It can be calculated using the following formula:

```
red*65536 + green*256 + blue
```

Example

In the following example, the script verifies whether the font size is 12. The number 16711680 calculates as 255*65536 + 0*256 + 0. This represents the color red, which the script verifies for the background color.

```
Assert.That(control.GetStyle("fontSize"), [Is].EqualTo("12"))
Assert.That(label.GetStyle("backgroundColor"),
[Is].EqualTo("16711680"))
```

Configuring Flex Applications for Adobe Flash Player Security Restrictions

The security model in Adobe Flash Player 10 has changed from earlier versions. When you record tests that use Flash Player, recording works as expected. However, when you play back tests, unexpected results occur when high-level clicks are used in certain situations. For instance, a **File Reference** dialog box cannot be opened programmatically and when you attempt to play back this scenario, the test fails because of security restrictions.

To work around the security restrictions, you can perform a low-level click on the button that opens the dialog box. To create a low-level click, add a parameter to the `click` method.

For example, instead of using `SparkButton.click()`, use `SparkButton.click(MouseButton.LEFT)`. A `click()` without parameters is a high-level click and a click with parameters (such as the button) is replayed as a low-level click.

1. Record the steps that use Flash Player.
2. Navigate to the `Click` method and add a parameter.
   For example, to open the **Open File** dialog box, specify:

   ```
   SparkButton("@caption='Open File Dialog…'").click(MouseButton.LEFT)
   ```

   When you play back the test, it works as expected.

Attributes for Adobe Flex Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Flex applications include:

- automationName
- caption (similar to automationName)
- automationClassName (e.g. `FlexButton`)
- className (the full qualified name of the implementation class, e.g. `mx.controls.Button`)
- automationIndex (the index of the control in the view of the FlexAutomation, e.g. `index:1`)
- index (similar to automationIndex but without the prefix, e.g. `1`)
- id (the id of the control)
- windowId (similar to id)
- label (the label of the control)
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Java AWT/Swing Support

Silk4J provides built-in support for testing applications or applets that use the Java AWT/Swing controls. When you configure an application or applet that uses Java AWT/Swing, Silk4J automatically provides support for testing standard AWT/Swing controls.

Note: You can also test Java SWT controls embedded in Java AWT/Swing applications or applets as well as Java AWT/Swing controls embedded in Java SWT applications.

Note: Image click recording is not supported for applications or applets that use the Java AWT/Swing controls.

Sample Applications

Silk Test provides a sample Swing 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** > **Silk Test** > **Sample Applications** > **Java Swing** > **Swing Test Application**.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Supported Controls

For a complete list of the controls available for Java AWT/Swing testing, view a list of the supported Swing classes in the API Reference:

- `com.borland.silktest.jtf.swing` - contains Java Swing specific classes
- `com.borland.silktest.jtf.common.types` - contains data types

Attributes for Java AWT/Swing Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java AWT/Swing include:

- caption
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text input field, the caption of the closest label at the left side or above the control is used.
- name
- accessibleName
- *Swing only*: All custom object definition attributes set in the widget with `SetClientProperty("propertyName", "propertyValue")`

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Dynamically Invoking Java Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods of the SWT, AWT, or Swing widget
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- Primitive types (boolean, integer, long, double, string)

  Both primitive types, such as `int`, and object types, such as `java.lang.Integer` are supported. Primitive types are widened if necessary, allowing, for example, to pass an `int` where a `long` is expected.
- Enum types

  Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the enum type, `java.sql.ClientInfoStatus` you can use the string values of `REASON_UNKNOWN`, `REASON_UNKNOWN_PROPERTY`, `REASON_VALUE_INVALID`, or `REASON_VALUE_TRUNCATED`
- Lists

  Allows calling methods with list, array, or var-arg parameters. Conversion to an array type is done automatically, provided the elements of the list are assignable to the target array type.
- Other controls

  Control parameters can be passed or returned as TestObject.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.

Configuring Silk4J to Launch an Application that Uses the Java Network Launching Protocol (JNLP)

Applications that start using the Java Network Launching Protocol (JNLP) require additional configuration in Silk4J. Because these applications are started from the Web, you must manually configure the application configuration to start the actual application and launch the "Web Start". Otherwise, the test will fail on playback unless the application is already running.

1. Create a project and test class for the test application.
2. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.
3. Edit the base state to ensure that the Web Start launches during playback.
   a) Click **Edit Base State**. The **Edit Base State** dialog box opens.
   b) In the **Executable** text box, type the absolute path for the javaws.exe.
   For example, you might type:
   ```
   %ProgramFiles%\Java\jre6\bin\javaws.exe
   ```
   c) In the **Command Line Arguments** text box, type the command line pattern that includes the URL to the Web Start.
   ```
   "<url-to-jnlp-file>"
   ```

For example, for the SwingSet3 application, type:

```
"http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp"
```

d) Click **OK**.

4. Click **OK**. The test uses the base state to start the web-start application and the application configuration executable pattern to attach to javaw.exe to execute the test.

When you run the test, a warning states that the application configuration EXE file does not match the base state EXE file. You can disregard the message because the test executes as expected.

Determining the priorLabel in the Java AWT/Swing Technology Domain

To determine the priorLabel in the Java AWT/Swing technology domain, all labels and groups in the same window as the target control are considered. The decision is then made based upon the following criteria:

- Only labels either above or to the left of the control, and groups surrounding the control, are considered as candidates for a priorLabel.
- If a parent of the control is a `JViewPort` or a `ScrollPane`, the algorithm works as if the parent is the window that contains the control, and nothing outside is considered relevant.
- In the simplest case, the label closest to the control is used as the priorLabel.
- If two labels have the same distance to the control, and one is to the left and the other above the control, the left one is preferred.
- If no label is eligible, the caption of the closest group is used.

Java SWT and Eclipse RCP Support

Silk Test provides built-in support for testing applications that use widgets from the Standard Widget Toolkit (SWT) controls. When you configure a Java SWT/RCP application, Silk Test automatically provides support for testing standard Java SWT/RCP controls.

Silk Test supports:

- Testing Java SWT controls embedded in Java AWT/Swing applications as well as Java AWT/Swing controls embedded in Java SWT applications.
- Testing Java SWT applications.
- Any Eclipse-based application that uses SWT widgets for rendering. Silk Test supports both Eclipse IDE-based applications and RCP-based applications.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from http://supportline.microfocus.com/productdoc.aspx.

Supported Controls

For a complete list of the widgets available for SWT testing, see *Java SWT Class Reference*.

Java SWT Class Reference

When you configure a Java SWT application, Silk4J automatically provides built-in support for testing standard Java SWT controls.

Java SWT Custom Attributes

You can add custom attributes to a test application to make a test more stable. For example, in Java SWT, the developer implementing the GUI can define an attribute (for example, `'silkTestAutomationId'`) for a widget that uniquely identifies the widget in the application. A tester using Silk4J can then add that attribute to the list of custom attributes (in this case, `'silkTestAutomationId'`), and can identify controls by that unique ID. Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index will change whenever another widget is added before the one you have defined already.

If more than one object is assigned the same custom attribute value, all the objects with that value will return when you call the custom attribute. For example, if you assign the unique ID, `'loginName'` to two different text fields, both fields will return when you call the `'loginName'` attribute.

Java SWT Example

If you create a button in the application that you want to test using the following code:

```
Button myButton = Button(parent, SWT.NONE);

myButton.setData("SilkTestAutomationId", "myButtonId");
```

To add the attribute to your XPath query string in your test, you can use the following query:

```
Dim button =
desktop.PushButton("@SilkTestAutomationId='myButton'")
```

To enable a Java SWT application for testing custom attributes, the developers must include custom attributes in the application. Include the attributes using the `org.swt.widgets.Widget.setData(String key, Object value)` method.

Attributes for Java SWT Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Dynamically Invoking Java Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods of the SWT, AWT, or Swing widget
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- Primitive types (boolean, integer, long, double, string)

  Both primitive types, such as `int`, and object types, such as `java.lang.Integer` are supported. Primitive types are widened if necessary, allowing, for example, to pass an `int` where a `long` is expected.
- Enum types

  Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the enum type, `java.sql.ClientInfoStatus` you can use the string values of `REASON_UNKNOWN`, `REASON_UNKNOWN_PROPERTY`, `REASON_VALUE_INVALID`, or `REASON_VALUE_TRUNCATED`
- Lists

  Allows calling methods with list, array, or var-arg parameters. Conversion to an array type is done automatically, provided the elements of the list are assignable to the target array type.
- Other controls

  Control parameters can be passed or returned as TestObject.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.

.NET Support

Silk Test provides built-in support for testing .NET applications including:

- Windows Forms (Win Forms) applications
- Windows Presentation Foundation (WPF) applications
- Microsoft Silverlight applications

For details about supported versions, click **Start** > **Programs** > **Silk** > **Silk Test** > **Release Notes** to view the *Release Notes*.

Windows Forms Support

Silk4J provides built-in support for testing .NET standalone and No-Touch Windows Forms (Win Forms) applications. However, side-by-side execution is supported only on standalone applications. Silk4J can record and play back controls embedded in:

- Framework version 2.0
- Framework version 3.0
- Framework version 3.5

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Object Recognition

The name that was given to an element in the application is used as `automationId` attribute for the locator if available. As a result, most objects can be uniquely identified using only this attribute.

Supported Controls

For a complete list of the record and replay controls available for Win Forms testing, see *Windows Forms Class Reference*.

Windows Forms Class Reference

When you configure a Windows Forms application, Silk4J automatically provides built-in support for testing standard Windows Forms controls.

Attributes for Windows Forms Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows Forms applications include:

- automationid
- caption
- windowid
- priorlabel (For controls that do not have a caption, the priorlabel is used as the caption automatically. For controls with a caption, it may be easier to use the caption.)

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Dynamically Invoking Windows Forms Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

The invoke Method

For a Windows Forms or a WPF control, you can use the `invoke` method to call the following methods:

- Public methods that the MSDN defines for the control.
- Public static methods that the MSDN defines.
- User-defined public static methods of any type.

First Example for the invoke Method

For an object of the Silk4J type `DataGrid`, you can call all methods that MSDN defines for the type `System.Windows.Forms.DataGrid`.

To call the method `IsExpanded` of the `System.Windows.Forms.DataGrid` class, use the following code:

```
//Java code
boolean isExpanded = (Boolean) dataGrid.invoke("IsExpanded", 3);
```

Second Example for the invoke Method

To invoke the static method `String.compare(String s1, String s2)` inside the AUT, use the following code:

```
//Java code
int result = (Integer)
mainWindow.invoke("System.String.Compare", "a", "b");
```

Third Example for the invoke Method

This example shows how you can dynamically invoke the user-generated method `GetContents`.

You can write code which you can use to interact with a control in the application under test (AUT), in this example an `UltraGrid`. Instead of creating complex dynamic invoke calls to retrieve the contents of the `UltraGrid`, you can generate a new method `GetContents` and then just dynamically invoke the new method.

In Visual Studio, the following code in the AUT defines the `GetContents` method as a method of the `UltraGridUtil` class:

```
//C# code, because this is code in the AUT
namespace UltraGridExtensions {
  public class UltraGridUtil {
    /// <summary>
    /// Retrieves the contents of an UltraGrid as nested list
    /// </summary>
    /// <param name="grid"></param>
    /// <returns></returns>
    public static List<List<string>>
GetContents(Infragistics.Win.UltraWinGrid.UltraGrid grid) {
      var result = new List<List<string>>();
      foreach (var row in grid.Rows) {
        var rowContent = new List<string>();
        foreach (var cell in row.Cells) {
          rowContent.Add(cell.Text);
        }
        result.Add(rowContent);
      }
      return result;
    }
  }
}
```

The code for the `UltraGridUtil` class needs to be added to the AUT. You can do this in the following ways:

- The application developer can compile the code for the class into the AUT. The assembly needs to be already loaded.
- You can create a new assembly that is loaded into the AUT during test execution.

To load the assembly, you can use the following code:

```
FormsWindow.LoadAssembly(String assemblyFileName)
```

You can load the assembly by using the full path, for example:

```
mainWindow.LoadAssembly("C:/temp/ultraGridExtensions.dll")
```

When the code for the UltraGridUtil class is in the AUT, you can add the following code to your test script to invoke the `GetContents` method:

```
List<List<String>> contents =
mainWindow.invoke("UltraGridExtensions.UltraGridUtil.GetContents
", ultraGrid);
```

The `mainWindow` object, on which the `invoke` method is called, only identifies the AUT and can be replaced by any other object in the AUT.

The invokeMethods Method

For a Windows Forms or a WPF control, you can use the `invokeMethods` method to invoke a sequence of nested methods. You can call the following methods:

- Public methods that the MSDN defines for the control.
- Public static methods that the MSDN defines.
- User-defined public static methods of any type.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods and properties that the MSDN defines for the control.
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

  Silk4J types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
- Enum types

  Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visiblity` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
- .NET structs and objects

  .NET struct and object parameters must be passed as a list. The elements in the list must match one constructor for the .NET object in the test application. For example, if the method expects a parameter of the .NET type `System.Windows.Vector`, you can pass a list with two integers. This works because the `System.Windows.Vector` type has a constructor with two integer arguments.
- Other controls

  Control parameters can be passed or returned as TestObject.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.

Windows Presentation Foundation (WPF) Support

Silk4J provides built-in support for testing Windows Presentation Foundation (WPF) applications. Silk4J supports standalone WPF applications and can record and play back controls embedded in .NET version 3.5 or later.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Supported Controls

For a complete list of the controls available for WPF testing, see *WPF Class Reference*.

All supported WPF classes for Silk4J WPF support start with the prefix *WPF*, such as `WPFWindow` and `WPFListBox`.

Supported methods and properties for WPF controls depend on the actual implementation and runtime state. The methods and properties may differ from the list that is defined for the corresponding class. To determine the methods and properties that are supported in a specific situation, use the following code:

- `GetPropertyList()`
- `GetDynamicMethodList()`

For additional information abut WPF, refer to *MSDN*.

WPF Class Reference

When you configure a WPF application, Silk4J automatically provides built-in support for testing standard WPF controls.

Attributes for Windows Presentation Foundation (WPF) Applications

Supported attributes for WPF applications include:

- *automationId*
- *caption*
- *className*
- *name*
- All dynamic locator attributes.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Object Recognition

To identify components within WPF scripts, you can specify the *automationId*, *caption*, *className*, or *name*. The name that is given to an element in the application is used as the *automationId* attribute for the locator if available. As a result, most objects can be uniquely identified using only this attribute. For example, a locator with an *automationId* might look like: `//WPFButton[@automationId='okButton']"`.

If you define an *automationId* and any other attribute, only the *automationId* is used during replay. If there is no *automationId* defined, the *name* is used to resolve the component. If neither a *name* nor an *automationId* are defined, the *caption* value is used. If no caption is defined, the *className* is used. We recommend using the *automationId* because it is the most useful property.

Attribute Type	Description	Example
automationId	An ID that was provided by the developer of the test application.	`//WPFButton[@automationId='okButton']"`
name	The name of a control. The Visual Studio designer automatically assigns a name to every control that is created with the designer. The application developer uses this name to identify the control in the application code.	`//WPFButton[@name='okButton']"`
caption	The text that the control displays. When testing a localized application in multiple languages, use the automationId or name attribute instead of the caption.	`//WPFButton[@automationId='Ok']"`
className	The simple .NET class name (without namespace) of the WPF control. Using the class name attribute can help to identify a custom control that is derived from a standard WPF control that Silk4J recognizes.	`//WPFButton[@className='MyCustomButton']"`

During recording, Silk4J creates a locator for a WPF control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has a *automationId* and a *name*, Silk4J uses the *automationId* when creating the locator.

The following example shows how an application developer can define a *name* and an *automationId* for a WPF button in the XAML code of the application:

```
<Button Name="okButton" AutomationProperties.AutomationId="okButton"
Click="okButton_Click">Ok</Button>
```

Classes that Derive from the WPFItemsControl Class

Silk4J can interact with classes that derive from `WPFItemsControl`, such as `WPFListBox`, `WPFTreeView`, and `WPFMenu`, in two ways:

- Working with the control

  Most controls contain methods and properties for typical use cases. The items are identified by text or index.
- Working with individual items, such as `WPFListBoxItem`, `WPFTreeViewItem`, or `WPFMenuItem`

  For advanced use cases, use individual items. For example, use individual items for opening the context menu on a specific item in a list box, or clicking a certain position relative to an item.

Custom WPF Controls

Generally, Silk4J provides record and playback support for all standard WPF controls.

Silk4J handles custom controls based on the way the custom control is implemented. You can implement custom controls by using the following approaches:

- Deriving classes from `UserControl`

  This is a typical way to create compound controls. Silk4J recognizes these user controls as `WPFUserControl` and provides full support for the contained controls.
- Deriving classes from standard WPF controls, such as `ListBox`

  Silk4J treats these controls as an instance of the standard WPF control that they derive from. Record, playback, and recognition of children may not work if the user control behavior differs significantly from its base class implementation.
- Using standard controls that use templates to change their visual appearance

  Low-level replay might not work in certain cases. Switch to high-level playback mode in such cases. To change the replay mode, use the **Script Options** dialog box and change the **OPT_REPLAY_MODE** option.

Silk4J filters out certain controls that are typically not relevant for functional testing. For example, controls that are used for layout purposes are not included. However, if a custom control derives from an excluded class, specify the name of the related WPF class to expose the filtered controls during recording and playback.

Dynamically Invoking WPF Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

The invoke Method

For a Windows Forms or a WPF control, you can use the `invoke` method to call the following methods:

- Public methods that the MSDN defines for the control.
- Public static methods that the MSDN defines.
- User-defined public static methods of any type.

First Example for the invoke Method

For an object of the Silk4J type `DataGrid`, you can call all methods that MSDN defines for the type `System.Windows.Forms.DataGrid`.

To call the method `IsExpanded` of the `System.Windows.Forms.DataGrid` class, use the following code:

```
//Java code
boolean isExpanded = (Boolean) dataGrid.invoke("IsExpanded", 3);
```

Second Example for the invoke Method

To invoke the static method `String.compare(String s1, String s2)` inside the AUT, use the following code:

```
//Java code
int result = (Integer)
mainWindow.invoke("System.String.Compare", "a", "b");
```

Third Example for the invoke Method

This example shows how you can dynamically invoke the user-generated method `GetContents`.

You can write code which you can use to interact with a control in the application under test (AUT), in this example an `UltraGrid`. Instead of creating complex dynamic invoke calls to retrieve the contents of the `UltraGrid`, you can generate a new method `GetContents` and then just dynamically invoke the new method.

In Visual Studio, the following code in the AUT defines the `GetContents` method as a method of the `UltraGridUtil` class:

```
//C# code, because this is code in the AUT
namespace UltraGridExtensions {
  public class UltraGridUtil {
    /// <summary>
    /// Retrieves the contents of an UltraGrid as nested list
    /// </summary>
    /// <param name="grid"></param>
    /// <returns></returns>
    public static List<List<string>>
GetContents(Infragistics.Win.UltraWinGrid.UltraGrid grid) {
      var result = new List<List<string>>();
      foreach (var row in grid.Rows) {
        var rowContent = new List<string>();
        foreach (var cell in row.Cells) {
          rowContent.Add(cell.Text);
        }
        result.Add(rowContent);
      }
      return result;
    }
  }
}
```

The code for the `UltraGridUtil` class needs to be added to the AUT. You can do this in the following ways:

- The application developer can compile the code for the class into the AUT. The assembly needs to be already loaded.
- You can create a new assembly that is loaded into the AUT during test execution.

To load the assembly, you can use the following code:

```
FormsWindow.LoadAssembly(String assemblyFileName)
```

You can load the assembly by using the full path, for example:

```
mainWindow.LoadAssembly("C:/temp/ultraGridExtensions.dll")
```

When the code for the UltraGridUtil class is in the AUT, you can add the following code to your test script to invoke the `GetContents` method:

```
List<List<String>> contents =
mainWindow.invoke("UltraGridExtensions.UltraGridUtil.GetContents
", ultraGrid);
```

The `mainWindow` object, on which the `invoke` method is called, only identifies the AUT and can be replaced by any other object in the AUT.

The invokeMethods Method

For a Windows Forms or a WPF control, you can use the `invokeMethods` method to invoke a sequence of nested methods. You can call the following methods:

- Public methods that the MSDN defines for the control.
- Public static methods that the MSDN defines.
- User-defined public static methods of any type.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods and properties that the MSDN defines for the control.
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

  Silk4J types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
- Enum types

  Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visiblity` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
- .NET structs and objects

  .NET struct and object parameters must be passed as a list. The elements in the list must match one constructor for the .NET object in the test application. For example, if the method expects a parameter of the .NET type `System.Windows.Vector`, you can pass a list with two integers. This works because the `System.Windows.Vector` type has a constructor with two integer arguments.
- WPF controls

  WPF control parameters can be passed as `TestObject`.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.
- A string for all other types

  Call `ToString` on returned .NET objects to retrieve the string representation

Example

For example, when an application developer creates a custom calculator control that offers the following methods and the following property:

```
public void Reset()
public int Add(int number1, int number2)
public System.Windows.Vector StrechVector(System.Windows.Vector
vector, double
factor)
public String Description { get;}
```

The tester can call the methods directly from his test. For example:

```
customControl.invoke("Reset");
int sum = customControl.invoke("Add", 1, 2);
// the vector can be passed as list of integer
List<Integer> vector = new ArrayList<Integer>();
vector.add(3);
vector.add(4);
// returns "6;8" because this is the string representation of
the .NET object
String strechedVector = customControl.invoke("StrechVector",
vector, 2.0);
String description = customControl.getProperty("Description");
```

Setting WPF Classes to Expose During Recording and Playback

Silk4J filters out certain controls that are typically not relevant for functional testing. For example, controls that are used for layout purposes are not included. However, if a custom control derives from an excluded class, specify the name of the related WPF class to expose the filtered controls during recording and playback.

Specify the names of any WPF classes that you want to expose during recording and playback. For example, if a custom class called *MyGrid* derives from the WPF `Grid` class, the objects of the *MyGrid* custom class are not available for recording and playback. `Grid` objects are not available for recording and playback because the `Grid` class is not relevant for functional testing since it exists only for layout purposes. As a result, `Grid` objects are not exposed by default. In order to use custom classes that are based on classes that are not relevant to functional testing, add the custom class, in this case *MyGrid*, to the **OPT_WPF_CUSTOM_CLASSES** option. Then you can record, playback, find, verify properties, and perform any other supported actions for the specified classes.

1. Click **Silk4J** > **Edit Options**.
2. Click the plus sign (+) next to **Record** in the **Options** menu tree. The **Record** options display in the right side panel.
3. Click **WPF**.
4. In the **Custom WPF class names** grid, type the name of the class that you want to expose during recording and playback.

   Separate class names with a comma.
5. Click **OK**.

Silverlight Application Support

Microsoft Silverlight (Silverlight) is an application framework for writing and running rich internet applications, with features and purposes similar to those of Adobe Flash. The run-time environment for Silverlight is available as a plug-in for most web browsers.

Silk4J provides built-in support for testing Silverlight applications. Silk4J supports Silverlight applications that run in a browser as well as out-of-browser and can record and play back controls in .NET version 3.5 or later.

The following applications, that are based on Silverlight, are supported:

- Silverlight applications that run in Internet Explorer.
- Silverlight applications that run in Mozilla Firefox.
- Out-of-Browser Silverlight applications.

Supported Controls

Silk4J includes record and replay support for Silverlight controls.

For a complete list of the controls available for Silverlight testing, see the *Silverlight Class Reference*.

Prerequisites

The support for testing Silverlight applications in Microsoft Windows XP requires the installation of Service Pack 3 and the Update for Windows XP with the Microsoft User Interface Automation that is provided in Windows 7. You can download the update from *http://www.microsoft.com/download/en/details.aspx?id=13821*.

Note: The Microsoft User Interface Automation needs to be installed for the Silverlight support. If you are using a Windows operating system and the Silverlight support does not work, you can install the update with the Microsoft User Interface Automation, which is appropriate for your operating system, from *http://support.microsoft.com/kb/971513*.

Silverlight Class Reference

When you configure a Silverlight application, Silk4J automatically provides built-in support for testing standard Silverlight controls.

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

- *automationId*
- *caption*
- *className*
- *name*
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

To identify components within Silverlight scripts, you can specify the *automationId*, *caption*, *className*, *name* or any dynamic locator attribute. The *automationId* can be set by the application developer. For example, a locator with an *automationId* might look like `//SLButton[@automationId="okButton"]`.

We recommend using the *automationId* because it is typically the most useful and stable attribute.

Attribute Type	Description	Example
automationId	An identifier that is provided by the developer of the application under test. The Visual Studio designer automatically assigns an *automationId* to every control that is created with the designer. The application developer uses this ID to identify the control in the application code.	`//SLButton[@automationId="okButton"]`
caption	The text that the control displays. When testing a localized application in multiple languages, use the *automationId* or *name* attribute instead of the *caption*.	`//SLButton[@caption="Ok"]`
className	The simple .NET class name (without namespace) of the Silverlight control. Using the *className* attribute can help to identify a custom control that is derived from a standard Silverlight control that Silk4J recognizes.	`//SLButton[@className='MyCustomButton']`
name	The name of a control. Can be provided by the developer of the application under test.	`//SLButton[@name="okButton"]`

Attention: The *name* attribute in XAML code maps to the locator attribute *automationId*, not to the locator attribute *name*.

During recording, Silk4J creates a locator for a Silverlight control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has an *automationId* and a *name*, Silk4J uses the *automationId*, if it is unique, when creating the locator.

The following table shows how an application developer can define a Silverlight button with the text "Ok" in the XAML code of the application:

XAML Code for the Object	Locator to Find the Object from Silk Test
`<Button>Ok</Button>`	`//SLButton[@caption="Ok"]`
`<Button Name="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`
`<Button AutomationProperties.AutomationId="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`

XAML Code for the Object	Locator to Find the Object from Silk Test
`<Button AutomationProperties.Name="okButton">Ok</Button>`	`//SLButton[@name="okButton"]`

Dynamically Invoking Silverlight Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types.

  Silk4J types include primitive types, for example boolean, int, and string, lists, and other types, for example Point and Rect.
- Enum types.

  Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visiblity` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
- .NET structs and objects.

  Pass .NET struct and object parameters as a list. The elements in the list must match one constructor for the .NET object in the test application. For example, if the method expects a parameter of the .NET type `System.Windows.Vector`, you can pass a list with two integers. This works because the `System.Windows.Vector` type has a constructor with two integer arguments.
- Other controls.

  Control parameters can be passed as `TestObject`.

Supported Methods and Properties

The following methods and properties can be called:

- All public methods and properties that the MSDN defines for the `AutomationElement` class. For additional information, see *http://msdn.microsoft.com/en-us/library/system.windows.automation.automationelement.aspx*.
- All methods and properties that MSUIA exposes. The available methods and properties are grouped in "patterns". Pattern is a MSUIA specific term. Every control implements certain patterns. For an overview of patterns in general and all available patterns see *http://msdn.microsoft.com/en-us/library/ms752362.aspx*. A custom control developer can provide testing support for the custom control by implementing a set of MSUIA patterns.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types.
- All methods that have no return value return NULL.
- A string for all other types.

  To retrieve this string representation, call the `ToString` method on returned .NET objects in the application under test.

Example

A `TabItem` in Silverlight, which is an item in a `TabControl`.

```
tabItem.invoke("SelectionItemPattern.Select");
mySilverligtObject.getProperty("IsPassword");
```

Scrolling in Silverlight

Silk4J provides two different sets of scrolling-related methods and properties, depending on the Silverlight control.

- The first type of controls includes controls that can scroll by themselves and therefore do not expose the scrollbars explicitly as children. For example combo boxes, panes, list boxes, tree controls, data grids, auto complete boxes, and others.
- The second type of controls includes controls that cannot scroll by themselves but expose scrollbars as children for scrolling. For example text fields.

This distinction in Silk4J exists because the controls in Silk4J implement scrolling in those two ways.

Controls that support scrolling

In this case, scrolling-related methods and property are available for the control that contains the scrollbars. Therefore, Silk4J does not expose scrollbar objects.

Examples

The following command scrolls a list box to the bottom:

```
listBox.SetVerticalScrollPercent(100)
```

The following command scrolls the list box down by one unit:

```
listBox.ScrollVertical(ScrollAmount.SmallIncrement)
```

Controls that do not support scrolling

In this case the scrollbars are exposed. No scrolling-related methods and properties are available for the control itself. The horizontal and vertical scrollbar objects enable you to scroll in the control by specifying the increment or decrement, or the final position, as a parameter in the corresponding API functions. The increment or decrement can take the values of the ScrollAmount enumeration. For additional information, refer to the Silverlight documentation. The final position is related to the position of the object, which is defined by the application designer.

Examples

The following command scrolls a vertical scrollbar within a text box to position 15:

```
textBox.SLVerticalScrollBar().ScrollToPosition(15)
```

The following command scrolls a vertical scrollbar within a text box to the bottom:

```
textBox.SLVerticalScrollBar().ScrollToMaximum()
```

Troubleshooting when Testing Silverlight Applications

Silk4J cannot see inside the Silverlight application and no green rectangles are drawn during recording

The following reasons may cause Silk4J to be unable to see inside the Silverlight application:

Reason	Solution
You use a Mozilla Firefox version prior to 4.0.	Use Mozilla Firefox 4.0 or later.
You use a Silverlight version prior to 3.	Use Silverlight 3 (Silverlight Runtime 4) or Silverlight 4 (Silverlight Runtime 4).

Your Silverlight application is running in windowless mode.	Silk4J does not support Silverlight applications that run in windowless mode. To test such an application, you need to change the Web site where your Silverlight application is running. Therefore you need to set the `windowless` parameter in the object tag of the HTML or ASPX file, in which the Silverlight application is hosted, to `false`.
	The following sample code sets the `windowless` parameter to `false`:
	`<object ...>`
	`  <param name="windowless" value="false"/>`
	`  ...`
	`</object>`

Rumba Support

Rumba is the world's premier Windows desktop terminal emulation solution. Silk Test provides built-in support for recording and replaying Rumba.

When testing with Rumba, please consider the following:

- The Rumba version must be compatible to the Silk Test version. Versions of Rumba prior to version 8.1 are not supported.
- All controls that surround the green screen in Rumba are using basic WPF functionality (or Win32).
- The supported Rumba desktop types are:
  - Mainframe Display
  - AS400 Display

For a complete list of the record and replay controls available for Rumba testing, see the *Rumba Class Reference*.

Rumba Class Reference

When you configure a Rumba application, Silk4J automatically provides built-in support for testing standard Rumba controls.

Enabling and Disabling Rumba

Rumba is the world's premier Windows desktop terminal emulation solution. Rumba provides connectivity solutions to mainframes, mid-range, UNIX, Linux, and HP servers.

Enabling Support

Before you can record and replay Rumba scripts, you need to enable support:

1. Install Rumba desktop client software version 8.1 or later.
2. Click **Start** > **Programs** > **Silk** > **Silk Test** > **Administration** > **Rumba plugin** > **Enable Silk Test Rumba plugin**.

Disabling Support

Click **Start** > **Programs** > **Silk** > **Silk Test** > **Administration** > **Rumba plugin** > **Disable Silk Test Rumba plugin**.

Locator Attributes for Identifying Rumba Controls

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. Supported attributes include:

caption
The text that the control displays.

priorlabel
Since input fields on a form normally have a label explaining the purpose of the input, the intention of **priorlabel** is to identify the text input field, **RumbaTextField**, by the text of its adjacent label field, **RumbaLabel**. If no preceding label is found in the same line of the text field, or if the label at the right side is closer to the text field than the left one, a label on the right side of the text field is used.

StartRow
This attribute is not recorded, but you can manually add it to the locator. Use **StartRow** to identify the text input field, **RumbaTextField**, that starts at this row.

StartColumn
This attribute is not recorded, but you can manually add it to the locator. Use **StartColumn** to identify the text input field, **RumbaTextField**, that starts at this column.

All dynamic locator attributes.
For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Using Screen Verifications with Rumba

To automatically insert screen verifications in Rumba, turn on the following option in the **Options** dialog box: **Record** > **General** > **Record Screen Verifications**.

To manually insert screen verifications:

1. In your test, click the **Create Verification Type Logic** button to open the **Test Logic Designer - Verification**.
2. Click **Next**.
3. Select **The Contents of a Screen**.

   Any excluded objects as identified in **Tools** > **Options** > **Record** > **Rumba** > **Excluded Objects** will be used. You can customize these further in the **Properties** window of the test after you finish performing this procedure.
4. Click **Next**.
5. Click the **Identify** button.
6. Select the control on the Rumba Screen that you want to identify. The whole screen will be captured.
7. Click **Next**.
8. Click **Finish**.

SAP Support

Silk4J provides built-in support for testing SAP client/server applications based on the Windows-based GUI module.

Note: You can only test SAP applications with Silk4J if you have a Premium license for Silk4J. For additional information on the licensing modes, see *Licensing Information*.

Note: If you use SAP NetWeaver with Internet Explorer or Firefox, Silk4J tests the application using the xBrowser technology domain.

Note: Check the Release Notes for the latest version information and known issues.

Supported Controls

For a complete list of the record and replay controls available for SAP testing, see the *SAP Class Reference*.

For a list of supported attributes, see *Attributes for SAP Applications*.

SAP Class Reference

When you configure a SAP application, Silk4J automatically provides built-in support for testing standard SAP controls.

The classes included in the SAP class reference, along with all included properties and methods, are part of the SAP Automation module that is directly accessible through Silk4J.

Note: The interface, including the underlying algorithms and the behavior of the interface is not under the control of Silk4J.

Attributes for SAP Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for SAP include:

- automationId
- caption

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Dynamically Invoking SAP Methods

Call dynamic methods on objects with the `invoke` method. For information on the supported automation methods of an object, refer to the SAP GUI Scripting documentation.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control.
- All public methods that the SAP automation interface defines
- If the control is a custom control that is derived from a standard control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

  Silk4J types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
- UI controls

  UI controls can be passed or returned as TestObject.

Returned Values

The following values are returned for properties and methods that have a return value:

- The correct value for all built-in Silk4J types. These types are listed in the *Supported Parameter Types* section.
- All methods that have no return value return `null`.

Dynamically Invoking Methods on SAP Controls

When Silk4J cannot record actions against an SAP control, you can record the actions with the recorder that is available in SAP and then dynamically invoke the recorded methods in a Silk4J script. By doing so, you can replay actions against SAP controls that you cannot record.

1. To record the actions that you want to perform against the control, use the **SAP GUI Scripting** tool that is available in SAP.

   For additional information on the **SAP GUI Scripting** tool, refer to the SAP documentation.
2. Open the recorded actions from the location to which the **SAP GUI Scripting** tool has saved them and see what methods were recorded.
3. In Silk4J, dynamically invoke the recorded methods from your script.

Examples

For example, if you want to replay pressing a special control in the SAP UI, which is labeled *Test* and which is a combination of a button and a list box, and selecting the sub-menu *subsub2* of the control, you can record the action with the recorder that is available in SAP. The resulting code will look like the following:

```
session.findById("wnd[0]/usr/cntlCONTAINER/shellcont/
shell").pressContextButton "TEST"
session.findById("wnd[0]/usr/cntlCONTAINER/shellcont/
shell").selectContextMenuItem "subsub2"
```

Now you can use the following code to dynamically invoke the methods `pressContextButton` and `selectContextMenuItem` in your script in Silk4J:

```
.SapToolbarControl("shell
ToolbarControl").invoke("pressContextButton", "TEST")
.SapToolbarControl("shell
ToolbarControl").invoke("selectContextMenuItem", "subsub2")
```

Replaying this code will press the control in the SAP UI and select the sub-menu.

Configuring Automation Security Settings for SAP

Before you launch an SAP application, you must configure the security warning settings. Otherwise, a security warning, `A script is trying to attach to the GUI`, displays each time a test plays back an SAP application.

1. In **Windows Control Panel**, choose **SAP Configuration**. The **SAP Configuration** dialog box opens.
2. In the **Design Selection** tab, uncheck the **Notify When a Script Attaches to a Running SAP GUI**.

Windows API-Based Application Support

Silk4J provides built-in support for testing Microsoft Windows API-based applications. Several objects exist in Microsoft applications that Silk4J can better recognize if you enable Accessibility. For example, without enabling Accessibility Silk4J records only basic information about the menu bar in Microsoft Word and the tabs that appear in Internet Explorer versions later than version 7.0. However, with Accessibility enabled, Silk4J fully recognizes those objects. You can also improve Silk4J object recognition by defining a new window, if necessary.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Supported Controls

For a complete list of the record and replay controls available for Windows-based testing, see *Win32 Class Reference*.

Win32 Class Reference

When you configure a Win32 application, Silk4J automatically provides built-in support for testing standard Windows API-based controls.

Attributes for Windows API-based Client/Server Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows API-based client/server applications include:

- caption
- windowid
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text box, the caption of the closest label at the left side or above the control is used.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Determining the priorLabel in the Win32 Technology Domain

To determine the priorLabel in the Win32 technology domain, all labels and groups in the same window as the target control are considered. The decision is then made based upon the following criteria:

- Only labels either above or to the left of the control, and groups surrounding the control, are considered as candidates for a priorLabel.
- In the simplest case, the label closest to the control is used as the priorLabel.
- If two labels have the same distance to the control, the priorLabel is determined based upon the following criteria:
  - If one label is to the left and the other above the control, the left one is preferred.
  - If both levels are to the left of the control, the upper one is preferred.
  - If both levels are above the control, the left one is preferred.
- If the closest control is a group control, first all labels within the group are considered according to the rules specified above. If no labels within the group are eligible, then the caption of the group is used as the priorLabel.

xBrowser Support

Use the xBrowser technology domain to test Web applications that use:

- Internet Explorer
- Mozilla Firefox
- Google Chrome
- Embedded browser controls

The xBrowser technology domain supports the testing of plain HTML pages as well as AJAX pages. AJAX pages require additional, sophisticated strategies for object recognition and synchronization.

Note: You must record tests for Web applications using Internet Explorer. To create tests that use another supported browser, record them with Internet Explorer and play them back with the other browser. Or, you can manually create tests for the other browser using the **Identify Objects** dialog box to identify the locators in the supported browser that you want to use.

Note: Before you record or playback Web applications, disable all browser add-ons that are installed in your system. To disable add-ons in Internet Explorer, click **Tools** > **Internet Options**, click the **Programs** tab, click **Manage add-ons**, select an add-on and then click **Disable**.

For information about supported versions, any known issues, and workarounds, refer to the *Release Notes*.

Sample Applications

To access the Silk Test sample Web applications, go to:

- *http://demo.borland.com/InsuranceWebExtJS/*
- *http://demo.borland.com/gmopost*

Test Objects for xBrowser

Silk4J uses the following classes to model a Web application:

Class	Description
`BrowserApplication`	Exposes the main window of a Web browser and provides methods for tabbing.
`BrowserWindow`	Provides access to tabs and embedded browser controls and provides methods for navigating to different pages.
`DomElement`	Exposes the DOM tree of a Web application (including frames) and provides access to all DOM attributes. Specialized classes are available for several DOM elements.

Object Recognition for xBrowser Objects

The xBrowser technology domain supports dynamic object recognition.

Test cases use locator strings to find and identify objects. A typical locator includes a locator name and at least one locator attribute, such as `"//LocatorName[@locatorAttribute='value']"`.

Locator Names

With other technology types, such as Java SWT, locator names are created using the class name of the test object. With xBrowser, the tag name of the DOM element can also be used as locator name. The following locators describe the same element:

1. Using the tag name: `"//a[@href='http://www.microfocus.com']"`
2. Using the class name: `"//DomLink[@href='http://www.microfocus.com']"`

To optimize replay speed, use tag names rather than class names.

Locator Attributes

All DOM attributes can be used as locator string attributes. For example, the element `<button automationid='123'>Click Me</button>` can be identified using the locator `"//button[@automationid='123']"`.

Recording Locators

Silk4J uses a built-in locator generator when recording test cases and using the **Identify Object** dialog box. You can configure the locator generator to improve the results for a specific application.

Page Synchronization for xBrowser

Synchronization is performed before and after every method call. A method call is not started and does not end until the synchronization criteria is met.

Note: Any property access is not synchronized.

Synchronization Modes

Silk4J includes synchronization modes for HTML and AJAX.

Using the HTML mode ensures that all HTML documents are in an interactive state. With this mode, you can test simple Web pages. If more complex scenarios with Java script are used, it might be necessary to manually script synchronization functions, such as:

- `WaitForObject`

- `WaitForProperty`
- `WaitForDisappearance`
- `WaitForChildDisappearance`

The AJAX mode synchronization waits for the browser to be in a kind of idle state, which is especially useful for AJAX applications or pages that contain AJAX components. Using the AJAX mode eliminates the need to manually script synchronization functions (such as wait for objects to appear or disappear, wait for a specific property value, and so on), which eases the script creation process dramatically. This automatic synchronization is also the base for a successful record and playback approach without manual script adoptions.

Troubleshooting

Because of the true asynchronous nature of AJAX, generally there is no real idle state of the browser. Therefore, in rare situations, Silk4J will not recognize an end of the invoked method call and throws a timeout error after the specified timeout period. In these situations, it is necessary to set the synchronization mode to HTML at least for the problematic call.

Note: Regardless of the page synchronization method that you use, in tests where a Flash object retrieves data from a server and then performs calculations to render the data, you must manually add a synchronization method to your test. Otherwise, Silk4J does not wait for the Flash object to complete its calculations. For example, you might use `Thread.sleep(millisecs)`.

Some AJAX frameworks or browser applications use special HTTP requests, which are permanently open in order to retrieve asynchronous data from the server. These requests may let the synchronization hang until the specified synchronization timeout expires. To prevent this situation, either use the HTML synchronization mode or specify the URL of the problematic request in the **Synchronization exclude list** setting.

Use a monitoring tool to determine if playback errors occur because of a synchronization issue. For instance, you can use FindBugs, *http://findbugs.sourceforge.net/*, to determine if an AJAX call is affecting playback. Then, add the problematic service to the **Synchronization exclude list**.

Note: If you exclude a URL, synchronization is turned off for each call that targets the URL that you specified. Any synchronization that is needed for that URL must be called manually. For example, you might need to manually add `WaitForObject` to a test. To avoid numerous manual calls, exclude URLs for a concrete target, rather than for a top-level URL, if possible.

Configuring Page Synchronization Settings

You can configure page synchronization settings for each individual test or you can set global options that apply to all tests in the **Script Options** dialog box.

To add the URL to the exclusion filter, specify the URL in the **Synchronization exclude list** in the **Script Options** dialog box.

To configure individual settings for tests, record the test and then insert code to override the global playback value. For example, to exclude the time service, you might type:

```
desktop.setOption(CommonOptions.OPT_XBROWSER_SYNC_EXCLUDE_URLS,
    Arrays.asList("timeService"));
```

Comparing API Playback and Native Playback for xBrowser

Silk4J supports API playback and native playback for Web applications. If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using API playback.

Advantages of native playback include:

- With native playback, the agent emulates user input by moving the mouse pointer over elements and pressing the corresponding elements. As a result, playback works with most applications without any modifications.
- Native playback supports plug-ins, such as Flash and Java applets, and applications that use AJAX, while high-level API recordings do not.

Advantages of API playback include:

- With API playback, the Web page is driven directly by DOM events, such as `onmouseover` or `onclick`.
- Scripts that use API playback do not require that the browser be in the foreground.
- Scripts that use API playback do not need to scroll an element into view before clicking it.
- Generally API scripts are more reliable since high-level user input is insensitive to pop-up windows and user interaction during playback.
- API playback is faster than native playback.

You can use the **Script Options** dialog box to configure the types of functions to record and whether to use native user input.

Differences Between API and Native Playback Functions

The `DomElement` class provides different functions for API playback and native playback.

The following table describes which functions use API playback and which use native playback.

	API Playback	Native Playback

Mouse Actions	`DomClick`
	`DomDoubleClick`
	`DomMouseMove`
		`Click`
		`DoubleClick`
		`MoveMouse`
		`PressMouse`
		`ReleaseMouse`

Keyboard Actions	not available	`TypeKeys`

Specialized Functions	`Select`
	`SetText`
	etc.
		not available

Setting Browser Recording Options

Specify custom attributes, browser attributes to ignore while recording, and whether to record native user input instead of DOM functions.

Silk4J includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results. You can use any property that is available in the respective technology as a custom attribute given that they are either numbers (integers, doubles), strings, item identifiers, or enumeration values.

In xBrowser applications, you can also retrieve arbitrary properties and then use those properties as custom attributes. To achieve optimal results, add a custom automation ID to the elements that you want to interact with in your test.

1. Click **Silk4J** > **Edit Options**.

2. Click the plus sign (+) next to **Record** in the **Options** menu tree. The **Record** options display in the right side panel.
3. Click **xBrowser**.
4. To add a custom attribute for a Web application, in the **Custom attributes** text box, type the attributes that you want to use.

   Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index might change whenever another object is added before one you have defined already.

   Note: To include custom attributes in a Web application, add them to the html tag. For example type, `<input type='button' MyAutomationID='abc' value='click me' />` to add an attribute called `MyAutomationID`.

   If more than one object is assigned the same custom attribute value, all the objects with that value will return when you call the custom attribute. For example, if you assign the unique ID, `loginName` to two different text fields, both fields will return when you call the `loginName` attribute.

   Note: There is a 62 character limit to attribute names.
5. In the **Locator attribute name exclude list** text box, type the attribute names to ignore while recording.

   Use this list to specify attributes that change frequently, such as size, width, height, and style. You can include the wildcards '*' and '?' in the **Locator attribute name exclude list**.

   For example, if you do not want to record attributes named `height`, add the `height` attribute name to the list.

   Separate attribute names with a comma.
6. In the **Locator attribute value exclude list** text box, type the attribute values to ignore while recording. For example, if you do not want to record attributes assigned the value of `x-auto`, add the `x-auto` attribute value to the list.

   Some AJAX frameworks generate attribute values that change every time the page is reloaded. Use this list to ignore such values. You can also use wildcards in this list.

   Separate attribute names with a comma.
7. To record native user input instead of DOM functions, from the **Record native user input** list box, select **Yes**.

   For example, to record `Click` instead of `DomClick` and `TypeKeys` instead of `SetText`, select **Yes**.

   If your application uses a plug-in or AJAX, specify **Yes** to use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.
8. Click **OK**.

Setting Mouse Move Preferences

Specify whether mouse move actions are recorded for Web applications, Win32 applications, and Windows Forms applications that use mouse move events. You cannot record mouse move events for child domains of the xBrowser technology domain, for example Adobe Flex and Swing.

1. Click **Silk4J** > **Edit Options**.
2. Click the plus sign (+) next to **Record** in the **Options** menu tree. The **Record** options display in the right side panel.
3. Click **Recording**.
4. To record mouse move actions, check the OPT_RECORD_MOUSEMOVES option..

   Silk4J will only record mouse move events that cause changes to the hovered element or its parent in order to keep scripts short.

5. If you record mouse move actions, in the **Record mouse move delay** text box, specify how many milliseconds the mouse has to be motionless before a `MoveMouse` action is recorded

   By default this value is set to 200.

   Mouse move actions are only recorded if the mouse stands still for this time. A shorter delay will result in more unexpected move mouse actions, a longer delay will require you to keep the mouse still to record an action.
6. Click **OK**.

Browser Configuration Settings for xBrowser

Several browser settings help to sustain stable test executions. Although Silk4J works without changing any settings, there are several reasons that you might want to change the browser settings.

Increase replay speed	Use `about:blank` as home page instead of a slowly loading Web page
Avoid unexpected behavior of the browser	• Disable pop up windows and warning dialog boxes. • Disable auto-complete features. • Disable password wizards.
Prevent malfunction of the browser	Disable unnecessary third-party plugins.

The following sections describe where these settings are located in the corresponding browser.

Internet Explorer

The browser settings are located at **Tools** > **Internet Options**. The following table lists options that you might want to adjust.

Tab	Option	Configuration	Comments
General	Home page	Set to `about:blank`.	Minimize start up time of new tabs.
General	Tabs	• Disable warning when closing multiple tabs. • Enable to switch to new tabs when they are created.	• Avoid unexpected dialog boxes. • Links that open new tabs might not replay correctly otherwise.
Privacy	Pop-up blocker	Disable pop up blocker.	Make sure your Web site can open new windows.
Content	AutoComplete	Turn off completely	• Avoid unexpected dialog boxes. • Avoid unexpected behavior when typing keys.
Programs	Manage add-ons	Only enable add-ons that are absolutely required.	• Third-party add-ons might contain bugs. • Possibly not compatible to Silk4J.
Advanced	Settings	• Disable **Automatically check for Internet Explorer updates**. • Enable **Disable script debugging (Internet Explorer)**. • Enable **Disable script debugging (Other)**. • Disable **Enable automatic crash recovery**.	Avoid unexpected dialog boxes.

Tab	Option	Configuration	Comments
		• Disable **Display notification about every script error**. • Disable all **Warn ...** settings	

Mozilla Firefox

In Mozilla Firefox, you can edit all settings by navigating a tab to `about:config`. The following table lists options that you might want to adjust. If any of the options do not exist, you can create them by right-clicking the table and choosing **New**.

Option	Value	Comments
app.update.auto	false	Avoid unexpected behavior (disable auto update).
app.update.enabled	false	Avoid unexpected behavior (disable updates in general).
app.update.mode	0	Avoid unexpected dialog boxes (do not prompt for new updates).
app.update.silent	true	Avoid unexpected dialog boxes (do not prompt for new updates).
browser.sessionstore.resume_from_crash	false	Avoid unexpected dialog boxes (warning after a browser crash).
browser.sessionstore.max_tabs_undo	0	Enhance performance. Controls how many closed tabs are kept track of through the Session Restore service.
browser.sessionstore.max_windows_undo	0	Enhance performance. Controls how many closed windows are kept track of through the Session Restore service.
browser.sessionstore.resume_session_once	false	Avoid unexpected dialog boxes. Controls whether the last saved session is restored once the next time the browser starts.
browser.shell.checkDefaultBrowser	false	Avoid unexpected dialog boxes. Checks if Mozilla Firefox is the default browser.
browser.startup.homepage	"about:blank"	Minimize start up time of new tabs.
browser.startup.page	0	Minimize browser startup time (no start page in initial tab).
browser.tabs.warnOnClose	false	Avoid unexpected dialog boxes (warning when closing multiple tabs).
browser.tabs.warnOnCloseOtherTabs	false	Avoid unexpected dialog boxes (warning when closing other tabs).
browser.tabs.warnOnOpen	false	Avoid unexpected dialog boxes (warning when opening multiple tabs).
dom.max_chrome_script_run_time	180	Avoid unexpected dialog boxes (warning when XUL code takes too long to execute, timeout in seconds).
dom.max_script_run_time	600	Avoid unexpected dialog boxes (warning when script code takes too long to execute, timeout in seconds).
dom.successive_dialog_time_limit	0	Avoid unexpected **Prevent page from creating additional dialogs** dialog box.
extensions.update.enabled	false	Avoid unexpected dialog boxes. Disables automatic extension update.

Google Chrome

You do not have to change browser settings for Google Chrome. Silk4J automatically starts Google Chrome with the appropriate command-line parameters.

Configuring the Locator Generator for xBrowser

The Open Agent includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results.

A well-defined locator relies on attributes that change infrequently and therefore requires less maintenance. Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index might change when another object is added.

To achieve optimal results, add a custom automation ID to the elements that you want to interact with in your test. In Web applications, you can add an attribute to the element that you want to interact with, such as `<div myAutomationId=”my unique element name” />`. This approach can eliminate the maintenance associated with locator changes.

1. Click **Silk4J** > **Edit Options** and then click the **Custom Attributes** tab.
2. If you use custom automation IDs, from the **Select a TechDomain** list box, select **xBrowser** and then add the IDs to the list.

   The custom attributes list contains attributes that are suitable for locators. If custom attributes are available, the locator generator uses these attributes before any other attribute. The order of the list also represents the priority in which the attributes are used by the locator generator. If the attributes that you specify are not available for the objects that you select, Silk4J uses the default attributes for xBrowser.
3. Click the **Browser** tab.
4. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording.

   For example, use this list to specify attributes that change frequently, such as size, width, height, and style. You can include the wildcards ‘*’ and ‘?’ in the Locator attribute name blacklist.

   Separate attribute names with a comma.
5. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording.

   Some AJAX frameworks generate attribute values that change every time the page is reloaded. Use this list to ignore such values. You can also use wildcards in this list.

   Separate attribute values with a comma.
6. Click **OK**.

   You can now record or manually create a test case.

Playing Back a Script in Another Browser Instead of Internet Explorer

You must create scripts for Web applications with Internet Explorer. However, you can playback scripts in another supported browser if necessary.

For information about supported versions, any known issues, and workarounds, refer to the *Release Notes*.

1. Record a script or create one manually for the Web application that you want to test.

   You must use Internet Explorer to record a script.
2. Click **Silk4J** > **Edit Application Configurations** The **Edit Application Configurations** dialog box displays.

3. Click **Configure Browser**. The **Edit Base State** dialog box displays.
4. Click on the browser icon in the **Browser Type** section to select the browser that you want to use.
5. Click **OK**.

Prerequisites for Replaying Tests with Google Chrome

Command-line parameters

When you use Google Chrome to replay a test or to record locators, Google Chrome is started with the following command:

```
%LOCALAPPDATA%\Google\Chrome\Application\chrome.exe
  --enable-logging
  --log-level=1
  --disable-web-security
  --disable-hang-monitor
  --disable-prompt-on-repost
  --dom-automation
  --full-memory-crash-report
  --no-default-browser-check
  --no-first-run
  --homepage=about:blank
  --disable-web-resources
  --disable-preconnect
  --enable-logging
  --log-level=1
  --safebrowsing-disable-auto-update
  --test-type=ui
  --noerrdialogs
  --metrics-recording-only
  --allow-file-access-from-files
  --disable-tab-closeable-state-watcher
  --allow-file-access
  --disable-sync
  --testing-channel=NamedTestingInterface:st_42
```

When you use the wizard to hook on to an application, these command-line parameters are automatically added to the base state. If an instance of Google Chrome is already running when you start testing, without the appropriate command-line parameters, Silk4J closes Google Chrome and tries to restart the browser with the command-line parameters. If the browser cannot be restarted, an error message displays.

Note: The command-line parameter `disable-web-security` is required when you want to record or replay cross-domain documents.

Limitations for Testing with Google Chrome

The support for playing back tests and recording locators with Google Chrome is not as complete as the support for the other supported browsers. The following list lists the known limitations for playing back tests and recording locators with Google Chrome:

- Silk Test does not support testing child technology domains of the xBrowser domain with Google Chrome. For example Adobe Flex or Microsoft Silverlight are not supported with Google Chrome.
- Silk Test does not provide native support for Google Chrome. You cannot test internal Google Chrome functionality. For example, in a test, you cannot change the currently displayed Web page by adding text to the navigation bar through Win32. As a workaround, you can use API calls to navigate between Web pages. Silk Test supports handling alerts and similar dialog boxes.
- The page synchronization for Google Chrome is not as advanced as for the other supported browsers. Changing the synchronization mode has no impact on the synchronization for Google Chrome.
- Silk Test does not support the methods `TextClick` and `TextSelect` when testing applications with Google Chrome.

- Silk Test does not recognize the **Log In** and **Cancel** buttons in the authentication dialog box of Google Chrome. Use one of the following solutions to work around this limitation:
  - Specify the username and the password in the URL of the website that you want to test. For example, to log in to the website *www.example.com/loginrequired.html*, use the following code:
    ```
    http://myusername:mypassword@example.com/loginrequired.html
    ```
  - Use `TypeKeys` to enter the username and password in the dialog box. For example, use the following code:
    ```
     desktop.find("//Window[@caption='Authentication Required']/
    Control[2]").TypeKeys("myusername")
    desktop.find("//Window[@caption='Authentication Required']/
    Control[1]").TypeKeys("mypassword<Enter>")
    ```

    Note: `Control[2]` is the username field, and `Control[1]` is the password field. The `<Enter>` key at the end of the second `TypeKeys` confirms the entries in the dialog box.
- Silk Test does not recognize opening the **Print** dialog box in Google Chrome by using the Google Chrome menu. To add opening this dialog box in Google Chrome to a test, you have to send **Ctrl+Shift+P** using the `TypeKeys` method. Internet Explorer does not recognize this shortcut, so you have to first record your test in Internet Explorer, and then manually add pressing **Ctrl+Shift+P** to your test.
- When two Google Chrome windows are open at the same time and the second window is detached from the first one, Silk Test does not recognize the elements on the detached Google Chrome window. For example, start Google Chrome and open two tabs. Then detach the second tab from the first one. Silk Test does no longer recognize the elements on the second tab. To recognize elements with Silk Test on multiple Google Chrome windows, use **CTRL+N** to open a new Google Chrome window.
- When you want to test a Web application with Google Chrome and the **Continue running background apps when Google Chrome is closed** check box is checked, Silk Test cannot restart Google Chrome to load the automation support.

xBrowser Frequently Asked Questions

This section includes a collection of questions that you might encounter when testing your Web application.

How do I Verify the Font Type Used for the Text of an Element?

You can access all attributes of the `currentStyle` attribute of a DOM element by separating the attribute name with a `":"`.

Internet Explorer 8 or earlier	`wDomElement.GetProperty("currentStyle:fontName")`
All other browsers, for example Internet Explorer 9 or later and Mozilla Firefox	`wDomElement.GetProperty("currentStyle:font-name")`

What is the Difference Between textContents, innerText, and innerHtml?

- `textContents` is all text contained by an element and all its children that are for formatting purposes only.
- `innerText` returns all text contained by an element and all its child elements.
- `innerHtml` returns all text, including html tags, that is contained by an element.

Consider the following html code.

```
<div id="mylinks">
  This is my <b>link collection</b>:
  <ul>
    <li><a href="www.borland.com">Bye bye <b>Borland</b> </a></li>
    <li><a href="www.microfocus.com">Welcome to <b>Micro Focus</b></a></li>
```

```
  </ul>
</div>
```

The following table details the different properties that return.

Code	Returned Value
`browser.DomElement("//div[@id='mylinks']").GetProperty("textContents")`	`This is my link collection:`

`browser.DomElement("//div[@id='mylinks']").GetProperty("innerText")`	`This is my link collection:Bye bye Borland`
	`Welcome to Micro Focus`

`browser.DomElement("//div[@id='mylinks']").GetProperty("innerHtml")`	`This is my <b>link collection</b>:`
	`<ul>`
	`  <li><a href="www.borland.com">Bye bye`
	`<b>Borland</b></a></li>`
	`  <li><a href="www.microfocus.com">Welcome to`
	`<b>Micro Focus</b></a></li>`
	`</ul>`

Note: In Silk Test 13.5 or later, whitespace in texts, which are retrieved through the `textContents` property of an element, is trimmed consistently across all supported browsers. For some browser versions, this whitespace handling differs to Silk Test versions prior to Silk Test 13.5. You can re-enable the old behavior by setting the OPT_COMPATIBILITY option to a version lower than 13.5.0.

I Configured innerText as a Custom Class Attribute, but it Is Not Used in Locators

A maximum length for attributes used in locator strings exists. `InnerText` tends to be lengthy, so it might not be used in the locator. If possible, use `textContents` instead.

What Should I Take Care Of When Creating Cross-Browser Scripts?

When you are creating cross-browser scripts, you might encounter one or more of the following issues:

- Different attribute values. For example, colors in Internet Explorer are returned as `"# FF0000"` and in Mozilla Firefox as `"rgb(255,0,0)"`.
- Different attribute names. For example, the font size attribute is called `"fontSize"` in Internet Explorer 8 or earlier and is called `"font-size"` in all other browsers, for example Internet Explorer 9 or later and Mozilla Firefox.
- Some frameworks may render different DOM trees.

How Can I See Which Browser I Am Currently Using?

The `BrowserApplication` class provides a property `"browsertype"` that returns the type of the browser. You can add this property to a locator in order to define which browser it matches.

For information about new features, supported platforms and versions, known issues, and work-arounds, refer to the *Silk Test Release Notes*, available from *http://supportline.microfocus.com/productdoc.aspx*.

Examples

To get the browser type, type the following into the locator:

```
browserApplication.GetProperty("browsertype")
```

Additionally, the `BrowserWindow` provides a method `GetUserAgent` that returns the user agent string of the current window.

Which Locators are Best Suited for Stable Cross-Browser Testing?

The built in locator generator attempts to create stable locators. However, it is difficult to generate quality locators if no information is available. In this case, the locator generator uses hierarchical information and indices, which results in fragile locators that are suitable for direct record and replay but ill-suited for stable, daily execution. Furthermore, with cross-browser testing, several AJAX frameworks might render different DOM hierarchies for different browsers.

To avoid this issue, use custom IDs for the UI elements of your application.

Logging Output of My Application Contains Wrong Timestamps

This might be a side effect of the synchronization. To avoid this problem, specify the HTML synchronization mode.

My Test Script Hangs After Navigating to a New Page

This can happen if an AJAX application keeps the browser busy (open connections for Server Push / ActiveX components). Try to set the HTML synchronization mode. Check the *Page Synchronization for xBrowser* topic for other troubleshooting hints.

Recorded an Incorrect Locator

The attributes for the element might change if the mouse hovers over the element. Silk4J tries to track this scenario, but it fails occasionally. Try to identify the affected attributes and configure Silk4J to ignore them.

Rectangles Around Elements in Internet Explorer are Misplaced

- Make sure the zoom factor is set to 100%. Otherwise, the rectangles are not placed correctly.
- Ensure that there is no notification bar displayed above the browser window. Silk4J cannot handle notification bars.

Link.Select Does Not Set the Focus for a Newly Opened Window in Internet Explorer

This is a limitation that can be fixed by changing the Browser Configuration Settings. Set the option to always activate a newly opened window.

DomClick(x, y) Is Not Working Like Click(x, y)

If your application uses the `onclick` event and requires coordinates, the `DomClick` method does not work. Try to use `Click` instead.

FileInputField.DomClick() Will Not Open the Dialog

Try to use `Click` instead.

The Move Mouse Setting Is Turned On but All Moves Are Not Recorded. Why Not?

In order to not pollute the script with a lot of useless `MoveMouse` actions, Silk4J does the following:

- Only records a `MoveMouse` action if the mouse stands still for a specific time.
- Only records `MoveMouse` actions if it observes activity going on after an element was hovered over. In some situations, you might need to add some manual actions to your script.

- Silk4J supports recording mouse moves only for Web applications, Win32 applications, and Windows Forms applications. Silk4J does not support recording mouse moves for child technology domains of the xBrowser technology domain, for example Adobe Flex and Swing.

I Need Some Functionality that Is Not Exposed by the xBrowser API. What Can I Do?

You can use `ExecuteJavaScript()` to execute JavaScript code directly in your Web application. This way you can build a workaround for nearly everything.

Why Are the Class and the Style Attributes Not Used in the Locator?

These attributes are on the ignore list because they might change frequently in AJAX applications and therefore result in unstable locators. However, in many situations these attributes can be used to identify objects, so it might make sense to use them in your application.

Dialog is Not Recognized During Replay

When recording a script, Silk4J recognizes some windows as `Dialog`. If you want to use such a script as a cross-browser script, you have to replace `Dialog` with `Window`, because some browsers do not recognize `Dialog`.

For example, the script might include the following line:

```
/BrowserApplication//Dialog//PushButton[@caption='OK']
```

Rewrite the line to enable cross-browser testing to:

```
/BrowserApplication//Window//PushButton[@caption='OK']
```

Why Do I Get an Invalidated-Handle Error?

This topic describes what you can do when you test a Web application and Silk4J displays the following error message: `The handle for this object has been invalidated.`

This message indicates that something caused the object on which you called a method, for example `WaitForProperty`, to disappear. For example, if something causes the browser to navigate to a new page, during a method call in a Web application, all objects on the previous page are automatically invalidated.

Sometimes the reason for this problem is the built-in synchronization. For example, suppose that the application under test includes a shopping cart, and you have added an item to this shopping cart. You are waiting for the next page to be loaded and for the shopping cart to change its status to `contains items`. If the action, which adds the item, returns too soon, the shopping cart on the first page will be waiting for the status to change while the new page is loaded, causing the shopping cart of the first page to be invalidated. This behavior will result in an invalidated-handle error.

As a workaround, you should wait for an object that is only available on the second page before you verify the status of the shopping cart. As soon as the object is available, you can verify the status of the shopping cart, which is then correctly verified on the second page.

Why Are Clicks Recorded Differently in Internet Explorer 10?

When you record a `Click` on a `DomElement` in Internet Explorer 10 and the `DomElement` is dismissed after the `Click`, then the recording behavior might not be as expected. If another `DomElement` is located beneath the initial `DomElement`, Silk Test records a `Click`, a `MouseMove`, and a `ReleaseMouse`, instead of recording a single `Click`.

A possible workaround for this unexpected recording behavior depends on the application under test. Usually it is sufficient to delete the unnecessary `MouseMove` and `ReleaseMouse` events from the recorded script.

Attributes for Web Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Web applications include:

- caption (supports wildcards *?* and *)
- all DOM attributes (supports wildcards *?* and *)

Note: Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

```
<a>abc
abc</a>
```

Uses the following locator:

```
//A[@innerText='abc abc']
```

xBrowser Class Reference

When you configure an xBrowser application, Silk4J automatically provides built-in support for testing standard xBrowser controls.

64-bit Application Support

Silk4J supports testing 64-bit applications for the following technology types:

- Windows Forms
- Windows Presentation Foundation (WPF)
- Microsoft Windows API-based
- Java AWT/Swing
- Java SWT

Check the *Release Notes* for the most up-to-date information about supported versions, any known issues, and workarounds.

Supported Attribute Types

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. If necessary, you can change the attribute type in one of the following ways:

- Manually typing another attribute type and value.
- Specifying another preference for the default attribute type by changing the **Preferred attribute list** values.

Attributes for Adobe Flex Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Flex applications include:

- automationName
- caption (similar to automationName)
- automationClassName (e.g. `FlexButton`)
- className (the full qualified name of the implementation class, e.g. `mx.controls.Button`)
- automationIndex (the index of the control in the view of the FlexAutomation, e.g. `index:1`)
- index (similar to automationIndex but without the prefix, e.g. `1`)
- id (the id of the control)
- windowId (similar to id)
- label (the label of the control)
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards `?` and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Attributes for Java AWT/Swing Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java AWT/Swing include:

- caption
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text input field, the caption of the closest label at the left side or above the control is used.
- name
- accessibleName
- *Swing only*: All custom object definition attributes set in the widget with `SetClientProperty("propertyName", "propertyValue")`

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards `?` and `*`.

Attributes for Java SWT Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Attributes for SAP Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for SAP include:

- automationId
- caption

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

- *automationId*
- *caption*
- *className*
- *name*
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

To identify components within Silverlight scripts, you can specify the *automationId*, *caption*, *className*, *name* or any dynamic locator attribute. The *automationId* can be set by the application developer. For example, a locator with an *automationId* might look like `//SLButton[@automationId="okButton"]`.

We recommend using the *automationId* because it is typically the most useful and stable attribute.

Attribute Type	Description	Example
automationId	An identifier that is provided by the developer of the application under test. The Visual Studio designer automatically assigns an *automationId* to every control that is created with the designer. The application developer uses this ID to identify the control in the application code.	`//SLButton[@automationId="okButton"]`
caption	The text that the control displays. When testing a localized application in multiple languages, use the *automationId* or *name* attribute instead of the *caption*.	`//SLButton[@caption="Ok"]`
className	The simple .NET class name (without namespace) of the Silverlight control. Using the *className* attribute can help to identify a custom control that is derived from a standard Silverlight control that Silk4J recognizes.	`//SLButton[@className='MyCustomButton']`
name	The name of a control. Can be provided by the developer of the application under test.	`//SLButton[@name="okButton"]`

Attention: The *name* attribute in XAML code maps to the locator attribute *automationId*, not to the locator attribute *name*.

During recording, Silk4J creates a locator for a Silverlight control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has an *automationId* and a *name*, Silk4J uses the *automationId*, if it is unique, when creating the locator.

The following table shows how an application developer can define a Silverlight button with the text "Ok" in the XAML code of the application:

XAML Code for the Object	Locator to Find the Object from Silk Test
`<Button>Ok</Button>`	`//SLButton[@caption="Ok"]`
`<Button Name="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`
`<Button AutomationProperties.AutomationId="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`
`<Button AutomationProperties.Name="okButton">Ok</Button>`	`//SLButton[@name="okButton"]`

Locator Attributes for Identifying Rumba Controls

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. Supported attributes include:

caption
The text that the control displays.

priorlabel
Since input fields on a form normally have a label explaining the purpose of the input, the intention of **priorlabel** is to identify the text input field, **RumbaTextField**, by the text of its adjacent label field, **RumbaLabel**. If no preceding label is found in the same line of the text field, or if the label at the right side is closer to the text field than the left one, a label on the right side of the text field is used.

StartRow
This attribute is not recorded, but you can manually add it to the locator. Use **StartRow** to identify the text input field, **RumbaTextField**, that starts at this row.

StartColumn
This attribute is not recorded, but you can manually add it to the locator. Use **StartColumn** to identify the text input field, **RumbaTextField**, that starts at this column.

All dynamic locator attributes.
For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Attributes for Web Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Web applications include:

- caption (supports wildcards *?* and `*`)
- all DOM attributes (supports wildcards *?* and `*`)

Note: Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

```
<a>abc
abc</a>
```

Uses the following locator:

```
//A[@innerText='abc abc']
```

Attributes for Windows Forms Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows Forms applications include:

- automationid
- caption
- windowid
- priorlabel (For controls that do not have a caption, the priorlabel is used as the caption automatically. For controls with a caption, it may be easier to use the caption.)

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Attributes for Windows Presentation Foundation (WPF) Applications

Supported attributes for WPF applications include:

- *automationId*
- *caption*
- *className*
- *name*
- All dynamic locator attributes.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Object Recognition

To identify components within WPF scripts, you can specify the *automationId*, *caption*, *className*, or *name*. The name that is given to an element in the application is used as the *automationId* attribute for the locator if available. As a result, most objects can be uniquely identified using only this attribute. For example, a locator with an *automationId* might look like: `//WPFButton[@automationId='okButton']"`.

If you define an *automationId* and any other attribute, only the *automationId* is used during replay. If there is no *automationId* defined, the *name* is used to resolve the component. If neither a *name* nor an *automationId* are defined, the *caption* value is used. If no caption is defined, the *className* is used. We recommend using the *automationId* because it is the most useful property.

Attribute Type	Description	Example
automationId	An ID that was provided by the developer of the test application.	`//WPFButton[@automationId='okButton']"`
name	The name of a control. The Visual Studio designer automatically assigns a name to every control that is created with the designer. The application developer uses this name to identify the control in the application code.	`//WPFButton[@name='okButton']"`
caption	The text that the control displays. When testing a localized application in multiple languages, use the automationId or name attribute instead of the caption.	`//WPFButton[@automationId='Ok']"`
className	The simple .NET class name (without namespace) of the WPF control. Using the class name attribute can help to identify a custom control that is derived from a standard WPF control that Silk4J recognizes.	`//WPFButton[@className='MyCustomButton']"`

During recording, Silk4J creates a locator for a WPF control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has a *automationId* and a *name*, Silk4J uses the *automationId* when creating the locator.

The following example shows how an application developer can define a *name* and an *automationId* for a WPF button in the XAML code of the application:

```
<Button Name="okButton" AutomationProperties.AutomationId="okButton"
Click="okButton_Click">Ok</Button>
```

Attributes for Windows API-based Client/Server Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows API-based client/server applications include:

- caption
- windowid
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text box, the caption of the closest label at the left side or above the control is used.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Dynamic Locator Attributes

In a locator for identifying a control during replay you can use a pre-defined set of locator attributes, for example *caption* and *automationId*, which depend on the technology domain. But you can also use every property, including dynamic properties, of a control as locator attribute. A list of available properties for a certain control can be retrieved with the `GetPropertyList` method. All returned properties can be used for identifying a control with a locator.

Note: You can use the `GetProperty` method to retrieve the actual value for a certain property of interest. You can then use this value in your locator.

Example

If you want to identify the button that has the user input focus in a Silverlight application, you can type:

```
browser.Find("//SLButton[@IsKeyboardFocused=true]")
```

or alternatively

```
Dim button = dialog.SLButton("@IsKeyboardFocused=true")
```

This works because Silk4J exposes a property called `IsDefault` for the Silverlight button control.

Example

If you want to identify a button in a Silverlight application with the font size 12 you can type:

```
Dim button = browser.Find("//SLButton[@FontSize=12]")
```

or alternatively

```
Dim button = browser.SLButton("@FontSize=12")
```

This works because the underlying control in the application under test, in this case the Silverlight button, has a property called `FontSize`.

Best Practices for Using Silk4J

As a best practice, we recommend creating a separate method for finding controls that you use often within tests. For example:

```
public Dialog getSaveAsDialog(Desktop desktop) {
 return desktop.find("//Dialog[@caption = 'Save As']");
}
```

The `Find` and `FindAll` methods return a handle for each matching object, which is only valid as long as the object in the application exists. For example, a handle to a dialog is invalid once the dialog is closed. Any attempts to execute methods on this handle after the dialog closes will throw an `InvalidObjectHandleException`. Similarly, handles for DOM objects on a Web page become invalid if the Web page is reloaded. Since it is a common practice to design test methods to be independent of each other and of order of execution, get new handles for the objects in each test method. In order not to duplicate the XPath query, helper methods, like `getSaveAsDialog`, can be created. For example:

```
@Test
public void testSaveAsDialog() {
 // ... some code to open the 'Save As' dialog (e.g by clicking a menu
item) ...
 Dialog saveAsDialog = getSaveAsDialog(desktop);
 saveAsDialog.close();
 // ... some code to open the 'Save As' dialog again
 getSaveAsDialog(desktop).click(); // works as expected
 saveAsDialog.click(); //  fails because an InvalidObjectHandleException is
thrown
}
```

The final line of code fails because it uses the object handle that no longer exists.

Object Recognition

Within Silk4J, literal references to identified objects are referred to as *locators*. Silk4J uses locators to find and identify objects in the application under test (AUT). Locators are a subset of the XPath query language, which is a common XML-based language defined by the World Wide Web Consortium, W3C.

Locator Basic Concepts

Silk4J supports a subset of the XPath query language.

Object Type and Search Scope

A locator typically contains the type of object to identify and a search scope. The search scope is one of the following:

- //
- /

Locators rely on the current object, which is the object for which the locator is specified. The current object is located in the object hierarchy of the application's UI. All locators depend on the position of the current object in this hierarchy, much like a file system.

XPath expressions rely on the *current context*, which is the position of the object in the hierarchy on which the `Find` method was invoked. All XPath expressions depend on this position, much like a file system.

Note:

The object type in a locator for an HTML element is either the HTML tag name or the class name that Silk4J uses for this object. For example, the locators `//a` and `//DomLink`, where `DomLink` is the name for hyperlinks in Silk4J, are equivalent. For all non-HTML based technologies only the Silk4J class name can be used.

Example

- `//a` identifies hyperlink objects in any hierarchy relative to the current object.
- `/a` identifies hyperlink objects that are direct children of the current object.

Note: *<a>* is the HTML tag for hyperlinks on a Web page.

Example

The following code sample identifies the first hyperlink in a browser. This example assumes that a variable with the name *browserWindow* exists in the script that refers to a running browser instance. Here the type is "a" and the current object is *browserWindow*.

```
DomLink link = browserWindow.<DomLink>find("//a");
```

Using Attributes to Identify an Object

To identify an object based on its properties, you can use locator attributes. The locator attributes are specified in square brackets after the type of the object.

Example

The following sample uses the `textContents` attribute to identify a hyperlink with the text *Home*. If there are multiple hyperlinks with the same text, the locator identifies the first one.

```
DomLink link = browserWindow.<DomLink>find(//
a[@textContents='Home']");
```

Locator Syntax

Silk4J supports a subset of the XPath query language to locate UI controls.

The following table lists the constructs that Silk4J supports.

Note: <*a*> is the HTML tag for hyperlinks on a Web page.

Supported Locator Construct	Sample	Description
//	`//a`	Identifies objects that are descendants of the current object. The example identifies hyperlinks on a Web page.
/	`/a`	Identifies objects that are direct children of the current object. Objects located on lower hierarchy levels are not recognized. The example identifies hyperlinks on a Web page that are direct children of the current object.
Attribute	`//a[@textContents='Home']`	Identifies objects by a specific attribute. The example identifies hyperlinks with the text *Home*.
Index	Example 1: `//a[3]` Example 2: `//a[@textContents='Home'][2]`	Identifies a specific occurrence of an object if there are multiple ones. Indices are 1-based in locators. Example 1 identifies the third hyperlink and Example 2 identifies the second hyperlink with the text *Home*.
Logical Operators: and, or, not, =, !=	Example 1: `//a[@textContents='Remove' or @textContents='Delete']` Example 2: `//a[@textContents!='Remove']`	Identifies objects by using logical operators to combine attributes. Example 1 identifies hyperlinks that either have the caption *Remove* or *Delete*, Example 2 identifies hyperlinks with a text that is not *Remove*, and Example 3 shows how to combine different logical operators.

Supported Locator Construct	Sample	Description
	Example 3: `//a[not(@textContents='Delete' or @id='lnkDelete') and @href='*/delete']`	
..	Example 1: `//a[@textContents='Edit']/..` Example 2: `//a[@textContents='Edit']/..//a[@textContents='Delete']`	Identifies the parent of an object. Example 1 identifies the parent of the hyperlink with the text *Edit* and Example 2 identifies a hyperlink with the text *Delete* that has a sibling hyperlink with the text *Edit*.
*	Example 1: `//*[@textContents='Home']` Example 2: `/*/a`	Identifies objects without considering their types, like hyperlink, text field, or button. Example 1 identifies objects with the given text content, regardless of their type, and Example 2 identifies hyperlinks that are second-level descendants of the current object.

The following table lists the locator constructs that Silk4J does not support.

Unsupported Locator Construct	Example
Comparing two attributes with each other.	`//a[@textContents = @id]`
An attribute name on the right side is not supported. An attribute name must be on the left side.	`//a['abc' = @id]`
Combining multiple locators with `and` or `or`.	`//a[@id = 'abc'] or ..//Checkbox`
More than one set of attribute brackets.	`//a[@id = 'abc] [@textContents = '123']` (use `//a [@id = 'abc' and @textContents = '123']` instead)
More than one set of index brackets.	`//a[1][2]`

Any construct that does not explicitly specify a class or the class wildcard, such as including a wildcard as part of a class name.	`//[@id = 'abc']`
	(use `//*[@id = 'abc']` instead)
	`"//*//a[@id='abc']"`

Using Locators

Within Silk4J, literal references to identified objects are referred to as *locators*. For convenience, you can use shortened forms for the locator strings in scripts. Silk4J automatically expands the syntax to use full locator strings when you playback a script. When you manually code a script, you can omit the following parts in the following order:

- The search scope, `//`.

- The object type name. Silk4J defaults to the class name.
- The surrounding square brackets of the attributes, `[ ]`.

When you manually code a script, we recommend that you use the shortest form available.

Note: When you identify an object, the full locator string is captured by default.

The following locators are equivalent:

- The first example uses the full locator string.

  ```
  _desktop.<DomLink>find("//BrowserApplication//BrowserWindow//
  a[@textContents='Home']").select();
  ```

  To confirm the full locator string, use the **Locator Spy** dialog box.
- The second example works when the browser window already exists.

  ```
  browserWindow.<DomLink>find("//a[@textContents='Home']").select();
  ```

  Alternatively, you can use the shortened form.

  ```
  browserWindow.<DomLink>find("@textContents='Home'").select();
  ```

To find an object that has no real attributes for identification, use the index. For instance, to select the second hyperlink on a Web page, you can type:

```
browserWindow.<DomLink>find("//DomLink[2]").select();
```

Additionally, to find the first object of its kind, which might be useful if the object has no real attributes, you can type:

```
browserWindow.<DomLink>find("//DomLink").select();
```

Using Locators to Check if an Object Exists

You can use the `Exists` method to determine if an object exists in the application under test.

The following code checks if a hyperlink with the text *Log out* exists on a Web page:

```
if (browserWindow.exists( "//a[@textContents='Log out']" )) {
  // do something
}
```

Identifying Multiple Objects with One Locator

You can use the `FindAll` method to identify all objects that match a locator rather that only identifying the first object that matches the locator.

Example

The following code example uses the `FindAll` method to retrieve all hyperlinks of a Web page:

```
List<DomLink> links = browserWindow.<DomLink>findAll("//a");
```

Troubleshooting Performance Issues for XPath

When testing applications with a complex object structure, for example complex web applications, you may encounter performance issues, or issues related to the reliability of your scripts. This topic describes how you can improve the performance of your scripts by using different locators than the ones that Silk4J has automatically generated during recording.

Note: In general, we do not recommend using complex locators. Using complex locators might lead to a loss of reliability for your tests. Small changes in the structure of the tested application can break such a complex locator. Nevertheless, when the performance of your scripts is not satisfying, using more specific locators might result in tests with better performance.

The following is a sample element tree for the application MyApplication:

```
Root
  Node id=1
    Leaf id=2
    Leaf id=3
    Leaf id=4
    Leaf id=5
  Node id=6
    Node id=7
      Leaf id=8
      Leaf id=9
    Node id=9
      Leaf id=10
```

You can use one or more of the following optimizations to improve the performance of your scripts:

- If you want to locate an element in a complex object structure , search for the element in a specific part of the object structure, not in the entire object structure. For example, to find the element with the identifier 4 in the sample tree, if you have a query like `Root.Find("//Leaf[@id='4']")`, replace it with a query like `Root.Find("/Node[@id='1']/Leaf[@id='4']")`. The first query searches the entire element tree of the application for leafs with the identifier 4. The first leaf found is then returned. The second query searches only the first level nodes, which are the node with the identifier 1 and the node with the identifier 6, for the node with the identifier 1, and then searches in the subtree of the node with the identifier 1 for all leafs with the identifier 4.
- When you want to locate multiple items in the same hierarchy, first locate the hierarchy, and then locate the items in a loop. If you have a query like `Root.FindAll("/Node[@id='1']/Leaf")`, replace it with a loop like the following:

  ```
  public void test() {
    TestObject node;
    int i;

    node = desktop.find("//Node[@id='1']");
    for (i=1; i<=4; i++)
    node.find("/Leaf[@id='"+i+"']");
  }
  ```

Locator Spy

Use the **Locator Spy** to identify the caption or the XPath locator string for GUI objects. You can copy the relevant XPath locator strings and attributes into methods in your scripts. Additionally, you can manually edit the attributes of the XPath locator strings in your test scripts and validate the changes in the **Locator Spy**. Using the **Locator Spy** ensures that the XPath query string is valid.

Note: The locator attributes table of the **Locator Spy** displays all attributes that you can use in the locator. For Web applications, the table also includes any attributes that you have defined to be ignored during recording.

Object Maps

An object map is a test asset that contains items that associate a logical name (an alias) with a control or a window, rather than the control or window's locator. Once a control is registered in an object map asset, all references to it in scripts are made by its alias, rather than by its actual locator name.

Multiple tests can reference a single object map item definition, which enables users to update that object map definition once and have Silk4J update it in all tests that reference the object map definition.

Example

The following construct shows a definition for a `BrowserWindow` where the locator is used:

```
_desktop.BrowserApplication("cnn_com").BrowserWindow("//
BrowserWindow[1]")
```

The name of the object map asset is *cnn_com*. The locator that can be substituted by an alias in the object map is the following:

```
"//BrowserWindow[1]"
```

The object map entry for the `BrowserWindow` is *BrowserWindow*.

The resulting definition of the `BrowserWindow` in the script is the following:

```
_desktop.BrowserApplication("cnn_com").BrowserWindow("BrowserWin
dow")
```

If the index in the locator changes, you can just change the alias in the object map, instead of having to change every appearance of the locator in your test script. Silk4J will update all tests that reference the object map definition.

Advantages of Using Object Maps

Object maps have the following advantages:

- They substitute complex locator names with descriptive names, which can make scripts easier to read.
- They eliminate dependence on locators, which may change if the test application is modified.
- They simplify test maintenance by applying changes made to a locator for an object map item to all tests that include the corresponding object map item.

Turning Object Maps Off and On

You can configure Silk4J to use the locator name or the alias from the object map during recording.

To use the alias from the object map during recording:

1. Click **Silk4J** > **Edit Options**.
2. Click **Recording**.
3. Check the **Record object maps** setting.

   By default, Silk4J records the alias from the object map during recording. If you set the **Record object maps** setting unchecked, Silk4J records the locator name during recording. You can turn the **Record object maps** setting off and on as you find necessary. However, when a test is recorded with locators, you must re-record it in order to use object map items.

4. If you want Silk4J to use additional attributes of the element when merging object maps during locator recording, check the **Apply smart merge of locators in object maps** setting.

   If the **Apply smart merge of locators in object maps** setting is unchecked, Silk4J uses only the XPath for merging. Additional attributes, which might lead to ambiguous usage of object map IDs in a recorded script, are not used to map locators to existing object map entries.

Note: When you enable the **Record object maps** setting, object map item names display in place of locators throughout Silk4J. For instance, if you view the **Application Configurations** category in the **Properties** pane, you will notice that the **Locator** box shows the object map item name rather than the locator name.

Using Assets in Multiple Projects

In Silk4J, image assets, image verifications, and object maps are referred to as *assets*. If you want to use assets outside of the scope of the project in which they are located, you need to add a direct project dependency from the project in which you want to use the assets to the project in which the assets are located. If you are playing back test using the Java automation, the scope of the assets automatically includes all projects in the classpath.

During replay, when an asset is used, Silk4J firstly searches in the current project for the asset. If Silk4J does not find the asset in the current project, Silk4J additionally searches the projects to which the current project has a project dependency.If the asset is still not found, Silk4J throws an error. If you are playing back test using the Java automation, Silk4J firstly searches in the current project for the asset and then searches all projects in the classpath.

Silk4J persists the order of any found assets.

If assets with the same name exist in more than one project, and you do not want to use the asset that is included in the current project, you can define which specific asset you want to use in any method that uses the asset. To define which asset you want to use, add the asset namespace as a prefix to the asset name when calling the method. The asset namespace defaults to the project name.

Note: When you start working with Silk4J, the asset namespace option is added to the `silk4j.settings` file of every Silk4J project in your workspace that has been created with a previous version of Silk4J.

Example: Adding a project dependency

If the project *ProjectA* contains a test that calls the following code:

```
imageClick("imageAsset");
```

and the image asset *imageAsset* is located in project *ProjectB*, you need to add a direct project dependency from *ProjectA* to *ProjectB*.

Example: Calling a specific asset

If *ProjectA* and *ProjectB* both contain an image asset with the name *anotherImageAsset*, and you explicitly want to click the image asset from *ProjectB*, use the following code:

```
imageClick("ProjectB:anotherImageAsset")
```

Using Object Maps with Web Applications

By default, when you record actions against a Web application, Silk4J creates an object map with the name *WebBrowser* for native browser controls and an object map asset for every Web domain during recording in the *Common* project.

For common browser controls which are not specific for a Web domain, like the main window or the dialog boxes for printing or settings, an additional object map is generated in the current project with the name *WebBrowser*.

In the object map, you can edit the URL pattern by which the object map entries are grouped. When you edit the pattern, Silk4J performs a syntactical validation of the pattern. You can use the wildcards * and ? in the pattern.

Example

When you record some actions on *http://www.borland.com* and *http://www.microfocus.com* and then open the printer dialog, the following three new object map assets are added to the **Asset Browser**:

- WebBrowser
- borland_com
- microfocus_com

Note: Silk4J generates the new object map assets only for projects without an object map. If you record actions against a Web application for which Silk4J already includes an object map that was generated with a version of Silk4J prior to version 14.0, the additionally recorded entries are stored into the existing object map, and there are no additional object map assets generated for the Web domains.

Renaming an Object Map Item

You can manually rename items and locators in an object map.

Warning: Renaming an object map item affects every script that uses that item. For example, if you rename the **Cancel** button object map item from **CancelMe** to **Cancel**, every script that uses **CancelMe** must be changed manually to use **Cancel**.

Object map items must be unique. If you try to add a duplicate object map item, Silk4J notifies you that the object must be unique.

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error. Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,]. Invalid locator paths include: empty or incomplete locator paths.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:
   - Double-click the object map that includes the object map item that you want to rename.
   - Right-click the object map that includes the object map item that you want to rename and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Navigate to the object map item that you want to rename.
   For example, you might need to expand a node to locate the item that you want to rename.

4. Click the object that you want to rename and then click the object again.
5. Type the item name that you want to use and then press `Enter`.

   If you use an invalid character, the item name displays in red.

   The new name displays in the **Item name** list.
6. Press **CTRL+S** to save your changes

If any existing scripts use the item name that you changed, you must manually change the scripts to use the new item name.

Note: All child nodes of any node in the object map tree are sorted alphabetically when you save the object map.

Modifying a Locator in an Object Map

Locators are automatically associated with an object map item when you record a script. However, you might want to modify a locator path to make it more generic. For example, if your test application automatically assigns the date or time to a specific control, you might want to modify the locator for that control to use a wildcard. Using a wildcard enables you to use the same locator for each test even though each test inserts a different date or time.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that includes the locator that you want to modify.
   - Right-click the object map that includes the locator that you want to modify and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Navigate to the locator that you want to modify.
   For example, you might need to expand a node to locate the locator that you want to modify.
4. Click the locator path that you want to modify and then click the locator path again.
5. If you have a valid locator path, you can type the item name and locator path that you want to use and then press `Enter`. To determine a valid locator path, use the **Locator Spy** dialog box as described in the following steps:
   a) Click **Silk4J** > **Record Locators**
   b) Position the mouse over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
   c) Select the locator that you want to use in the **Locator Details** table.
   d) Copy and paste the locator into the object map.
6. If necessary, modify the item name or locator text to meet your needs.

   If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

   Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

   Invalid locator paths include: empty or incomplete locator paths.
7. Press **CTRL+S** to save your changes

If any existing scripts use the locator path that you modified, you must manually change the visual tests or scripts to use the new locator path.

Updating Object Maps from the Test Application

If items in the test application change, you can use the **Object Map** UI to update the locators for these items.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that you want to use.
   - Right-click the object map that you want to use and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Click **Update Locator**. The **Locator Spy** displays and Silk4J opens the test application.
4. Position the mouse cursor over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
5. Select the locator that you want to use in the **Locator Details** table.
6. Remove any attributes that you do not want to use from the locator that is displayed in the **Locator** text field.
7. Click **Validate Locator** to validate that the locator works.
8. Click **Paste Locator to Editor** to update the locator in the object map.
9. Save the changed object map.

When you update an object map item from the AUT, you can change only the XPath representations of leaf nodes in the object map tree. You cannot change the XPath representations of any parent nodes. When the XPath representations of higher-level nodesin the object map tree are not consistent after the update, an error message displays.

Example

For example, suppose you have an object map item with an object map ID that has the following three hierarchy levels:

```
WebBrowser.Dialog.Cancel
```

The corresponding XPath representation of these hierarchy levels is the following:

```
 /BrowserApplication//Dialog//PushButton[@caption='Cancel']
```

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//Dialog`
- Third hierarchy level: `//PushButton[@caption='Cancel']`

You can use the following locator to update the object map item:

```
 /BrowserApplication//Dialog//PushButton[@id='123']
```

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//Dialog`
- Third hierarchy level: `//PushButton[@id='123']`

You cannot use the following locator cannot to update the object map item, because the second level hierarchy nodes do not match:

```
 /BrowserApplication//BrowserWindow//PushButton[@id='9999999']
```

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//BrowserWindow`

- Third hierarchy level: `//PushButton[@id='9999999']`

Copying an Object Map Item

You can copy and paste object map entries within or between object maps. For example, if the same functionality exists in two separate test applications, you might copy a portion of one object map into another object map.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that includes the object map item that you want to copy.
   - Right-click the object map that includes the object map item that you want to copy and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Navigate to the object map item that you want to copy.
   For example, you might need to expand a node to locate the item that you want to copy.
4. Choose one of the following:

   - Right-click the object map item that you want to copy and choose **Copy tree**.
   - Click the object map item that you want to copy and then press `Ctrl+C`.
5. In the object map hierarchy, navigate to the position where you want to paste the item that you copied.

   For instance, to include an item on the first level of the hierarchy, click the first item name in the item list. To position the copied item a level below a specific item, click the item that you want to position the copied item below.

   To copy and paste between object maps, you must exit the map where you copied the object map item and open and edit the object map where you want to paste the object map item.
6. Choose one of the following:

   - Right-click the position in the object map where you want to paste the copied object map item and choose **Paste**.
   - Click the position in the object map where you want to paste the copied object map item and then press `Ctrl+V`.

   The object map item displays in its new position in the hierarchy.
7. Press **CTRL+S** to save your changes

If any existing scripts use the object map item name that you moved, you must manually change the scripts to use the new position in the hierarchy.

Adding an Object Map Item

Object map items are automatically created when you record a script. Occasionally, you might want to manually add an object map item.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that includes the object map item that you want to rename.
   - Right-click the object map that includes the object map item that you want to rename and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. In the object map hierarchy, navigate to the position where you want to add the object map item.

   For instance, to include an item on the first level of the hierarchy, click the first item name in the item list. To position the new item a level below a specific item, click the item that you want to position the copied item below.
4. Click **Insert new**. A new item is added to the hierarchy.
5. If you have a valid locator path, you can type the item name and locator path that you want to use and then press `Enter`. To determine a valid locator path, use the **Locator Spy** dialog box as described in the following steps:
   a) Click **Silk4J** > **Record Locators**
   b) Position the mouse over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
   c) Select the locator that you want to use in the **Locator Details** table.
   d) Copy and paste the locator into the object map.
6. If necessary, modify the item name or locator text to meet your needs.

   If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

   Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

   Invalid locator paths include: empty or incomplete locator paths.
7. Press **CTRL+S** to save your changes

Note: All child nodes of any node in the object map tree are sorted alphabetically when you save the object map.

Opening an Object Map from a Script

When you are editing a script, you can open an object map by right clicking on an object map entry in the script and selecting **Open Silk4JAsset.** This will open the object map in the GUI.

Use `Ctrl+Click` and click on an object map entry and the object map entry will turn into a hyperlink. Click it to open it.

Example

```
@Test
public void test() {
        Window mainWindow = desktop.<Window>find("Untitled -
Notepad");
     mainWindow.<TextField>find("TextField").typeKeys("hello");
}
```

In the previous code sample, right-click `Untitled - Notepad` to open the entry `Untitled - Notepad` in the object map, or right-click `TextField` to open the entry `Untitled - Notepad.TextField` in the object map.

Highlighting an Object Map Item in the Test Application

After you add or record an object map item, you can click **Highlight** to highlight the item in the test application. You might want to highlight an item to confirm that it's the item that you want to modify in the object map.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that you want to use.
   - Right-click the object map that you want to use and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. In the object map hierarchy, select the object map item that you want to highlight in the test application.

   Note: Ensure that only one instance of the test application is running. Running multiple instances of the test application will cause an error because multiple objects will match the locator.
4. Click **Highlight**.

   The **Select Application** dialog box might open if the test application has not been associated with the object map. If this happens, select the application that you want to test and then click **OK**.

   Silk4J opens the test application and displays a green box around the control that the object map item represents.

Navigating from a Locator to an Object Map Entry in a Script

If you want to see more than the **ID** of an object map entry, you can easily see the raw locator that will be used by the Open Agent when the command is executed by doing the following:

1. Open a script.
2. Place your cursor within a string in a line of the script that you want to identify.
3. Right click and select **Open Silk4J Asset**.

   Note:

   If the cursor is in a string that does not represent an object map entry, Silk4J will still assume that it is an object map entry and you may not get the results that you expect.

   The **Object Map** window opens with the proper item selected in the tree view.

Finding Errors in an Object Map

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error. Use the toolbar in the **Object Map** window to navigate to any errors.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that you want to troubleshoot.
   - Right-click the object map that you want to troubleshoot and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Look for any item name or locator text displayed in red.
4. If necessary, modify the item name or locator text to meet your needs.

   If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

   Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

   Invalid locator paths include: empty or incomplete locator paths.

5. Press **CTRL+S** to save your changes

Deleting an Object Map Item

You might want to delete an item from an object map if it no longer exists in the test application or for some other reason.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

   - Double-click the object map that includes the object map item that you want to delete.
   - Right-click the object map that includes the object map item that you want to delete and choose **Open**.

   The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Navigate to the object map item that you want to delete.
   For example, you might need to expand a node to locate the object map item that you want to delete.
4. Choose one of the following:

   - Right-click the object map item that you want to delete and choose **Delete**, or choose **Delete tree** to additionally delete all child items of the object map item.
   - Click the object map item that you want to delete and then press **DEL**, or press **CTRL+DEL** to additionally delete all child items of the object map item.
5. Press **CTRL+S** to save your changes

If any existing scripts use the object map item or its children that you deleted, you must manually change any references to that object map item in the scripts.

Best Practices with Object Maps

As a best practice, we recommend reviewing all object map items before you record your tests. For example, you might create a test that clicks every object and opens every window and dialog box in your test application. Then, you can review what each object is named and make any necessary modifications before you record your functional tests. You can delete the test that you created for the object map items after you review and modify the object item names.

You can navigate the tree in an object map by using the arrow keys.

Image Recognition Support

You can use image recognition in the following situations:

- To conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *image clicks* instead of coordinate-based clicks to click on a specified image.
- To test graphical objects in the application under test, for example charts.
- To perform a check of the visible UI of the application under test.

If you want to click on a control that is otherwise not recognizable, you can use the `imageClick` method with an image asset. If you want to verify that an otherwise not recognizable control exists in your application under test, you can use the `verifyAsset` method with an image verification.

Image recognition methods are supported for all technology domains that are supported by Silk4J.

Image Click Recording

Image click recording is disabled by default in favor of coordinate-based click recording, because image click recording might generate a confusingly large number of images. To enable image click recording, you can perform one of the following:

- In the **Recording** dialog box, check **Record image clicks**.
- Click **Silk4J** > **Edit Options**, select the **Recording** tab, and check the check box in the **Record image clicks** section.

When image click recording is enabled, Silk4J records `ImageClick` methods when object recognition or text recognition is not possible. You can insert image clicks in your script for any control, even if the image clicks are not recorded.

If you do not whish to record an `ImageClick` action, you can turn off image click recording and record normal clicks or text clicks.

Note: The recorded images are not reused. Silk4J creates a new image asset for each image click that you record.

Note: Image click recording is not supported for applications or applets that use the Java AWT/Swing controls.

Image Recognition Methods

Silk4J provides the following methods for image recognition:

Method	Description
`imageClick`	Clicks in the middle of the image that is specified in an asset. Waits until the image is found or the *Object resolve timeout*, which you can define in the synchronization options, is over.
`imageExists`	Returns whether the image that is specified in an asset exists.
`imageRectangle`	Returns the object-relative rectangle of the image that is specified in an asset.
`imageClickFile`	Clicks on the image that is specified in a file.

Method	Description
`imageExistsFile`	Returns whether the image that is specified in a file exists.
`imageRectangleFile`	Returns the object-relative rectangle of the image that is specified in a file.
`verifyAsset`	Executes a verification asset. Throws a VerificationFailedException if the verification does not pass.
`tryVerifyAsset`	Executes a verification asset and returns whether the verification passed.

Image Assets

You can use image assets in the following situations:

- To conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *image clicks* instead of coordinate-based clicks to click on a specified image.
- To test graphical objects in the application under test, for example charts.

Image assets consist of an image with some additional information that is required by Silk4J to work with the asset.

Silk4J provides the following methods for image assets:

Method	Description
`imageClick`	Clicks in the middle of the specified image asset. Waits until the image is found or the *Object resolve timeout*, which you can define in the synchronization options, is over.
`imageExists`	Returns whether the specified image asset exists.
`imageRectangle`	Returns the object-relative rectangle of the specified image asset.

Image assets must be located in the `Image Assets` folder of the project. The `.imageasset` files must be embedded resources.

Creating an Image Asset

You can create image assets in one of the following ways:

- By inserting a new image asset into an existing script.
- During recording.
- From the menu.

To create a new image asset from the menu, perform the following steps:

1. In the menu, click **Silk4J** > **New Image Asset**.
2. Select the project, to which you want to add the new image asset, and type a meaningful name for the asset into the **Name** field.
3. Click **Finish**. The image asset UI opens.
4. Select how you want to add an image to the asset.
   - If you want to use an existing image, click **Browse** and select the image file.
   - If you want to capture a new image from the UI of the application under test, click **Capture**.
5. If you have selected to capture a new image, select the area of the screen that you want to capture and click **Capture Selection**.

6. *Optional:* You can set the option **Client Area Only** to define that only the part of the image that is actually part of the AUT is considered when Silk4J compares the image verification to the UI of the AUT.
7. Specify the **Accuracy Level**.

   The accuracy level defines how much the image to be clicked is allowed to be different to the image in the application under test, before Silk4J declares the images as different. This is helpful if you are testing multiple systems or browsers with different screen resolutions. We recommend to choose a high level of accuracy in order to prevent false positives. The default value for the accuracy level is 6. You can change the default accuracy level in the options.

   Note: When you set the **Accuracy Level** to less than five, the actual colors of the images are no longer considered for the comparison. Only the grayscale representations of the images are compared.
8. Save the image asset.

The new image asset is listed under the current project in the **Package Explorer**, and you can use it to perform image clicks.

You can add multiple images to the same image asset.

Adding Multiple Images to the Same Image Asset

During testing, you will often need to test functionality on multiple environments and with different testing configurations. In a different environment, the actual image might differ in such a degree from the image that you have captured in the image asset, that image clicks might fail, although the image is existing. In such a case, you can add multiple images to the same image asset.

To add an additional image to an image asset:

1. Double-click on the image asset to which you want to add an additional image. The image asset UI opens.
2. Click on the plus sign in the lower part of the UI to add a new image to the image asset.
3. Save the image asset.

The new image is added to the asset. Each time an image click is called, and until a match is achieved, Silk4J will compare the images in the asset with the images in the UI of the application under test. By default, Silk4J compares the images in the order in which they have been added to the asset.

Note: To change the order in which Silk4J compares the images, click on an image in the lower part of the image asset UI and drag the image to the position that you want. The order lowers from left to right. The image that is compared first is the image in the left-most position.

Opening an Asset from a Script

When you are editing a script, you can open an asset by right clicking it and selecting **Open Silk4JAsset.** This will open the asset in the GUI.

If the asset is a reference to a file on the system, for example, referenced by `ImageClickFile`, the file will be opened by your system's default editor.

Use `Ctrl+Click` and click on an asset and the asset will turn into a hyperlink. Click it to open it.

Image Verifications

You can use an *Image Verification* to check if an image exists in the UI of the application under test (AUT) or not.

Image verifications consist of an image with some additional information that is required by Silk4J to work with the asset.

To execute an image verification, use the `verifyAsset` method.

Image verification assets must be located in the `Verifications` folder of the project.
The `.verification` files must be embedded resources.

An image verification fails when Silk4J cannot find the image in the AUT. In this case the script breaks execution and throws a VerificationFailedException. To avoid this behavior, use the `tryVerifyAsset` method.

If the locator for the image verification is not found in the AUT, Silk4J throws an `ObjectNotFoundException`.

You can open a successful image verification in TrueLog Explorer by clicking **Open Verification** in the **Info** tab of the verification step. You can open a failed image verification in TrueLog Explorer by clicking **Show Differences** in the **Info** tab of the verification step. If a failed image verification would have been successful if a lower accuracy level had been used, the accuracy level that would have succeeded is suggested.

Creating an Image Verification

You can create image verifications in one of the following ways:

- By using the menu.
- During recording.

To create a new image verification in the menu, perform the following steps:

1. Click **Silk4J** > **New Image Verification**.
2. Select the project, to which you want to add the new image verification, and type a meaningful name for the verification into the **Name** field.
3. Click **Finish**. The image verification UI opens.
4. Click **Identify** to identify the image that you want to verify in the application under test.
5. *Optional:* If you want to recapture the same image from the application under test, because there is a change in comparison to the image that you had initially captured, click **Recapture**.
6. *Optional:* You can click **Verify** to test if the image verification works.
7. *Optional:* You can add an exclusion area to the image verification, which will not be considered when Silk4J compares the image verification to the UI of the application under test (AUT).
8. *Optional:* You can set the option **Client Area Only** to define that only the part of the image that is actually part of the AUT is considered when Silk4J compares the image verification to the UI of the AUT.
9. Specify the **Accuracy Level**.

   The accuracy level defines how much the image to be verified is allowed to be different to the image in the application under test, before Silk4J declares the images as different. This is helpful if you are testing multiple systems or browsers with different screen resolutions. We recommend to choose a high level of accuracy in order to prevent false positives. You can change the default accuracy level in the options.

   Note: When you set the **Accuracy Level** to less than five, the actual colors of the images are no longer considered for the comparison. Only the grayscale representations of the images are compared.
10. Save the image verification.

The new image verification is listed in the **Package Explorer**, and you can use it to check if the image exists in the UI of your application under test.

Adding an Image Verification During Recording

You can add image verifications to your scripts to check if controls which are otherwise not recognizable exist in the UI of the application under test. To add an image verification during the recording of a script, perform the following steps:

1. Begin recording.
2. Move the mouse cursor over the image that you want to verify and click **Ctrl + Alt**. Silk4J asks you if you want to verify a property or an image.
3. Select **Create or Insert an Image Verification**.
4. Perform one of the following steps:
    - To create a new image verification in the image verification UI, select **New** from the list box.
    - To insert an existing image verification asset, select the image verification asset from the list box.
5. Click **OK**.
    - If you have chosen to create a new image verification, the image verification UI opens.
    - If you have chosen to use an existing image verification, the image verification is added to your script. You can skip the remaining steps in this topic.
6. To create a new image verification, click **Verify** in the image verification UI.
7. Move the mouse cursor over the image in the AUT and click **CTRL+ALT.** The image verification UI displays the new image verification.
8. Click **OK**. The new image verification is added to the current project.
9. Continue recording.

Using Assets in Multiple Projects

In Silk4J, image assets, image verifications, and object maps are referred to as *assets*. If you want to use assets outside of the scope of the project in which they are located, you need to add a direct project dependency from the project in which you want to use the assets to the project in which the assets are located. If you are playing back test using the Java automation, the scope of the assets automatically includes all projects in the classpath.

During replay, when an asset is used, Silk4J firstly searches in the current project for the asset. If Silk4J does not find the asset in the current project, Silk4J additionally searches the projects to which the current project has a project dependency.If the asset is still not found, Silk4J throws an error. If you are playing back test using the Java automation, Silk4J firstly searches in the current project for the asset and then searches all projects in the classpath.

Silk4J persists the order of any found assets.

If assets with the same name exist in more than one project, and you do not want to use the asset that is included in the current project, you can define which specific asset you want to use in any method that uses the asset. To define which asset you want to use, add the asset namespace as a prefix to the asset name when calling the method. The asset namespace defaults to the project name.

Note: When you start working with Silk4J, the asset namespace option is added to the `silk4j.settings` file of every Silk4J project in your workspace that has been created with a previous version of Silk4J.

Example: Adding a project dependency

If the project *ProjectA* contains a test that calls the following code:

```
imageClick("imageAsset");
```

and the image asset *imageAsset* is located in project *ProjectB*, you need to add a direct project dependency from *ProjectA* to *ProjectB*.

Example: Calling a specific asset

If *ProjectA* and *ProjectB* both contain an image asset with the name *anotherImageAsset*, and you explicitly want to click the image asset from *ProjectB*, use the following code:

```
imageClick("ProjectB:anotherImageAsset")
```

Supported Attribute Types

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. If necessary, you can change the attribute type in one of the following ways:

- Manually typing another attribute type and value.
- Specifying another preference for the default attribute type by changing the **Preferred attribute list** values.

Attributes for Adobe Flex Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Flex applications include:

- automationName
- caption (similar to automationName)
- automationClassName (e.g. `FlexButton`)
- className (the full qualified name of the implementation class, e.g. `mx.controls.Button`)
- automationIndex (the index of the control in the view of the FlexAutomation, e.g. `index:1`)
- index (similar to automationIndex but without the prefix, e.g. `1`)
- id (the id of the control)
- windowId (similar to id)
- label (the label of the control)
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Attributes for Java AWT/Swing Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java AWT/Swing include:

- caption
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text input field, the caption of the closest label at the left side or above the control is used.
- name
- accessibleName
- *Swing only*: All custom object definition attributes set in the widget with `SetClientProperty("propertyName", "propertyValue")`

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Attributes for Java SWT Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Attributes for MSUIA Applications

Note: MSUIA is deprecated. For new tests, use the WPF technology domain.

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for MSUIA applications include:

- acceleratorkey
- accesskey
- automationid
- classname
- controltype
- frameworkid
- haskeyboardfocus
- helptext
- isenabled
- isoffscreen
- ispassword
- caption
- name
- nativewindowhandle
- orientation

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Locator Attributes for Identifying Rumba Controls

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. Supported attributes include:

caption	The text that the control displays.

priorlabel	Since input fields on a form normally have a label explaining the purpose of the input, the intention of **priorlabel** is to identify the text input field, **RumbaTextField**, by the text of its adjacent label field, **RumbaLabel**. If no preceding label is found in the same line of the text field, or if the label at the right side is closer to the text field than the left one, a label on the right side of the text field is used.
StartRow	This attribute is not recorded, but you can manually add it to the locator. Use **StartRow** to identify the text input field, **RumbaTextField**, that starts at this row.
StartColumn	This attribute is not recorded, but you can manually add it to the locator. Use **StartColumn** to identify the text input field, **RumbaTextField**, that starts at this column.
All dynamic locator attributes.	For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Attributes for SAP Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for SAP include:

- automationId
- caption

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

- *automationId*
- *caption*
- *className*
- *name*
- All dynamic locator attributes

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

To identify components within Silverlight scripts, you can specify the *automationId*, *caption*, *className*, *name* or any dynamic locator attribute. The *automationId* can be set by the application developer. For example, a locator with an *automationId* might look like `//SLButton[@automationId="okButton"]`.

We recommend using the *automationId* because it is typically the most useful and stable attribute.

Attribute Type	Description	Example
automationId	An identifier that is provided by the developer of the application under test. The Visual Studio designer automatically assigns an *automationId* to every control that is created with the designer. The application developer uses this ID to identify the control in the application code.	`//SLButton[@automationId="okButton"]`
caption	The text that the control displays. When testing a localized application in multiple languages, use the *automationId* or *name* attribute instead of the *caption*.	`//SLButton[@caption="Ok"]`
className	The simple .NET class name (without namespace) of the Silverlight control. Using the *className* attribute can help to identify a custom control that is derived from a standard Silverlight control that Silk4J recognizes.	`//SLButton[@className='MyCustomButton']`
name	The name of a control. Can be provided by the developer of the application under test.	`//SLButton[@name="okButton"]`

⚠ **Attention:** The *name* attribute in XAML code maps to the locator attribute *automationId*, not to the locator attribute *name*.

During recording, Silk4J creates a locator for a Silverlight control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has an *automationId* and a *name*, Silk4J uses the *automationId*, if it is unique, when creating the locator.

The following table shows how an application developer can define a Silverlight button with the text "Ok" in the XAML code of the application:

XAML Code for the Object	Locator to Find the Object from Silk Test
`<Button>Ok</Button>`	`//SLButton[@caption="Ok"]`
`<Button Name="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`
`<Button AutomationProperties.AutomationId="okButton">Ok</Button>`	`//SLButton[@automationId="okButton"]`
`<Button AutomationProperties.Name="okButton">Ok</Button>`	`//SLButton[@name="okButton"]`

Attributes for Web Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Web applications include:

- caption (supports wildcards ? and *)
- all DOM attributes (supports wildcards ? and *)

✎ **Note:** Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

```
<a>abc
abc</a>
```

Uses the following locator:

```
//A[@innerText='abc abc']
```

Attributes for Windows API-based Client/Server Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows API-based client/server applications include:

- caption
- windowid
- priorlabel: Helps to identify text input fields by the text of its adjacent label field. Every input field of a form usually has a label that explains the purpose of the input. For controls that do not have a caption, the attribute **priorlabel** is automatically used in the locator. For the **priorlabel** value of a control, for example a text box, the caption of the closest label at the left side or above the control is used.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Attributes for Windows Forms Applications

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests.

Supported attributes for Windows Forms applications include:

- automationid
- caption
- windowid
- priorlabel (For controls that do not have a caption, the priorlabel is used as the caption automatically. For controls with a caption, it may be easier to use the caption.)

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Attributes for Windows Presentation Foundation (WPF) Applications

Supported attributes for WPF applications include:

- *automationId*
- *caption*
- *className*

- *name*
- All dynamic locator attributes.

Note: Attribute names are case sensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and `*`.

For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Object Recognition

To identify components within WPF scripts, you can specify the *automationId*, *caption*, *className*, or *name*. The name that is given to an element in the application is used as the *automationId* attribute for the locator if available. As a result, most objects can be uniquely identified using only this attribute. For example, a locator with an *automationId* might look like: `//WPFButton[@automationId='okButton']"`.

If you define an *automationId* and any other attribute, only the *automationId* is used during replay. If there is no *automationId* defined, the *name* is used to resolve the component. If neither a *name* nor an *automationId* are defined, the *caption* value is used. If no caption is defined, the *className* is used. We recommend using the *automationId* because it is the most useful property.

Attribute Type	Description	Example
automationId	An ID that was provided by the developer of the test application.	`//WPFButton[@automationId='okButton']"`
name	The name of a control. The Visual Studio designer automatically assigns a name to every control that is created with the designer. The application developer uses this name to identify the control in the application code.	`//WPFButton[@name='okButton']"`
caption	The text that the control displays. When testing a localized application in multiple languages, use the automationId or name attribute instead of the caption.	`//WPFButton[@automationId='Ok']"`
className	The simple .NET class name (without namespace) of the WPF control. Using the class name attribute can help to identify a custom control that is derived from a standard WPF control that Silk4J recognizes.	`//WPFButton[@className='MyCustomButton']"`

During recording, Silk4J creates a locator for a WPF control by using the *automationId*, *name*, *caption*, or *className* attributes in the order that they are listed in the preceding table. For example, if a control has a *automationId* and a *name*, Silk4J uses the *automationId* when creating the locator.

The following example shows how an application developer can define a *name* and an *automationId* for a WPF button in the XAML code of the application:

```
<Button Name="okButton" AutomationProperties.AutomationId="okButton"
Click="okButton_Click">Ok</Button>
```

Dynamic Locator Attributes

In a locator for identifying a control during replay you can use a pre-defined set of locator attributes, for example *caption* and *automationId*, which depend on the technology domain. But you can also use every property, including dynamic properties, of a control as locator attribute. A list of available properties for a certain control can be retrieved with the `GetPropertyList` method. All returned properties can be used for identifying a control with a locator.

Note: You can use the `GetProperty` method to retrieve the actual value for a certain property of interest. You can then use this value in your locator.

Example

If you want to identify the button that has the user input focus in a Silverlight application, you can type:

```
browser.Find("//SLButton[@IsKeyboardFocused=true]")
```

or alternatively

```
Dim button = dialog.SLButton("@IsKeyboardFocused=true")
```

This works because Silk4J exposes a property called `IsDefault` for the Silverlight button control.

Example

If you want to identify a button in a Silverlight application with the font size 12 you can type:

```
Dim button = browser.Find("//SLButton[@FontSize=12]")
```

or alternatively

```
Dim button = browser.SLButton("@FontSize=12")
```

This works because the underlying control in the application under test, in this case the Silverlight button, has a property called `FontSize`.

Enhancing Tests

This section describes how you can enhance a test.

Calling Windows DLLs

This section describes how you can call DLLs. You can call a DLL either within the process of the Open Agent or in the application under test (AUT). This allows the reuse of existing native DLLs in test scripts.

DLL calls in the Open Agent are typically used to call global functions that do not interact with UI controls in the AUT.

DLL calls in the AUT are typically used to call functions that interact with UI controls of the application. This allows Silk4J to automatically synchronize the DLL call during playback.

Note: In 32-bit applications, you can call 32-bit DLLs, while in 64-bit applications you can call 64-bit DLLs. The Open Agent can execute both 32-bit and 64-bit DLLs.

Note: You can only call DLLs with a C interface. Calling of .NET assemblies, which also have the file extension .dll, is not supported.

Calling a Windows DLL from Within a Script

All classes and annotations that are related to DLL calling are located in the package `com.borland.silktest.jtf.dll`.

A declaration for a DLL starts with an interface that has a Dll attribute. The syntax of the declaration is the following:

```
@Dll("dllname.dll")
public interface DllInterfaceName {
  FunctionDeclaration
  [FunctionDeclaration]…
}
```

dllname	The name of or the full path to the DLL file that contains the functions you want to call from your Java scripts. Environment variables in the DLL path are automatically resolved. You do not have to use double backslashes (\\) in the path, single backslashes (\) are sufficient.
DllInterfaceName	The identifier that is used to interact with the DLL in a script.
FunctionDeclaration	A function declaration of a DLL function you want to call.

DLL Function Declaration Syntax

A function declaration for a DLL typically has the following form:

```
return-type function-name( [arg-list] )
```

For functions that do not have a return value, the declaration has the following form:

```
void function-name( [arg-list] )
```

return-type	The data type of the return value.
function-name	The name of the function.

arg-list
A list of the arguments that are passed to the function.

The list is specified as follows:

```
data-type identifier
```

data-type
The data type of the argument.

- To specify arguments that can be modified by a function or passed out from a function, use the `InOutArgument` and the `OutArgument` class.
- If you want the DLL function to set the value of the argument, use the `OutArgument` class.
- If you want to pass a value into the function, and have the function change the value and pass the new value out, use the `InOutArgument` class.

identifier
The name of the argument.

DLL Calling Example

This example writes the text *hello world!* into a field by calling the `SendMessage` DLL function from `user32.dll`.

DLL Declaration:

```
@Dll("user32.dll")
public interface IUserDll32Functions {
  int SendMessageW(TestObject obj, int message, int wParam, Object lParam);
}
```

The following code shows how to call the declared DLL function in the AUT:

```
IUserDll32Functions user32Function =
DllCall.createInProcessDllCall(IUserDll32Functions.class, desktop);
TextField textField = desktop.find("//TextField");
user32Function.SendMessageW(textField, WindowsMessages.WM_SETTEXT, 0, "my
text");
```

Note: You can only call DLL functions in the AUT if the first parameter of the DLL function has the C data type HWND.

The following code shows how to call the declared DLL functions in the process of the Open Agent:

```
IUserDll32Functions user32Function =
DllCall.createAgentDllCall(IUserDll32Functions.class, desktop);
TextField textField = desktop.find("//TextField");
user32Function.SendMessageW(textField, WindowsMessages.WM_SETTEXT, 0, "my
text");
```

Note: The example code uses the `WindowsMessages` class that contains useful constants for usage with DLL functions that relate to Windows messaging.

Passing Arguments to DLL Functions

DLL functions are written in C, so the arguments that you pass to these functions must have the appropriate C data types. The following data types are supported:

int
Use this data type for arguments or return values with the following data types:

- int
- INT
- long
- LONG
- DWORD
- BOOL
- WPARAM
- HWND

The Java type int works for all DLL arguments that have a 4-byte value.

long
Use this data type for arguments or return values with the C data types long and int64. The Java type long works for all DLL arguments that have an 8-byte value.

short
Use this data type for arguments or return values with the C data types short and WORD. The Java type short works for all DLL arguments that have a 2-byte value.

boolean
Use this data type for arguments or return values with the C data type bool.

String
Use this for arguments or return values that are Strings in C.

double
Use this for arguments or return values with the C data type double.

com.borland.silktest.jtf.Rect
Use this for arguments with the C data type RECT. Rect cannot be used as a return value.

com.borland.silktest.jtf.Point
Use this for arguments with the C data type POINT. Point cannot be used as a return value.

com.borland.silktest.jtf.TestObject
Use this for arguments with the C data type HWND. TestObject cannot be used as a return value, however you can declare DLL functions that return a HWND with an Integer as the return type.

Note: The passed TestObject must implement the com.borland.silktest.jtf.INativeWindow interface so that Silk4J is able to determine the window handle for the TestObject that should be passed into the DLL function. Otherwise an exception is thrown when calling the DLL function.

List
Use this for arrays for user defined C structs. Lists cannot be used as a return value.

Note: When you use a List as an in/out parameter, the list that is passed in must be large enough to hold the returned contents.

Note: A C struct can be represented by a List, where every list element corresponds to a struct member. The first struct member is represented by the first element in the list, the second struct members is represented by the second element in the list, and so on.

Note: Any argument that you pass to a DLL function must have one of the preceding Java data types.

Passing Arguments that Can Be Modified by the DLL Function

An argument whose value will be modified by a DLL function needs to be passed either by using an InOutArgument, if the value can be changed, or by using an OutArgument.

Example

This example uses the `GetCursorPos` function of the `user32.dll` in order to retrieve the current cursor position.

DLL declaration:

```
@Dll( "user32.dll" )
public interface IUserDll32Functions {
  int GetCursorPos( OutArgument<Point> point);
}
```

Usage:

```
IUserDll32Functions user32Function =
DllCall.createAgentDllCall(IUserDll32Functions.class, desktop);

OutArgument<Point> point = new OutArgument<Point>(Point.class);
user32Function.GetCursorPos(point);

System.out.println("cursor position = " + point.getValue());
```

Passing String Arguments to DLL Functions

Strings that are passing into a DLL function or that are returned by a DLL function are treated by default as Unicode Strings. If your DLL function requires ANSI String arguments, use the `CharacterSet` property of the `DllFunctionOptions` attribute.

Example

```
@Dll( "user32.dll" )
public interface IUserDll32Functions {
  @FunctionOptions(characterSet=DllCharacterSet.Ansi)
  int SendMessageA(TestObject obj, int message, int wParam,
Object lParam);
}
```

Passing a String back from a DLL call as an OutArgument works per default if the String's size does not exceed 256 characters length. If the String that should be passed back is longer than 256 characters, you need to pass an InOurArgument with a String in that is long enough to hold the resulting String.

Example

Use the following code to create a String with 1024 blank characters:

```
char[] charArray = new char[1024];
Arrays.fill(charArray,' ');
String longEmptyString = new String(charArray);
```

Pass this InOutArgument as an argument into a DLL function and the DLL function will pass back Strings of up to 1024 characters of length.

When passing a String back from a DLL call as a function return value, the DLL should implement a DLL function called `FreeDllMemory` that accepts the C String pointer returned by the DLL function and that frees the previously allocated memory. If no such function exists the memory will be leaked.

Aliasing a DLL Name

If a DLL function has the same name as a reserved word in Java, or the function does not have a name but an ordinal number, you need to rename the function within your declaration and use the alias statement to map the declared name to the actual name.

Example

For example, the `goto` statement is reserved by the Java compiler. Therefore, to call a function named `goto`, you need to declare it with another name, and add an alias statement, as shown here:

```
@Dll("mydll.dll")
public interface IMyDllFunctions {
  @FunctionOptions(alias="break")
  void MyBreak();
}
```

Conventions for Calling DLL Functions

The following calling conventions are supported when calling DLL functions:

- __stdcall
- __cdecl

The *__stdcall* calling convention is used by default when calling DLL functions. This calling convention is used by all Windows API DLL functions.

You can change the calling convention for a DLL function by using the `CallingConvention` property of the `DllFunctionOptions` annotation.

Example

The following code example declares a DLL function with the *__decl* calling convention:

```
@Dll("msvcrt.dll")
public interface IMsVisualCRuntime {
  @FunctionOptions(callingConvention=CallingConvention.Cdecl)
  double cos(double inputInRadians);
}
```

Custom Controls

Silk4J provides the following features to support you when you are working with custom controls:

- The *dynamic invoke* functionality of Silk4J enables you to directly call methods, retrieve properties, or set properties on an actual instance of a control in the application under test (AUT).
- For Win32-based applications, the *class mapping* functionality enables you to map the name of a custom control class to the name of a standard Silk Test class. You can then use the functionality that is supported for the standard Silk Test class in your test.

- The **Manage Custom Controls** dialog box enables you to specify a name for a custom control that can be used in a locator and also enables you to write reusable code for the interaction with the custom control.

Note: For custom controls, you can only record methods like `click`,`textClick`, and `typeKeys` with Silk4J. You cannot record custom methods for custom controls except when you are testing Adobe Flex applications.

Dynamic Invoke

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

Call dynamic methods on objects with the `invoke` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Call multiple dynamic methods on objects with the `invokeMethods` method. To retrieve a list of supported dynamic methods for a control, use the `getDynamicMethodList` method.

Retrieve dynamic properties with the `getProperty` method and set dynamic properties with the `setProperty` method. To retrieve a list of supported dynamic properties for a control, use the `getPropertyList` method.

For example, to call a method named `SetTitle`, which requires the title to be set as an input parameter of type string, on an actual instance of a control in the application under test, type the following:

```
control.invoke("SetTitle","my new title");
```

Note: Typically, most properties are read-only and cannot be set.

Note: Reflection is used in most technology domains to call methods and retrieve properties.

Note: You cannot dynamically invoke methods for DOM elements.

Frequently Asked Questions About Dynamic Invoke

This section includes a collection of questions that you might encounter when you are dynamically invoking methods to test custom controls.

Which Methods Can I Call With the invoke Method?

To get a list of all the methods that you can call with the `invoke` method for a specific test object, you can use the `getDynamicMethodList`. To view the list, you can for example print it to the console or view it in the debugger.

Why Does an Invoke Call Return a Simple String when the Expected Return is a Complex Object?

The `invoke` method can only return simple data types. Complex types are returned as string. Silk4J uses the `ToString` method to retrieve the string representation of the return value. To call the individual methods and read properties of the complex object that is returned by the first method invocation, use `invokeMethods` instead of `invoke`.

How Can I Simplify My Scripts When I Use Many Calls To invokeMethods?

When you extensively use `invokeMethods` in your scripts, the scripts might become complex because you have to pass all method names as strings and all parameters as lists. To simplify such complex scripts, create a static method that interacts with the actual control in the AUT instead of interacting with the control through `invokeMethods`. For additional information, see *Adding Code to the Application Under Test to Test Custom Controls*.

Adding Code to the Application Under Test to Test Custom Controls

When you are testing Windows Forms applications or WPF applications, and you want to test complex custom controls or custom controls that you cannot test by simply using the `invoke` and `invokeMethods` methods, you can create a static method that interacts with the actual control in the application under test (AUT) and you can add this code to the AUT.

The benefit for you from adding code to the AUT is that the code in the AUT can use regular method calls for interacting with the control, instead of using the reflection-like style of calling methods with the dynamic invoke methods. Therefore you can use code completion and IntelliSense when you are writing you code. You can then call the code in the AUT with a simple invoke call, where you pass the control of interest as a parameter.

You can add code to the AUT in the following ways:

- Compile the code into the AUT. The implementation is simple, but you will be changing the AUT, which you might not want to do.
- Inject code to the AUT at runtime by using the `LoadAssembly` method in a test script. This requires more effort than compiling the code into the AUT, but the injected code will be located close to the test code. The `LoadAssembly` method is available for the classes `WPFWindow` and `FormsWindow`.

Example: Testing the UltraGrid Infragistics control

This example demonstrates how you can retrieve the content of an `UltraGrid` control. The `UltraGrid` control is included in the `NETAdvantage for Windows Forms` library which is provided by Infragistics.

To create the `UltraGridUtil` class, perform the following actions:

1. Open Microsoft Visual Studio and create a new class library project in C# or VB .NET. Call the new project `AUTExtensions`.

   Note: The class library should use the same .NET version as the AUT.

2. Add references to the required dependencies to the project. For example, for Infragistics version 12.2 you need to reference the following assemblies:

   - Infragistics4.Shared.v12.2
   - Infragistics4.Win.UltraWinGrid.v12.2
   - Infragistics4.Win.v12.2

   If you are not sure which version of Infragistics is used in your AUT you can use the **Process Explorer** tool from Microsoft to see which assemblies are loaded in your AUT.

   a. In the `AUTExtensions` project, create the new class UltraGridUtil with the following content:

```
' VB code
Public Class UltraGridUtil

  Public Shared Function GetContents(ultraGrid As
Infragistics.Win.UltraWinGrid.UltraGrid) As List(Of List(Of
String))
    Dim contents = New List(Of List(Of String))
    For Each row In ultraGrid.Rows
      Dim rowContents = New List(Of String)
      For Each cell In row.Cells
        rowContents.Add(cell.Text)
      Next
      contents.Add(rowContents)
    Next
    Return contents
  End Function

End Class
```

```
// C# code
using System.Collections.Generic;

namespace AUTExtensions {

  public class UltraGridUtil {

    public static List<List<string>>
GetContents(Infragistics.Win.UltraWinGrid.UltraGrid grid) {
      var result = new List<List<string>>();
      foreach (var row in grid.Rows) {
        var rowContent = new List<string>();
        foreach (var cell in row.Cells) {
          rowContent.Add(cell.Text);
        }
        result.Add(rowContent);
      }
      return result;
    }

  }

}
```

Note: The `Shared` modifier makes the `GetContents` method a static method.

3. Build the `AUTExtensions` project.
4. Load the assembly into the AUT during playback.

   - Open an existing test script or create a new test script.
   - Add code to the test script to load the assembly that you have built from the file system. For example:

     ```
     mainWindow.loadAssembly("C:/buildoutput/AUTExtensions.dll");
     ```
5. Call the static method of the injected code in order to get the contents of the `UltraGrid`:

   ```
   // Java code
   Control ultraGrid = mainWindow.find("//
   Control[@automationId='my grid']");
   List<List<String>> contents = (List<List<String>>)
   mainWindow.invoke("AUTExtensions.UltraGridUtil.GetContents",
   ultraGrid);
   ```

Frequently Asked Questions About Adding Code to the AUT

This section includes a collection of questions that you might encounter when you are adding code to the AUT to test custom controls.

Why is Code That I Have Injected Into the AUT With the LoadAssembly Method Not Updated in the AUT?

If code in the AUT is not replaced by code that you have injected with the `LoadAssembly` method into the AUT, the assembly might already be loaded in your AUT. Assemblies cannot be unloaded, so you have to close and re-start your AUT.

Why Do the Input Argument Types Not Match When I Invoke a Method?

If you invoke a method and you get an error that says that the input argument types do not match, the method that you want to invoke was found but the arguments are not correct. Make sure that you use the correct data types in your script.

If you use the `LoadAssembly` method in your script to load an assembly into the AUT, another reason for this error might be that your assembly is built against a different version of the third-party library than the version that is used by the AUT. To fix this problem, change the referenced assembly in your project. If you are not sure which version of the third-party library is used in your AUT, you can use the **Process Explorer** tool from Microsoft.

How Do I Fix the Compile Error when an Assembly Can Not Be Copied?

When you have tried to add code to the AUT with the `LoadAssembly` method, you might get the following compile error:
Could not copy '<assembly_name>.dll' to '<assembly_name>.dll'. The process cannot access the file.
The reason for this compile error is that the assembly is already loaded in the AUT and cannot be overwritten.

To fix this compile error, close the AUT and compile your script again.

Testing Adobe Flex Custom Controls

Silk4J supports testing Flex custom controls. By default, Silk4J provides record and playback support for the individual subcontrols of the custom control.

For testing custom controls, the following options exist:

- Basic support

  With basic support, you use dynamic invoke to interact with the custom control during replay. Use this low-effort approach when you want to access properties and methods of the custom control in the test application that Silk4J does not expose. The developer of the custom control can also add methods and properties to the custom control specifically for making the control easier to test. A user can then call those methods or properties using the dynamic invoke feature.

  The advantages of basic support include:

  - Dynamic invoke requires no code changes in the test application.
  - Using dynamic invoke is sufficient for most testing needs.

  The disadvantages of basic support include:

  - No specific class name is included in the locator (for example, Silk4J records “//FlexBox” rather than “//FlexSpinner”)
  - Only limited recording support
  - Silk4J cannot replay events.

  For more details about dynamic invoke, including an example, see *Dynamically Invoking Adobe Flex Methods*.
- Advanced support

With advanced support, you create specific automation support for the custom control. This additional automation support provides recording support and more powerful play-back support. The advantages of advanced support include:

- High-level recording and playback support, including the recording and replaying of events.
- Silk4J treats the custom control exactly the same as any other built-in Flex control.
- Seamless integration into Silk4J API
- Silk4J uses the specific class name in the locator (for example, Silk4J records “//FlexSpinner”)

The disadvantages of advanced support include:

- Implementation effort is required. The test application must be modified and the Open Agent must be extended.

Managing Custom Controls

You can create custom classes for custom controls for which Silk4J does not offer any dedicated support. Creating custom classes offers the following advantages:

- Better locators for scripts.
- An easy way to write reusable code for the interaction with the custom control.

Example: Testing the UltraGrid Infragistics control

Suppose that a custom grid control is recognized by Silk4J as the generic class `Control`. Using the custom control support of Silk4J has the following advantages:

Better object recognition because the custom control class name can be used in a locator.

Many objects might be recognized as `Control`. The locator requires an index to identify the specific object. For example, the object might be identified by the locator `//Control[13]`. When you create a custom class for this control, for example the class `UltraGrid`, you can use the locator `//UltraGrid`. By creating the custom class, you do not require the high index, which would be a fragile object identifier if the application under test changed.

You can implement reusable playback actions for the control in scripts.

When you are using custom classes, you can encapsulate the behavior for getting the contents of a grid into a method by adding the following code to your custom class, which is the class that gets generated when you specify the custom control in the user interface.

Typically, you can implement the methods in a custom control class in one of the following ways:

- You can use methods like `click`, `typeKeys`, `textClick`, and `textCapture`.
- You can dynamically invoke methods on the object in the AUT.
- You can dynamically invoke methods that you have added to the AUT. This is the approach that is described in this example.

You can use the following code to call the static method that is defined in the example in *Adding Code to the Application Under Test to Test*

Custom Controls. The method `GetContents` is added into the generated class `UltraGrid`.

```
// Java code
import
com.borland.silktest.jtf.Desktop;
import
com.borland.silktest.jtf.common.JtfO
bjectHandle;

public class UltraGrid extends
com.borland.silktest.jtf.Control {

  protected
UltraGrid(JtfObjectHandle handle,
Desktop desktop) {
    super(handle, desktop);
  }

  public List<List<String>>
getContents() {
    return (List<List<String>>)
invoke("AUTExtensions.UltraGridUtil.
GetContents", this);
  }
}
```

When you define a class as a custom control, you can use the class in the same way in which you can use any built-in class, for example the `Dialog` class.

```
// Java code
UltraGrid ultraGrid =
mainWindow.find("//
UltraGrid[@automationId='my
grid']");
List<List<String>> contents =
ultraGrid.getContents();
```

Supporting a Custom Control

To create a custom class for a custom control for which Silk4J does not offer any dedicated support.

1. Click **Silk4J** > **Manage Custom Controls**. The **Manage Custom Controls** dialog box opens.
2. In the **Silk4J Custom Controls Output Package** field, type in a name or click **Browse** to select the package that will contain the custom control.
3. Click on the tab of the technology domain for which you want to create a new custom class.
4. Click **Add**.
5. Click one of the following:
   - Click **Identify new custom control** to directly select a custom control in your application with the **Identify Object** dialog box.
   - Click **Add new custom control** to manually add a custom control to the list.

   A new row is added to the list of custom controls.
6. If you have chosen to manually add a custom control to the list:
   a) In the **Silk Test base class** column, select an existing base class from which your class will derive.

      This class should be the closest match to your type of custom control.

b) In the **Silk Test class** column, enter the name to use to refer to the class.

This is what will be seen in locators. For example: `//UltraGrid` instead of `//Control[13]`.

Note: After you add a valid class, it will become available in the **Silk Test base class** list. You can then reuse it as a base class.

c) In the **Custom control class name** column, enter the fully qualified class name of the class that is being mapped.

For example: `Infragistics.Win.UltraWinGrid.UltraGrid`. For Win32 applications, you can use the wildcards *?* and *** in the class name.

7. *Only for Win32 applications:* In the **Class mapping** column, set the class mapping to true to map the name of a custom control class to the name of a standard Silk Test class.

When you map the custom control class to the standard Silk Test class, you can use the functionality supported for the standard Silk Test class in your test.

8. Click **OK**.

9. *Only for scripts:*

a) Add custom methods and properties to your class for the custom control.

b) Use the custom methods and properties of your new class in your script.

Note: The custom methods and properties are not recorded.

Note: Do not rename the custom class or the base class in the script file. Changing the generated classes in the script might result in unexpected behavior. Use the script only to add properties and methods to your custom classes. Use the **Manage Custom Controls** dialog box to make any other changes to the custom classes.

Custom Controls Options

Silk4J > **Manage Custom Controls**.

In the **Silk4J Custom Controls Output Package**, define the package into which the new custom classes should be generated.

The following **Custom Controls** options are available:

Option	Description
Silk Test base class	Select an existing base class to use that your class will derive from. This class should be the closest match to your type of custom control.
Silk Test class	Enter the name to use to refer to the class. This is what will be seen in locators.
Custom control class name	Enter the fully qualified class name of the class that is being mapped. You can use the wildcards *?* and *** in the class name.
Class mapping	This option is available only for Win32 applications. Set the class mapping to true to map the name of a custom control class to the name of a standard Silk Test class. When you map the custom control class to the standard Silk Test class, you can use the functionality supported for the standard Silk Test class in your test.

Note: After you add a valid class, it will become available in the **Silk Test base class** list. You can then reuse it as a base class.

Example: Setting the options for the UltraGrid Infragistics control

To support the `UltraGrid` Infragistics control, use the following values:

Option	Value
Silk Test base class	`Control`

Option	Value
Silk Test class	`UltraGrid`
Custom control class name	`Infragistics.Win.UltraWinGrid.UltraGrid`

Improving Object Recognition with Microsoft Accessibility

You can use Microsoft Accessibility (Accessibility) to ease the recognition of objects at the class level. There are several objects in Internet Explorer and in Microsoft applications that Silk4J can better recognize if you enable Accessibility. For example, without enabling Accessibility Silk4J records only basic information about the menu bar in Microsoft Word and the tabs that appear. However, with Accessibility enabled, Silk4J fully recognizes those objects.

Example

Without using Accessibility, Silk4J cannot fully recognize a `DirectUIHwnd` control, because there is no public information about this control. Internet Explorer uses two `DirectUIHwnd` controls, one of which is a popup at the bottom of the browser window. This popup usually shows the following:

- The dialog box asking if you want to make Internet Explorer your default browser.
- The download options **Open**, **Save**, and **Cancel**.

When you start a project in Silk4J and record locators against the `DirectUIHwnd` popup, with accessibility disabled, you will see only a single control. If you enable Accessibility you will get full recognition of the `DirectUIHwnd` control.

Using Accessibility

Win32 uses the Accessibility support for controls that are recognized as generic controls. When Win32 locates a control, it tries to get the accessible object along with all accessible children of the control.

Objects returned by Accessibility are either of the class `AccessibleControl`, `Button` or `CheckBox`. `Button` and `Checkbox` are treated specifically because they support the normal set of methods and properties defined for those classes. For all generic objects returned by Accessibility the class is `AccessibleControl`.

Example

If an application has the following control hierarchy before Accessibility is enabled:

- Control
  - Control
- Button

When Accessibility is enabled, the hierarchy changes to the following:

- Control
  - Control
    - Accessible Control

        - Accessible Control
            - Button
- Button

Enabling Accessibility

If you are testing a Win32 application and Silk4J cannot recognize objects, you should first enable Accessibility. Accessibility is designed to enhance object recognition at the class level.

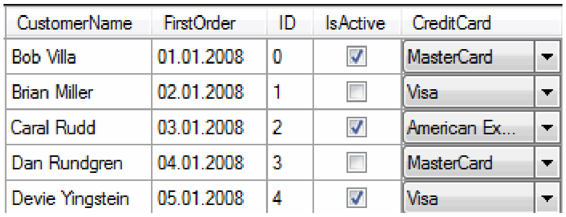
To enable Accessibility:

1. Click **Edit Options**. The **Script Options** dialog box opens.
2. Click **Advanced**.
3. Select the **Use Microsoft Accessibility** option. Accessibility is turned on.

Text Recognition Support

Text recognition methods enable you to conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *text clicks* instead of coordinate-based clicks to click on a specified text string within a control.

For example, you can simulate selecting the first cell in the second row of the following table:



CustomerName	FirstOrder	ID	IsActive	CreditCard
Bob Villa	01.01.2008	0	☑	MasterCard
Brian Miller	02.01.2008	1	□	Visa
Caral Rudd	03.01.2008	2	☑	American Ex...
Dan Rundgren	04.01.2008	3	□	MasterCard
Devie Yingstein	05.01.2008	4	☑	Visa

Specifying the text of the cell results in the following code line:

```
table.textClick("Brian Miller");
```

Text recognition methods are supported for the following technology domains:

- Win32.
- WPF.
- Windows Forms.
- Java SWT and Eclipse.
- Java AWT/Swing.

  Note: For Java Applets, and for Swing applications with Java versions prior to version 1.6.10, text recognition is supported out-of-the-box. For Swing applications with Java version 1.6.10 or later, you have to add the following command-line element when starting the application:

  ```
  -Dsun.java2d.d3d=false
  ```

  For example:

  ```
  javaw.exe -Dsun.java2d.d3d=false -jar mySwingApplication.jar
  ```
- xBrowser.

Text recognition methods

The following methods enable you to interact with the text of a control:

`TextCapture`	Returns the text that is within a control. Also returns text from child controls.
`TextClick`	Clicks on a specified text within a control. Waits until the text is found or the *Object resolve timeout*, which you can define in the synchronization options, is over.
`TextRectangle`	Returns the rectangle of a certain text within a control or a region of a control.
`TextExists`	Determines whether a given text exists within a control or a region of a control.

Text click recording

Text click recording is enabled by default. To disable text click recording, click **Silk4J** > **Edit Options** > **Recording** and uncheck the **OPT_RECORD_TEXT_CLICK** check box.

When text click recording is enabled, Silk4J records `TextClick` methods instead of clicks with relative coordinates. Use this approach for controls where `TextClick` recording produces better results than normal coordinate-based clicks. You can insert text clicks in your script for any control, even if the text clicks are not recorded.

If you do not whish to record a `TextClick` action, you can turn off text click recording and record normal clicks.

The text recognition methods prefer whole word matches over partially matched words. Silk4J recognizes occurrences of whole words previously than partially matched words, even if the partially matched words are displayed before the whole word matches on the screen. If there is no whole word found, the partly matched words will be used in the order in which they are displayed on the screen.

Example

The user interface displays the text *the hostname is the name of the host*. The following code clicks on *host* instead of *hostname*, although *hostname* is displayed before *host* on the screen:

```
control.textClick("host");
```

The following code clicks on the substring *host* in the word *hostname* by specifying the second occurrence:

```
control.textClick("host", 2);
```

Grouping Silk4J Tests

You can use the `SilkTestCategories` class to run Silk4J tests, write TrueLogs, and filter or group tests with annotations. Define categories of test classes to group the Silk4J tests into these categories, and to run only the tests that are included in a specified category or a subtype of that category.

To include a Silk4J test in a category, use the `@IncludeCategory` annotation.

Using the category `SilkTestCategories` class enables you to write TrueLogs for the Silk4J tests included in the category. You can also use the `SilkTestSuite` class to write TrueLogs. For additional information, see *Replaying a Test Method from the Command Line*.

Example

The following code sample shows how you can execute the Silk4J tests that are included in a category:

```
public interface FastTests {
}

public interface SlowTests {
}
```

```
public static class A {
  @Test
  public void a() {
    fail();
  }

  @Category(SlowTests.class)
  @Test
  public void b() {
  }
}

  @Category( { SlowTests.class, FastTests.class })
  public static class B {
    @Test
    public void c() {
  }
}

@RunWith(SilkTestCategories.class)
@IncludeCategory(SlowTests.class)
@SuiteClasses( { A.class, B.class })
// Note: SilkTestCategories is a kind of Suite
public static class SlowTestSuite {
}
```

Inserting a Result Comment in a Script

You can add result comments to a test script to provide supplemental information about the test. During the execution of the test, the result comments are added to the TrueLog file of the test.

You can add different comment types for information, warnings, and errors. The following code sample shows an example for each comment type:

```
desktop.logInfo("This is a comment!");
desktop.logWarning("This is a warning!");
desktop.logError("This is an error!");
```

Consuming Parameters from Silk Central

To enable Silk4J to use a parameter that has been set for a test in Silk Central, use the method `System.getProperty("myparam")`.

Concepts

This section includes the conceptual overview topics for Silk4J.

Silk Test Open Agent

The Silk Test Open Agent is the software process that translates the commands in your scripts into GUI-specific commands. In other words, the Open Agent drives and monitors the application that you are testing.

One Agent can run locally on the host machine. In a networked environment, any number of Agents can replay tests on remote machines. However, you can record only on a local machine.

Starting the Silk Test Open Agent

Before you can create a test or run a sample script, the Silk Test Open Agent must be running. Typically, the Agent starts when you launch the product. If you must manually start the Open Agent, perform this step.

Click **Start** > **Programs** > **Silk** > **Silk Test** > **Tools** > **Silk Test Open Agent** . The Silk Test Open Agent icon displays in the system tray.

Open Agent Port Numbers

When the Open Agent starts, a random, available port is assigned to Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4J, Silk4NET, and the application that you are testing. The port numbers are registered on the information service. Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J contact the information service to determine the port to use to connect to the Open Agent. The information service communicates the appropriate port, and Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J connects to that port. Communication runs directly between Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J and the Agent.

By default, the Open Agent communicates with the information service on port 22901. You can configure additional ports for the information service as alternate ports that work when the default port is not available. By default, the information service uses ports 2966, 11998, and 11999 as alternate ports.

Typically, you do not have to configure port numbers manually. However, if you have a port number conflict or an issue with a firewall, you must configure the port number for that machine or for the information service. You can use a different port number for a single machine or you can use the same available port number for all your machines.

Configuring the Port that Clients Use to Connect to the Information Service

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

The default port of the information service is 22901. When you can use the default port, you can type `hostname` without the port number for ease of use. If you do specify a port number, ensure that it matches the value for the default port of the information service or one of the additional information service ports. Otherwise, communication will fail.

If necessary, you can change the port number that all clients use to connect to the information service.

1. Navigate to the `infoservice.properties.sample` file and open it.

   This file is located in `C:\Documents and Settings\All Users\Application Data\Silk\Silk Test\conf`, where “`C:\Documents and Settings\All Users`” is equivalent to the content of the ALLUSERSPROFILE environment variable, which is set by default on Windows systems.

   This file contains commented text and sample alternate port settings.
2. Change the value for the appropriate port.

   Typically, you configure the information service port settings to avoid problems with a firewall by forcing communication on a specific port.

   Port numbers can be any number from 1 to 65535.

   - `infoservice.default.port` – The default port where the information service runs. By default, this port is set to 22901.
   - `infoservice.additional.ports` – A comma separated list of ports on which the information service runs if the default port is not available. By default, ports 2966, 11998, and 11999 are set as alternate ports.
3. Save the file as `infoservice.properties`.
4. If you are using Silk Test Recorder and you changed the information service port, specify the new port number in the Silk Test Recorder **Global Preferences** dialog box.
5. Restart the Open Agent, the Silk Test client, and the application that you want to test.

Configuring the Port that the Silk Test Client or the Test Application Uses to Connect to the Open Agent

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

If necessary, change the port number that the Silk Test client or the application that you want to test uses to connect to the Open Agent.

1. Navigate to the `agent.properties.sample` file and open it.

   By default, this file is located at: `%APPDATA%\Silk\Silk Test\conf`, which is typically `C:\Documents and Settings\<`*`user name`*`>\Application Data\Silk\Silk Test\conf` where <*user name*> equals the current user name.
2. Change the value for the appropriate port.

   Typically, you configure port settings to resolve a port conflict.

   Note: Each port number must be unique. Ensure that the port numbers for the Agent differ from the information service port settings.

   Port numbers can be any number from 1 to 65535.

   Port settings include:

   - `agent.vtadapter.port` – Controls communication between Silk Test Workbench and the Open Agent when running tests.
   - `agent.xpmodule.port` – Controls communication between Silk Test Classic and the Agent when running tests.
   - `agent.autcommunication.port` – Controls communication between the Open Agent and the application that you are testing.
   - `agent.rmi.port` – Controls communication with the Open Agent and Silk4J.

- `agent.ntfadapter.port` – Controls communication with the Open Agent and Silk4NET.

  Note: The ports for Adobe Flex testing are not controlled by this configuration file. The assigned port for Flex application testing is 6000 and increases by 1 for each Flex application that is tested. You cannot configure the starting port for Flex testing.

3. Save the file as `agent.properties`.
4. Restart the Open Agent, the Silk Test client, and the application that you want to test.

Configuring the Port that the Silk Test Client Uses to Connect to Silk Test Recorder

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

If necessary, change the port number that Silk Test Classic, Silk4J, or Silk4NET uses to connect to Silk Test Recorder.

1. Navigate to the `recorder.properties.sample` file and open it.

   By default, this file is located at: `%APPDATA%\Silk\Silk Test\conf`, which is typically `C:\Documents and Settings\<user name>\Application Data\Silk\Silk Test\conf` where <*user name*> equals the current user name.
2. Change the `recorder.api.rmi.port` to the port that you want to use.

   Port numbers can be any number from 1 to 65535.

   Note: Each port number must be unique. Ensure that the port numbers for the Agent settings differ from the recorder and the information service port settings.
3. Save the file as `recorder.properties`.
4. Restart the Open Agent, the Silk Test client, and the application that you want to test.

Base State

An application's base state is the known, stable state that you expect the application to be in before each test case begins execution, and the state the application can be returned to after each test case has ended execution. This state may be the state of an application when it is first started.

When you create a class for a Web application or standard configuration, Silk4J automatically creates a base state.

Base states are important because they ensure the integrity of your tests. By guaranteeing that each test case can start from a stable base state, you can be assured that an error in one test case does not cause subsequent test cases to fail.

Silk4J automatically ensures that your application is at its base state during the following stages:

- Before a test runs
- During the execution of a test
- After a test completes successfully

Modifying the Base State

You can change the executable location, working directory, locator, or URL of the base state if necessary. For example, if you want to launch tests from a production Web site that were previously tested on a testing Web site, change the base state URL and the tests will run in the new environment.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.
2. Click **Edit Base State**. The **Edit Base State** dialog box opens.
3. In the **Executable** text box, type the executable name and file path of the application that you want to test.
   For example, you might type `C:\Program Files\Internet Explorer\IEXPLORE.EXE` to specify Internet Explorer.
4. To use a command line pattern in combination with the executable file, in the **Command Line Arguments** text box, type the command line pattern.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, `*\javaw.exe` is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you want is enabled for testing. For example, if the command line of the application ends with **com.example.MyMainClass** you might want to use ***com.example.MyMainClass** as the command line pattern.
5. If the application that you want to test depends on a supplemental directory, specify a directory in the **Working Directory** text box.

   For example, if you use a batch file to start a Java application, the batch file may reference a JAR file that relies on a relative path. In this case, specify a working directory to reconcile the relative path.
6. In the **Locator** text box, type the XPath locator string or the object map ID that identifies the main window of the application.
7. If you are testing a Web site, in the **Url** text box, type the Web address for the Web page to launch when a test begins.
8. Click **OK**.

Application Configuration

An application configuration defines how Silk4J connects to the application that you want to test. Silk4J automatically creates an application configuration when you create the base state. However, at times, you might need to modify, remove, or add an additional application configuration. For example, if you are testing an application that modifies a database and you use a database viewer tool to verify the database contents, you must add an additional application configuration for the database viewer tool.

An application configuration includes the:

- Executable pattern

  All processes that match this pattern are enabled for testing. For example, the executable pattern for Internet Explorer is `*\IEXPLORE.EXE`. All processes whose executable is named `IEXPLORE.EXE` and that are located in any arbitrary directory are enabled.
- Command line pattern

  The command line pattern is an additional pattern that is used to constrain the process that is enabled for testing by matching parts of the command line arguments (the part after the executable name). An application configuration that contains a command line pattern enables only processes for testing that match both the executable pattern and the command line pattern. If no command-line pattern is defined, all processes with the specified executable pattern are enabled. Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, `*\javaw.exe` is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you want is enabled for testing. For example, if the command line of the application ends with **com.example.MyMainClass** you might want to use ***com.example.MyMainClass** as the command line pattern.

Modifying an Application Configuration

An application configuration defines how Silk4J connects to the application that you want to test. Silk4J automatically creates an application configuration when you create the base state. However, at times, you might need to modify, remove, or add an additional application configuration. For example, if you are testing an application that modifies a database and you use a database viewer tool to verify the database contents, you must add an additional application configuration for the database viewer tool.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.
2. To add an additional application configuration, click **Add**.

   The **New Application Configuration** wizard opens.

   a) Select the type of the application that you want to test and click **Next**.

   b) Click the configuration that corresponds with the application that you want to test.

      If the application that you want to test does not appear in the list, uncheck the **Hide processes without caption** check box. This option, checked by default, is used to filter only those applications that have captions.

   c) If you have selected to test a Web application, select whether you want to use an existing browser instance or start a new one.

   d) Click **Finish**. The executable pattern of the application is added to the list of application configurations in the **Edit Application Configurations** dialog box.
3. To remove an application configuration, click **Remove** next to the appropriate application configuration.
4. To edit an application configuration, click **Edit Base State**. The **Edit Base State** dialog box opens.
5. Click **OK**.

Application Configuration Errors

When the program cannot attach to an application, the following error message opens:
Failed to attach to application <Application Name>. For additional information, refer to the Help.

In this case, one or more of the issues listed in the following table may have caused the failure:

Issue	Reason	Solution
Time out	• The system is too slow. • The size of the memory of the system is too small.	Use a faster system or try to reduce the memory usage on your current system.
User Account Control (UAC) fails	You have no administrator rights on the system.	Log in with a user account that has administrator rights.
Command-line pattern	The command-line pattern is too specific. This issue occurs especially for Java. The replay may not work as intended.	Remove ambiguous commands from the pattern.

Desktop Class

The Desktop class is the entry point for accessing the application that you are testing and the Open Agent.

The Desktop class represents a desktop of a specific machine. Therefore, the desktop is associated with one agent, which runs on that machine.

Contacting Micro Focus

Micro Focus is committed to providing world-class technical support and consulting services. Micro Focus provides worldwide support, delivering timely, reliable service to ensure every customer's business success.

All customers who are under a maintenance and support contract, as well as prospective customers who are evaluating products are eligible for customer support. Our highly trained staff respond to your requests as quickly and professionally as possible.

Visit *http://supportline.microfocus.com/assistedservices.asp* to communicate directly with Micro Focus SupportLine to resolve your issues or email supportline@microfocus.com.

Visit Micro Focus SupportLine at *http://supportline.microfocus.com* for up-to-date support news and access to other support information. First time users may be required to register to the site.

Information Needed by Micro Focus SupportLine

When contacting Micro Focus SupportLine, please include the following information if possible. The more information you can give, the better Micro Focus SupportLine can help you.

- The name and version number of all products that you think might be causing an issue.
- Your computer make and model.
- System information such as operating system name and version, processors, and memory details.
- Any detailed description of the issue, including steps to reproduce the issue.
- Exact wording of any error messages involved.
- Your serial number.

To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Index