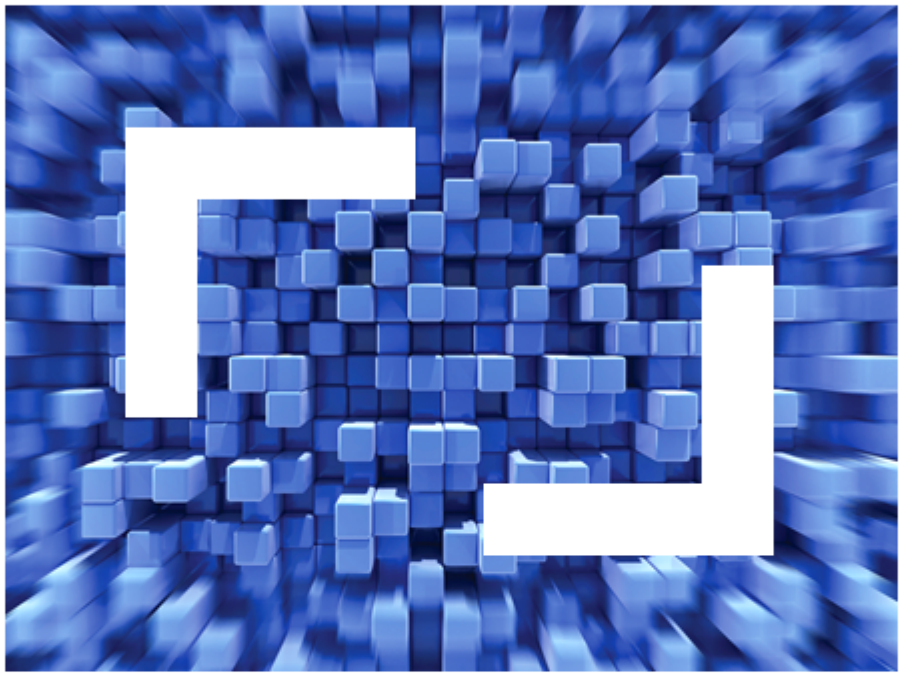


SilkTest 2011

Silk4J User Guide

Borland®
(A MICRO FOCUS COMPANY)

MICRO® FOCUS

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

2011-10-07

Contents

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
- Silverlight
- Windows API-based client/server (Win32)
- Windows Forms
- Windows Presentation Foundation (WPF)
- xBrowser (Web applications)

The *Silk4J User Guide* provides detailed information about Silk4J.

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

The following offer additional assistance, information, and resources:

- *Technical Publications Web Site*
- Borland has contracted for support of this product to be provided by its strategic partner Micro Focus. For support visit *Customer Care*.
- To contact Borland, visit the *Borland Home Page*.

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.

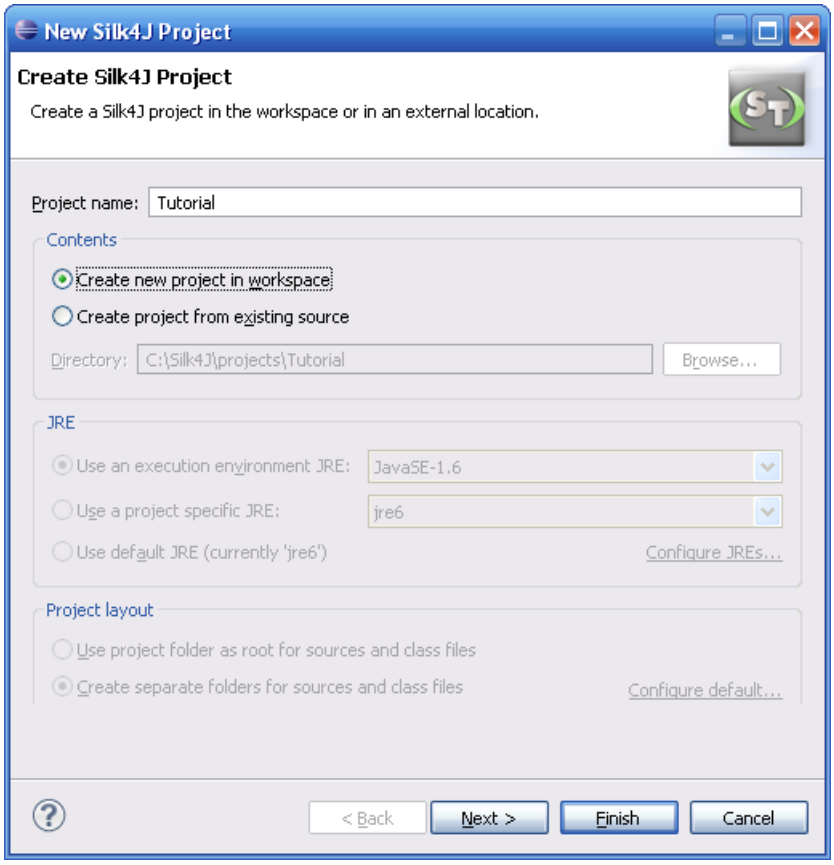
Note: You must have local administrator privileges to run Silk4J.

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:
   - Choose **File** > **New** > **Silk4J Project** .
   - Click the drop-down arrow next to the SilkTest 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.
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`.

Creating a Test Class for the Insurance Company Web Application

Create a class and a test method for the Insurance Company Web application. For a detailed version of how to add a class and configure test applications for each technology type, see *Creating a Test Class* in the *Creating Test Methods* section of the Silk4J User Guide.

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

   The **New Silk4J Test Class** 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`.
4. In the **Name** text box, specify the name for the test class.
   For example, type: `AutoQuoteInput`.
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** list box, select **Internet Explorer**.

   You can use Firefox 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.

Creating a Test Method for the Insurance Company Web Application

Create a test method that navigates to the **Agent Lookup** page in the Insurance Company Web application.

1. In the Package Explorer, perform one of the following steps:

   - Click the name of your project and choose **File** > **New** > **Silk4J Test Method** .

- Click the drop-down arrow next to the SilkTest toolbar icon  and choose **New Silk4J Test Method**.
- If you installed or updated Silk4J to an existing Eclipse location, choose **File** > **New** > **Other** . Expand the Silk4J folder and double-click **Silk4J Test Method**.

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

2. In the **Test method** text box, specify a name for the test method.
   For example, type `realtorPage`.
3. Click **Browse** in the **Test class** text box, select the existing test class that you created previously, and then click **OK**.
   For example, browse to **AutoQuoteInput - com.example** and then click **OK**.
4. Click **Finish**.
   The Insurance Company Web site opens.
5. In the Insurance Company Web application, move your mouse over the **Choose One** drop-down list on the **Automobile Instant Quote** page, click **Agent Lookup** and press `Ctrl+Alt` when the new page opens.
   The element identifier is displayed in the **Active Object** text box in the **Recording** dialog box.
6. Click **Stop Recording**.
   A new method is added to the existing class.
7. Choose **File** > **Save** to save the method.
   Every time you save, Eclipse compiles the source files. If something is incorrect, Eclipse underlines it with a red line.

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

Replaying Test Methods

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

The test results display in the JUnit view as the test runs. If the test passes, the status bar is green. If the test fails, the status bar is red. You can click a failed test to display the stack trace in the Failure Trace area.

Reviewing Sample Projects and Scripts

Silk4J provides several sample projects and scripts that you can use to better understand the product.

Installing Sample Applications

Silk4J provides several sample applications that you can use with tutorials or to create sample tests. When you download the sample applications, you also download sample projects.

1. Navigate to *http://techpubs.borland.com/silk_gauntlet/SilkTest/*.
2. To download sample applications and projects for Java AWT, Java Swing, Java SWT, Microsoft .NET (including Windows Forms and WPF applications), Windows-based client server, and xBrowser applications, click **SampleApplications**.
   You are prompted to open or save the `SampleApplications.zip` file.
3. After you download the sample zip files, unzip and install the sample applications.

Note: We no longer provide Adobe Flex sample applications that you can install. However, you can access Flex sample applications at *http://demo.borland.com/flex/SilkTest2011/index.html*.

Importing Silk4J Sample Projects

Before you perform this task, you must download and install the sample applications.

Use the sample projects to view typical script configurations and test method execution.

1. In the Eclipse workspace, choose **File** > **Import** .
   The **Import** wizard opens.
2. In the menu tree, click the plus sign (+) to expand the **General** folder and select **Existing Projects into Workspace**.
3. Click **Next**.
   The **Import Projects** dialog box opens.
4. In the **Select root directory** text box, click **Browse**.
   The **Browse For Folder** dialog box opens.
5. Navigate to the sample script for the technology type that you want to test.

Environment	File Location
Java SWT	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\SWT`
Windows Forms	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\Windows Forms`
Windows Presentation Foundation (WPF)	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\WPF`
xBrowser	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\xBrowser`

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

6. Select the root directory for the sample and then click **OK**.
7. In the **Projects** section, select the project that you want to import and then click **Finish**.
   The sample project shows in the **Package Explorer** view.

Running a Sample Test Method

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

1. Navigate to the sample project.
2. Perform one of the following steps:

   Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

   - Right-click the package name in the Package Explorer to run all tests in the project.

   For example, right-click **com.borland.flex.store**.

   - Right-click the class name in the Package Explorer to run all test methods for only that class.

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

Replaying Test Methods

Run a test method from within Eclipse or using the command line.

Replaying a Test Method from Eclipse

1. Navigate to the test method that you want to test.
2. 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.
3. Choose **Run As** > **JUnit Test** .
   The test results display in the JUnit view as the test runs. If all tests pass, the status bar is green. If one or more tests fail, the status bar is red. You can click a failed test to display the stack trace in the Failure Trace area.

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:\myproject\
   ```
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*.

Replaying a Test Method 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>
```

Setting Script Options

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

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 SilkTest toolbar icon and choose **Edit Options**.
   The **Script Options** dialog box opens.
2. 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.
3. To record absolute values for scroll events, check the **OPT_RECORD_SCROLLBAR_ABSOLUT** check box.
4. To record mouse move actions, check the **OPT_RECORD_MOUSEMOVES** check box.
5. 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.
6. 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 SilkTest 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. 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.

In xBrowser, WPF, Java SWT, and Swing applications, 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 SilkTest 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 application, use the `org.swt.widgets.Widget.setData(String key, Object value)` method.

Note: To include custom attributes in a Swing application, 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 SilkTest 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 SilkTest toolbar icon  and choose **Edit Options**.
   The **Script Options** dialog box opens.
2. Click the **Transparent Classes** 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 SilkTest 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_XBROWSER_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 SilkTest 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. 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 Environments

Silk4J supports several types of environments.

Adobe Flex Applications

Silk4J provides built-in support for testing Adobe Flex applications using Internet Explorer, FireFox, the Standalone Flash Player, and Adobe AIR.

Note: Silk4J supports testing with Adobe AIR for applications that are compiled with the Flex 4 compiler.

Silk4J also supports multiple application domains in 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, Flex 4.x sub-applications use the `SparkApplication` class. Flex 3.x sub-applications use the `FlexApplication` class.

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

Sample Applications

Silk4J provides sample applications for Flex applications. Sample Flex applications are available at:

- *http://demo.borland.com/flex/SilkTest2011/index.html*

Supported Controls

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

Flex applications typically include custom controls. You can configure Silk4J to test custom Flex controls. For details about how to configure Silk4J to test a custom control, see *Including Flex Custom Controls in a Test*.

For a list of supported attributes, see *Supported Attribute Types*.

Styles in Adobe Flex Applications

For applications developed in Adobe Flex 3.x, SilkTest Workbench 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(1)`. 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(1)
   ```
   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. The locator attributes support the wildcards ? and *.

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

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.

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 playback support. To create automation support, the test application must be modified and the Open Agent must be extended.

Before you test a custom control, 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 `<SilkTest 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.

   SilkTest 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 `<<SilkTest_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.

SilkTest 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 `<SilkTest_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 SilkTest 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>
```

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`.

Java AWT/Swing Applications and Applets

Silk4J provides built-in support for testing applications and applets that use Java 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.

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

After you install the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Java Swing** and select the sample application.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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

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

For a list of supported attributes, see *Supported Attribute 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. The locator attributes support the wildcards ? and `*`.

Dynamically Invoking Java 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.

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.

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 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 SilkTest 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 Applications

Silk4J provides support for testing applications that use widgets from the Standard Widget Toolkit (SWT) controls.

Silk4J 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 that use the IBM JDK or the Sun JDK.
- Any Eclipse-based application that uses SWT widgets for rendering. Silk4J supports both Eclipse IDE-based applications and RCP-based applications.

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

After you install the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Java SWT** and select the sample application that you want to use.

Import the Java SWT sample project from `Documents and Settings\All Users\Shared Documents\SilkTest\samples\Silk4J\SWT`. Run the sample scripts to better understand the product.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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

Supported Controls

For a complete list of the controls available for Java SWT testing, view a list of the supported Java SWT classes in the *API Reference*:

- `com.borland.silktest.jtf.swt` – contains Java SWT specific classes
- `com.borland.silktest.jtf` – contains classes from the common set, analogous to `winclass.inc` in SilkTest Classic 4Test

For a list of supported attributes, see *Supported Attribute Types*.

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. The locator attributes support the wildcards ? and *.

Dynamically Invoking Java 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.

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.

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 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`.

Windows API-based Client/Server Applications

Silk4J provides built-in support for testing Microsoft Windows API-based applications.

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

After you install the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Win32** and select the sample application that you want to use.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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

Supported Controls

For a complete list of the controls available for Windows API-based testing, view a list of the supported Windows classes in the *API Reference*:

- `com.borland.silktest.jtf.win32` - contains Windows API specific classes
- `com.borland.silktest.jtf` - contains classes from the common set, analogous to `winclass.inc` in SilkTest Classic 4Test

For a list of supported attributes, see *Supported Attribute Types*.

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. 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.

Rumba Support

Rumba is the world's premier Windows desktop terminal emulation solution. SilkTest 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 Silk4J 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*.

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. The locator attributes support the wildcards ? and *.

SAP Applications

Silk4J provides support for testing applications that use SAP controls.

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

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

Note: For SAP applications, you must set `Ctrl+Shift` as the shortcut key combination to use to pause recording. To change the default setting, in the **Script Options** dialog box, check the **OPT_ALTERNATE_RECORD_BREAK** check box.

Supported Controls

For a complete list of the controls available for SAP testing, view a list of the supported SAP classes in the `com.borland.silktest.jtf.sap` package in the *API Reference*.

For a list of supported attributes, see *Supported Attribute Types*.

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. The locator attributes support the wildcards ? and `*`.

Dynamically Invoking SAP Methods

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.

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`.

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 Silverlight 3 (Silverlight Runtime 4) or Silverlight 4 (Silverlight Runtime 4)

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

- Silverlight applications that run in Windows Internet Explorer.
- Silverlight applications that run in Mozilla Firefox 4.0 or later.
- Out-of-Browser Silverlight applications.

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

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*.

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. 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 Silk4J 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 SilkTest
`<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"]`

Dynamically Invoking Silverlight 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.

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.

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

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");
```

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>`

Windows Forms Applications

Silk4J provides built-in support for testing standalone Windows Forms applications. Silk4J can play back controls embedded in .NET Framework version 2.0 and later.

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

After you install the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Microsoft .NET** > **Windows Forms Sample Application**.

Import the Windows Forms sample project from `Documents and Settings\All Users\Shared Documents\SilkTest\samples\Silk4J\Windows Forms`. Run the sample scripts to better understand the product.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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

Supported Controls

For a complete list of the controls available for Windows Forms testing, view a list of the supported Windows Forms classes in the *API Reference*:

- `com.borland.silktest.jtf.windowsforms` – contains Windows Forms specific classes
- `com.borland.silktest.jtf.win32` – contains Windows API specific classes, which the Win Forms technology domain is based on

- `com.borland.silktest.jtf` – contains classes from the common set, analogous to `winclass.inc` in SilkTest Classic 4Test

For a list of supported attributes, see *Supported Attribute Types*.

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. The locator attributes support the wildcards ? and *.

Dynamically Invoking Windows Forms 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.

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.

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 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) Applications

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

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

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

After you install the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Microsoft .NET** and select the sample application that you want to use.

Import the WPF sample project from `Documents and Settings\All Users\Shared Documents\SilkTest\samples\Silk4J\WPF`. Run the sample scripts to better understand the product.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

Supported Controls

Silk4J includes record and replay support for Windows Presentation Foundation (WPF) controls. In Silk4J 2009, WPF replay support was provided. However, with the release of Silk4J 2010, the earlier WPF controls, which were prefixed with MSUIA, are deprecated and users should use the new WPF technology domain instead. When you record new test cases, Silk4J automatically uses the new WPF technology domain.

If you recorded tests with Silk4J 2009 that use the earlier MSUIA technology domain, the tests will continue to work. However, if you manually included the constant `TechDomain.WPF` (for the `Desktop.attach` or

the `Desktop.executeBaseState` method) with tests that use the earlier MSUIA classes, you need to change the value to `TechDomain.MSUIA` to run the tests successfully.

All Silk4J classes that work with MSUIA are marked as deprecated. Note that Eclipse strikes through all occurrences of deprecated APIs.

Silk4J uses Microsoft UI Automation (MSUIA) to automate WPF applications. For a complete list of the controls available for WPF testing, view a list of the supported WPF classes in the `com.borland.silktest.jtf.wpf` package in the *API Reference*. For a complete list of the deprecated controls, view a list of the deprecated WPF classes in the `com.borland.silktest.jtf.msuia` package in the *API Reference*.

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. 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']"`

Attribute Type	Description	Example
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 SilkTest 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

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.

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.

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 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

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");
```

Updating the Constant TechDomain.WPF to Use the Deprecated WPF TechDomain

If you recorded tests with Silk4J 2009 that use the earlier MSUIA technology domain, the tests will continue to work. However, if you manually included the constant `TechDomain.WPF` (for the `Desktop.attach` or the `Desktop.executeBaseState` method) with tests that use the earlier MSUIA classes, you need to change the value to `TechDomain.MSUIA` to run the tests successfully.

1. Navigate to the constant `TechDomain.WPF` for the `Desktop.attach` or the `Desktop.executeBaseState` method.
2. To continue using the deprecated WPF controls, change the constant to `TechDomain.MSUIA`.
   For example, change:

   ```
    desktop.attach("C:/myWpfApplication.exe",
   TechDomain.WPF);
   ```

   to:

   ```
    desktop.attach("C:/myWpfApplication.exe",
   TechDomain.MSUIA);
   ```

   Change:

   ```
    desktop.executeBaseState(new
   BaseState("myWpfApplication.exe", "//MsuiaWindow[@caption='my main
   window']", TechDomain.WPF));
   ```

   to:

   ```
    desktop.executeBaseState(new
   BaseState("myWpfApplication.exe", "//MsuiaWindow[@caption='my main
   window']", TechDomain.MSUIA));
   ```

Web Applications

Silk4J provides support for testing Web applications. You can test applications that use Internet Explorer, Firefox, or embedded browser controls. For example, you can test an application that runs in Internet Explorer or a browser that is embedded within a Java SWT application.

Sample Applications, Projects, and Scripts

Silk4J provides sample applications, projects, and scripts. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also.

Sample Web applications are available at:

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

Silk4J contains a sample Java SWT test application that contains an embedded browser. Sample applications are located at **Start** > **Programs** > **Silk** > **SilkTest** > **Sample Applications** > **Java SWT** > **SWT Test Application <version>** . Select the sample application version that is right for your environment. Choose **Control** > **Standard Ctrl Sample** and then click the **Browser** tab.

Import the xBrowser sample project from `Documents and Settings\All Users\Shared Documents\SilkTest\samples\Silk4J\xBrowser`. Run the sample scripts to better understand the product. The sample scripts show different ways to encapsulate the Silk4J native API to enhance the maintainability and readability of your test scripts.

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public\Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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

Quick Start Tutorial

The Quick Start Tutorial provides a step-by-step introduction to using Silk4J to test a Web application.

Supported Controls

For a complete list of the controls available for Web application testing, view a list of the supported xBrowser classes in the `com.borland.silktest.jtf.xbrowser` package in the *API Reference*.

For a list of supported attributes, see *Supported Attribute Types*.

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.

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`	`Click`
	`DomDoubleClick`	`DoubleClick`
	`DomMouseMove`	`MoveMouse`
		`PressMouse`
		`ReleaseMouse`
Keyboard Actions	not available	`TypeKeys`
Specialized Functions	`Select`	not available
	`SetText`	
	etc.	

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 Firefox and Internet Explorer. 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']
```

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

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.

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.

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
- 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`.

Sample Projects and Scripts

Use the sample projects and scripts that Silk4J provides to view typical script configurations and test case execution.

Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. When you download the sample applications, sample Silk4J projects and scripts are included also. After you review the samples, get started creating your test cases. You can modify these throughout the testing cycle as necessary.

Environment	File Location
Java SWT	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\SWT`
Windows Forms	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\Windows Forms`
Windows Presentation Foundation (WPF)	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\WPF`
xBrowser	`Documents and Settings\All Users\Shared Documents \SilkTest\samples\Silk4J\xBrowser`

Note: For Windows Vista and Windows 7.0 operating systems, the file location is `Users\Public \Documents\SilkTest\samples\Silk4J\` rather than `Documents and Settings\All Users\Shared Documents\SilkTest\samples`.

Note: The sample scripts contain a path in the base state to the location of the application under test. If SilkTest is not installed in the default location, edit the base state to adapt the paths to reference the correct location.

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.

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.

Dynamic Object Recognition

Dynamic object recognition enables you to write test methods that use XPath queries to find and identify objects. Dynamic object recognition uses a `Find` or `FindAll` method to identify an object in a test method. For example, the following query finds the first button with the caption "ok" that is a child of a given window:

```
Dim okButton = window.find("//PushButton[@caption=ok]")
```

Examples of the types of test environments where dynamic object recognition works well include:

- In any application environment where the graphical user interface is undergoing changes.

  For example, to test the **Check Me** check box in a dialog that belongs to a menu where the menu and the dialog name are changing, using dynamic object recognition enables you to test the check box without concern for what the menu and dialog name are called. You can then verify the check box name, dialog name, and menu name to ensure that you have tested the correct component.
- In a Web application that includes dynamic tables or text.

  For example, to test a table that displays only when the user points to a certain item on the Web page, use dynamic object recognition to have the test method locate the table without regard for which part of the page needs to be clicked in order for the table to display.
- In an Eclipse environment that uses views.

  For example, to test an Eclipse environment that includes a view component, use dynamic object recognition to identify the view without regard to the hierarchy of objects that need to open prior to the view.

Benefits of Using Dynamic Object Recognition

The benefits of using dynamic object recognition include:

- Dynamic object recognition uses a subset of the XPath query language, which is a common XML-based language defined by the World Wide Web Consortium, W3C.
- Dynamic object recognition requires a single object rather than a repository of objects for the application that you are testing. Using XPath queries, a test case can locate an object using a `Find` command followed by a supported XPath construct.

XPath Basic Concepts

Silk4J supports a subset of the XPath query language. For additional information about XPath, see *http://www.w3.org/TR/xpath20/*.

Basic Concepts

XPath expressions rely on the *current context*, 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. For example:

- `"//Shell"` finds all shells in any hierarchy relative to the current object.
- `"Shell"` finds all shells that are direct children of the current object.

Additionally, some XPath expressions are context sensitive. For example, `myWindow.find(xPath)` makes `myWindow` the current context.

Supported XPath Subset

Silk4J supports a subset of the XPath query language. Use a `FindAll` or a `Find` command followed by a supported construct to create a test case.

The following table lists the constructs that Silk4J supports.

Supported XPath Construct	Sample	Description
Attribute	`MenuItem[@caption='abc']`	Finds all menu items with the given caption attribute in their object definition that are children of the current context. The following attributes are supported: caption (without caption index), priorlabel (without index), windowid.
Index	`MenuItem[1]`	Finds the first menu item that is a child of the current context. Indices are 1-based in XPath.
Logical Operators: and, or, not, =, !=	`MenuItem[not(@caption='a' or @windowid!='b') and @priorlabel='p']`	
.	`TestApplication.Find("//Dialog[@caption='Check Box']/./.")`	Finds the context on which the `Find` command was executed. For instance, the sample could have been typed as `TestApplication.Find("//Dialog[@caption='Check Box']")`.
..	`Desktop.Find("//PushButton[@caption='Previous']/../PushButton[@caption='Ok']")`	Finds the parent of an object. For instance, the sample finds a PushButton with the caption "Ok" that has a sibling PushButton with the caption "Previous."
/	`/Shell`	Finds all shells that are direct children of the current object.  **Note:** "`/Shell`" is equivalent to "`Shell`".
/	`/Shell/MenuItem`	Finds all menu items that are a child of the current object.
//	`//Shell`	Finds all shells in any hierarchy relative to the current object.
//	`//Shell//MenuItem`	Finds all menu items that are direct or indirect children of a Shell that is a direct child of the current object.
//	`//MenuItem`	Finds all menu items that are direct or indirect children of the current context.

Supported XPath Construct	Sample	Description
*	`*[@caption='c']`	Finds all objects with the given caption that are a direct child of the current context.
*	`//MenuItem/*/Shell`	Finds all shells that are a grandchild of a menu item.

The following table lists the XPath constructs that Silk4J does not support.

Unsupported XPath Construct	Example
Comparing two attributes with each other	`PushButton[@caption = @windowid]`
An attribute name on the right side is not supported. An attribute name must be on the left side.	`PushButton['abc' = @caption]`
Combining multiple XPath expressions with 'and' or 'or'.	`PushButton [@caption = 'abc'] or .// Checkbox`
More than one set of attribute brackets	`PushButton[@caption = 'abc] [@windowid = '123']` (use `PushButton [@caption = 'abc and @windowid = '123']` instead)
More than one set of index brackets	`PushButton[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	`//[@caption = 'abc']`
	(use `//*[@caption = 'abc']` instead)
	`"//*Button[@caption='abc']"`

XPath Samples

The following table lists sample XPath queries and explains the semantics for each query.

XPath String	Description
`desktop.find("/Shell[@caption='SWT Test Application']")`	Finds the first top-level Shell with the given caption.
`desktop.find("//MenuItem[@caption='Control']")`	Finds the MenuItem in any hierarchy with the given caption.
`myShell.find("//MenuItem[@caption!='Control']")`	Finds a MenuItem in any child hierarchy of myShell that does not have the given caption.
`myShell.find("Menu[@caption='Control']/MenuItem[@caption!='Control']")`	Looks for a specified MenuItem with the specified Menu as parent that has myShell as parent.
`myShell.find("//MenuItem[@caption='Control' and @windowid='20']")`	Finds a MenuItem in any child hierarchy of myWindow with the given caption and windowid.
`myShell.find("//MenuItem[@caption='Control' or @windowid='20']")`	Finds a MenuItem in any child hierarchy of myWindow with the given caption or windowid.

XPath String	Description
`desktop.findAll("/Shell[2]/*/PushButton")`	Finds all PushButtons that have an arbitrary parent that has the second top-level shell as parent.
`desktop.findAll("/Shell[2]//PushButton")`	Finds all PushButtons that use the second shell as direct or indirect parent.
`myBrowser.find("//FlexApplication[1]//FlexButton[@caption='ok']")`	Looks up the first FlexButton within the first FlexApplication within the given browser.
`myBrowser.findAll("//td[@class='abc*']//a[@class='xyz']")`	Finds all link elements with attribute class `xyz` that are direct or indirect children of `td` elements with attribute class `abc*`.

Silk4J 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 the `Find` or `FindAll` methods in your scripts. Using the Locator Spy ensures that the XPath query string is valid.

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. 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. 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. 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. 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. 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. 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. 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 Silk4J 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 SilkTest
`<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 Firefox and Internet Explorer. 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. 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. 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. 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 SilkTest 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 a button on a dialog box in a WPF application you can type:

```
Dim button = dialog.Find("//WPFButton[@IsDefault=true]")
```

or alternatively

```
Dim button = dialog.WPFButton("@IsDefault=true")
```

This works because SilkTest Workbench exposes a property called `IsDefault` for the WPF button control.

Example

If you want to identify a button in a WPF application with the font size 12 you can type:

```
Dim button = dialog.Find("//WPFButton[@FontSize=12]")
```

or alternatively

```
Dim button = dialog.WPFButton("@FontSize=12")
```

This works because the underlying control in the application under test, in this case the WPF button, has a property called `FontSize`.

Technical Support

All customers who are under a maintenance and support contract, as well as prospective customers who are evaluating products are eligible for technical support. Our highly trained Technical Support staff will respond to your requests as quickly and professionally as possible.

How to access Technical Support

Borland has contracted for support of this product to be provided by its strategic partner Micro Focus. For support visit *Customer Care*.

Index

J

L

M

N

O

P

Q

R

S

T

V

W

X