Borland
A MICRO FOCUS COMPANY

Silk Test 13.5

Silk4J User Guide

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

Copyright © 2012 Micro Focus. All rights reserved. Portions Copyright © 2012 Borland Software Corporation (a Micro Focus company).

MICRO FOCUS, the Micro Focus logo, and Micro Focus product names are trademarks or registered trademarks of Micro Focus IP Development Limited or its subsidiaries or affiliated companies in the United States, United Kingdom, and other countries.

BORLAND, the Borland logo, and Borland product names are trademarks or registered trademarks of Borland Software Corporation or its subsidiaries or affiliated companies in the United States, United Kingdom, and other countries.

All other marks are the property of their respective owners.

2012-09-19

Contents

Welcome to Silk4J 6
Silk4J 7
Silk Test Product Suite 7
Product Notification Service 9
Silk4J Quick Start Tutorial 10
Creating a Silk4J Project 10
Creating a Test Class for the Insurance Company Web Application 11
Creating a Test Method for the Insurance Company Web Application 12
Replaying Test Methods 13
Replaying Test Methods 14
Replaying Test Methods from Eclipse 14
Replaying a Test Method from the Command Line 14
Troubleshooting when Replaying Test Methods from Ant 15
Enabling TrueLog 15
Enabling TrueLog 15
Why is TrueLog Not Displaying Non-ASCII Characters Correctly? 16
Setting Script Options 17
Setting TrueLog Options 17
Setting Recording Preferences 17
Setting Browser Recording Options 18
Setting Custom Attributes 19
Setting Classes to Ignore 20
Setting WPF Classes to Expose During Recording and Playback 20
Setting Synchronization Options 20
Setting Replay Options 21
Setting Silk4J Preferences 23
Testing Environments 24
Adobe Flex Applications 24
Styles in Adobe Flex Applications 24
Configuring Flex Applications for Adobe Flash Player Security Restrictions 25
Attributes for Adobe Flex Applications 25
Testing Adobe Flex Custom Controls 26
Dynamically Invoking Flex Methods 35
Java AWT/Swing Applications and Applets 35
Attributes for Java AWT/Swing Applications 36
Dynamically Invoking Java Methods 36
Configuring Silk4J to Launch an Application that Uses the Java Network Launching Protocol (JNLP) 37
Determining the priorLabel in the Java AWT/Swing Technology Domain 38
Java SWT Applications 38
Attributes for Java SWT Applications 39
Dynamically Invoking Java Methods 39
Windows API-Based Client/Server Applications 40
Attributes for Windows API-based Client/Server Applications 40
Determining the priorLabel in the Win32 Technology Domain 41
Rumba Support 41
Locator Attributes for Identifying Rumba Controls 41
SAP Applications 42
Attributes for SAP Applications 42
Dynamically Invoking SAP Methods 42

Silverlight Application Support 43
Locator Attributes for Identifying Silverlight Controls 44
Dynamically Invoking Silverlight Methods 45
Troubleshooting when Testing Silverlight Applications 46
Windows Forms Applications 47
Attributes for Windows Forms Applications 47
Dynamically Invoking Windows Forms Methods 47
Windows Presentation Foundation (WPF) Applications 48
Attributes for Windows Presentation Foundation (WPF) Applications 49
Classes that Derive from the WPFItemsControl Class 50
Custom WPF Controls 50
Dynamically Invoking WPF Methods 51
Updating the Constant TechDomain.WPF to Use the Deprecated WPF TechDomain 52
Web Applications 53
Page Synchronization for xBrowser 53
Comparing API Playback and Native Playback for xBrowser 54
Attributes for Web Applications 55
Prerequisites for Replaying Tests with Google Chrome 56
Limitations for Testing with Google Chrome 56
xBrowser Frequently Asked Questions 57
Active X/Visual Basic Applications 61
Dynamically Invoking ActiveX/Visual Basic Methods 61
64-bit Application Support 62
Best Practices for Using Silk4J 63
Dynamic Object Recognition 64
XPath Basic Concepts 64
Supported XPath Subset 65
XPath Samples 66
Troubleshooting Performance Issues for XPath 67
Locator Spy 68
Supported Attribute Types 68
Attributes for Adobe Flex Applications 68
Attributes for Java AWT/Swing Applications 68
Attributes for Java SWT Applications 69
Attributes for MSUIA Applications 69
Locator Attributes for Identifying Rumba Controls 70
Attributes for SAP Applications 70
Locator Attributes for Identifying Silverlight Controls 70
Attributes for Web Applications 71
Attributes for Windows API-based Client/Server Applications 72
Attributes for Windows Forms Applications 72
Attributes for Windows Presentation Foundation (WPF) Applications 72
Dynamic Locator Attributes 74
Enhancing Tests 75
Calling Windows DLLs 75
Calling a Windows DLL from Within a Script 75
DLL Function Declaration Syntax 75
DLL Calling Example 76
Passing Arguments to DLL Functions 76
Passing Arguments that Can Be Modified by the DLL Function 78
Passing String Arguments to DLL Functions 78
Aliasing a DLL Name 79
Conventions for Calling DLL Functions 79
Custom Controls 79

Supporting a Custom Control 80
Custom Controls Options 81
Windows Accessibility 82
Using Accessibility 82
Enabling Accessibility 82
Dynamic Invoke 83
Text Recognition Support 83
Grouping Silk4J Tests 84
Inserting a Result Comment in a Script 85
Concepts 86
Silk Test Open Agent 86
Starting the Silk Test Open Agent 86
Open Agent Port Numbers 86
Base State 88
Modifying the Base State 89
Application Configuration 89
Modifying an Application Configuration 90
Application Configuration Errors 90
Desktop Class 91
Contacting Micro Focus 92
Information Needed by Micro Focus SupportLine 92

Welcome to Silk4J

Welcome to Silk4J

About Silk4J
Product Suite

What's new

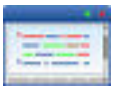
Release Notes

Featured sections

Best Practices for Using Silk4J
Creating Test Methods
Testing Environments

Tutorials and demonstrations

Quick Start Tutorial



Code samples

Enhancing Tests

Online resources

Micro Focus Infocenter
Micro Focus SupportLine
Micro Focus Product Updates
Micro Focus Knowledge Base
Micro Focus Community Forums
Micro Focus Training Store

Provide feedback

Contacting Micro Focus on page 92
Email us feedback regarding this Help

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*.

Silk Test Product Suite

The Silk Test product suite includes the following components:

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

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

Product Notification Service

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

Running the Service

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

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

Silk4J Quick Start Tutorial

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

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

For the sake of simplicity, this guide assumes that you have installed Silk4J and are using the sample Insurance Company Web application.

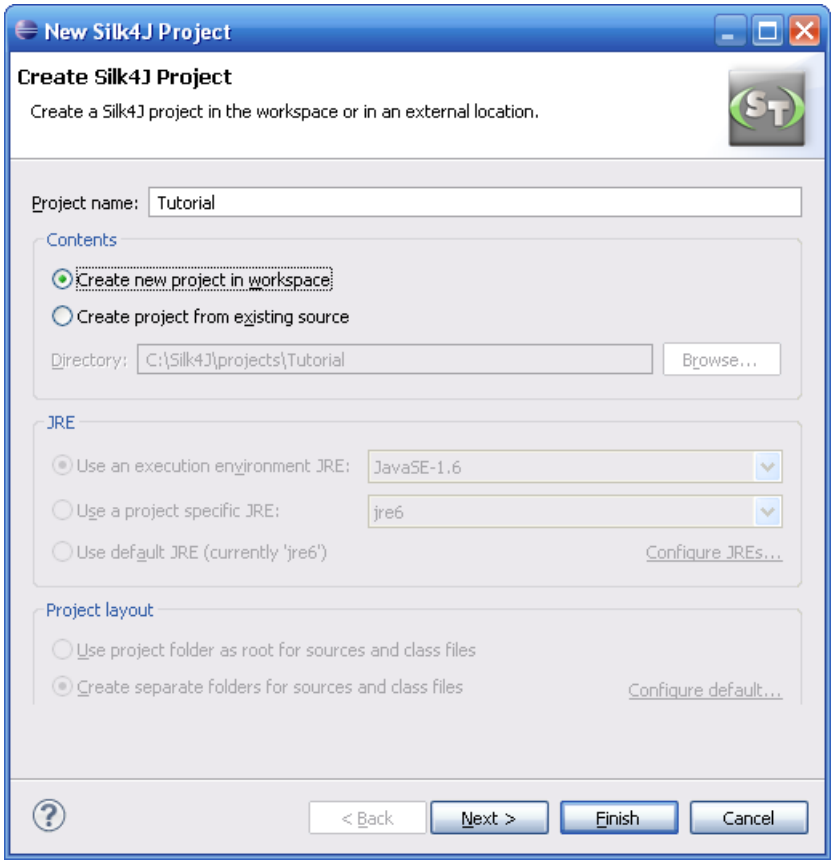
Note: You must have local administrator privileges to run Silk4J.

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 Silk Test toolbar icon and choose **New Silk4J Project**.
   - If you installed or updated Silk4J to an existing Eclipse location, choose **File** > **New** > **Other** . Expand the Silk4J folder and double-click **Silk4J Project**.

   The **New Silk4J Project** wizard opens.
2. In the **Project name** text box, type a name for your project.
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 Silk Test 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** group, select **Internet Explorer**.

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

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

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

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

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

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

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

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 Silk Test 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** > **Silk4J 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.

Replaying Test Methods

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

Replaying Test Methods from Eclipse

1. Navigate to the test method that you want to test.
2. 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** > **Silk4J Test** . The test results display in the JUnit view as the test runs. If all tests pass, the status bar is green. If one or more tests fail, the status bar is red. You can click a failed test to display the stack trace in the Failure Trace area.
4. When the test execution is complete, the **Playback Complete** dialog box opens. Click **Explore Results** to review the TrueLog for the completed test.

Replaying a Test Method from the Command Line

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

1. Set the CLASSPATH to:

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

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

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

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

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

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

Troubleshooting when Replaying Test Methods from Ant

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

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

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

Enabling TrueLog

To enable TrueLog:

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

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

Enabling TrueLog

To enable TrueLog:

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

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

Why is TrueLog Not Displaying Non-ASCII Characters Correctly?

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

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

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

Setting Script Options

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

Setting TrueLog Options

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

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

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

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

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

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

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

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

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

Setting Recording Preferences

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

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

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

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

   Note: For SAP applications, you must set `Ctrl+Shift` as the shortcut key combination.
4. To record absolute values for scroll events, check the **OPT_RECORD_SCROLLBAR_ABSOLUT** check box.
5. To record mouse move actions, check the **OPT_RECORD_MOUSEMOVES** check box.
6. If you record mouse move actions, in the **OPT_RECORD_MOUSEMOVE_DELAY** text box, specify how many milliseconds the mouse has to be motionless before a `MouseMove` is recorded

   By default this value is set to 200.
7. To record `TextClick` actions instead of `Click` actions, on those objects where `TextClick` actions usually are preferable to `Click` actions, check the **OPT_RECORD_TEXT_CLICK** check box.
8. Click **OK**.

Setting Browser Recording Options

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

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

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

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

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

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

   If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.
6. 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 Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Custom Attributes** tab.
3. From the **Select a tech domain** list box, select the technology domain for the application that you are testing.

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

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

   Separate attribute names with a comma.

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

   Note: To include custom attributes in a Java SWT 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 Silk Test toolbar icon and choose **Edit Options**. The **Script Options** dialog box opens.
2. Click the **Transparent Classes** tab.
3. In the **Transparent classes** grid, type the name of the class that you want to ignore during recording and playback.

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

Setting WPF Classes to Expose During Recording and Playback

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

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

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

Setting Synchronization Options

Specify the synchronization and timeout values for Web applications.

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

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

The synchronization algorithm configures the waiting period for the ready state of an invoke call. By default, this value is set to **AJAX**.

4. In the **Synchronization exclude list** text box, type the entire URL or a fragment of the URL for any service or Web page that you want to exclude.

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

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

   For example, you might type:

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

   Separate multiple entries with a comma.

   Note: If your application uses only one service, and you want to disable that service for testing, you must use the HTML synchronization mode rather than adding the service URL to the exclusion list.

5. To specify the maximum time, in milliseconds, to wait for an object to be ready, type a value in the **OPT_SYNC_TIMEOUT** text box.

   By default, this value is set to **300000**.

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

   By default, this value is set to **5000**.

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

   By default, this value is set to **500**.

8. Click **OK**.

Setting Replay Options

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

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

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

4. To ensure that the object that you want to test is active, check the **OPT_ENSURE_ACTIVE_OBJDEF** check box.

5. 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 Windows Internet Explorer, Mozilla Firefox, and the Standalone Flash Player, and Adobe AIR applications built with Flex 4 or later.

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.

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

Sample Applications

Silk4J provides sample applications for Flex applications. Sample Flex applications are available at *http://demo.borland.com/flex/SilkTest13.5/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.

Styles in Adobe Flex Applications

For applications developed in Adobe Flex 3.x, Silk Test 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(MouseButton.LEFT)`. A `click()` without parameters is a high-level click and a click with parameters (such as the button) is replayed as a low-level click.

1. Record the steps that use Flash Player.
2. Navigate to the `Click` method and add a parameter.
   For example, to open the **Open File** dialog box, specify:
   ```
   SparkButton("@caption='Open File Dialog…'").click(MouseButton.LEFT)
   ```
   When you play back the test, it works as expected.

Attributes for Adobe Flex Applications

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

Supported attributes for Flex applications include:

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

Note: Attribute names are case sensitive. 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 `<Silk Test installation directory>\ng\agent\plugins\com.borland.silktest.jtf.agent.customcontrols_<version>\config\classMapping.properties` file:

   ```
   FlexSpinner=customcontrols.FlexSpinner
   ```

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

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

Examples

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

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

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

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

Implementing Automation Support for a Custom Control

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

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

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

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

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

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

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

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

    [Mixin]
    public class SpinnerAutomationDelegate extends BoxAutomationImpl
    {

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

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

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

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

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

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

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

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

```
                helper.replayClick(button);
            }
            return result;
        }

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

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

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

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

    }
}
```

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<TypeInformation>
    <ClassInfo Name="FlexSpinner" Extends="FlexBox">
        <Implementation
            Class="customcontrols.Spinner" />
        <Events>
            <Event Name="Decrement">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="decrement" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
                </Property>
            </Event>
            <Event Name="Increment">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="increment" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
```

```
                </Property>
            </Event>
        </Events>
        <Properties>
            <Property Name="lowerBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="upperBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <!-- expose read and write access for the currentValue property
-->
            <Property Name="currentValue" accessType="both">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="stepSize" accessType="read">
                <PropertyType Type="integer" />
            </Property>
        </Properties>
    </ClassInfo>
</TypeInformation>
```

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

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

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

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

Flex Class Definition File

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

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

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

The XML file has the following basic structure:

```
<TypeInformation>

<ClassInfo>

<Implementation />

<Events>

<Event />

…

</Events>
```

```
<Properties>
<Property />
…
</Properties>
</ClassInfo>
</TypeInformation>
```

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.

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

Supported Controls

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

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

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

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

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

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

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

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

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

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

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

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

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

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

  Control parameters can be passed or returned as TestObject.

Returned Values

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

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

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

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

1. Create a project and test class for the test application.
2. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.

3. Edit the base state to ensure that the Web Start launches during playback.
   a) Click **Edit Base State**. The **Edit Base State** dialog box opens.
   b) In the **Executable** text box, type the absolute path for the javaws.exe.
      For example, you might type:
      ```
      %ProgramFiles%\Java\jre6\bin\javaws.exe
      ```
   c) In the **Command Line Arguments** text box, type the command line pattern that includes the URL to the Web Start.
      ```
      "<url-to-jnlp-file>"
      ```
      For example, for the SwingSet3 application, type:
      ```
      "http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp"
      ```
   d) Click **OK**.
4. Click **OK**. The test uses the base state to start the web-start application and the application configuration executable pattern to attach to javaw.exe to execute the test.

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

Determining the priorLabel in the Java AWT/Swing Technology Domain

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

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

Java SWT 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.

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

Supported Controls

For a complete list of the controls available for Java 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 Silk Test 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

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

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

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

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

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

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

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

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

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

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

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

  Control parameters can be passed or returned as TestObject.

Returned Values

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

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

Windows API-Based Client/Server Applications

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

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

Supported Controls

For a complete list of the 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 Silk Test 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. Silk Test provides built-in support for recording and replaying Rumba.

When testing with Rumba, please consider the following:

- The Rumba version must be compatible to the 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 Windows Internet Explorer or Mozilla Firefox, Silk Test Workbench tests the application using the xBrowser technology domain.

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

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*.

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. For information on the supported automation methods of an object, refer to the SAP GUI Scripting documentation.

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

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

  UI controls can be passed or returned as TestObject.

Returned Values

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

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

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

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

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

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

Dynamically Invoking Silverlight Methods

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

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

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

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

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

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

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

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

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types.

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

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

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

  Control parameters can be passed as `TestObject`.

Supported Methods and Properties

The following methods and properties can be called:

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

Returned Values

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

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

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

  Example

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

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

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.

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

Supported Controls

For a complete list of the controls available for 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 Silk Test 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

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

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

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

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

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

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

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

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

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that Silk4J supports for the control
- All public methods 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.

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

Supported Controls

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	`//WPFButton[@name='okButton']"`

Attribute Type	Description	Example
	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.	
caption	The text that the control displays. When testing a localized application in multiple languages, use the automationId or name attribute instead of the caption.	`//WPFButton[@automationId='Ok']"`
className	The simple .NET class name (without namespace) of the WPF control. Using the class name attribute can help to identify a custom control that is derived from a standard WPF control that Silk4J recognizes.	`//WPFButton[@className='MyCustomButton']"`

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

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

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

Classes that Derive from the WPFItemsControl Class

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

- Working with the control

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

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

Custom WPF Controls

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

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

- Deriving classes from `UserControl`

This is a typical way to create compound controls. Silk4J recognizes these user controls as `WPFUserControl` and provides full support for the contained controls.

- Deriving classes from standard WPF controls, such as `ListBox`

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

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

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

Dynamically Invoking WPF Methods

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

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

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

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

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

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

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

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

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 one of the supported browsers or embedded browser controls. For example, you can test an application that runs in Windows Internet Explorer or a browser that is embedded within a Java SWT application.

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

Sample Web Applications

Sample Web applications are available at:

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

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*.

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 each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single

space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

```
<a>abc
abc</a>
```

Uses the following locator:

```
//A[@innerText='abc abc']
```

Prerequisites for Replaying Tests with Google Chrome

Command-line parameters

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

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

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

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

Limitations for Testing with Google Chrome

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

- Silk Test does not support testing child technology domains of the xBrowser domain with Google Chrome. For example Adobe Flex or Microsoft Silverlight are not supported with Google Chrome.
- Silk Test does not provide native support for Google Chrome. You cannot test internal Google Chrome functionality. For example, in a test, you cannot change the currently displayed Web page by adding text to the navigation bar through Win32. As a workaround, you can use API calls to navigate between Web pages. Silk Test supports handling alerts and similar dialog boxes.

- The page synchronization for Google Chrome is not as advanced as for the other supported browsers. Changing the synchronization mode has no impact on the synchronization for Google Chrome.
- Silk Test does not support the methods `TextClick` and `TextSelect` when testing applications with Google Chrome.
- Silk Test does not recognize the **Log In** and **Cancel** buttons in the authentication dialog box of Google Chrome. Use one of the following solutions to work around this limitation:
  - Specify the username and the password in the URL of the website that you want to test. For example, to log in to the website *www.example.com/loginrequired.html*, use the following code:
    ```
    http://myusername:mypassword@example.com/loginrequired.html
    ```
  - Use `TypeKeys` to enter the username and password in the dialog box. For example, use the following code:
    ```
     desktop.find("//Window[@caption='Authentication Required']/
    Control[2]").TypeKeys("myusername")
    desktop.find("//Window[@caption='Authentication Required']/
    Control[1]").TypeKeys("mypassword<Enter>")
    ```

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

xBrowser Frequently Asked Questions

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

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

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

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

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

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

- `innerHtml` returns all text, including html tags, that is contained by an element.

Consider the following html code.

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

The following table details the different properties that return.

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

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

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

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

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

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

How Can I See Which Browser I Am Currently Using?

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

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

Examples

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

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

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

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

The built in locator generator attempts to create stable locators. However, it is difficult to generate quality locators if no information is available. In this case, the locator generator uses hierarchical information and indices, which results in fragile locators that are suitable for direct record and replay but ill-suited for stable, daily execution. Furthermore, with cross-browser testing, several AJAX frameworks might render different DOM hierarchies for different browsers.

To avoid this issue, use custom IDs for the UI elements of your application.

Logging Output of My Application Contains Wrong Timestamps

This might be a side effect of the synchronization. To avoid this problem, specify the HTML synchronization mode.

My Test Script Hangs After Navigating to a New Page

This can happen if an AJAX application keeps the browser busy (open connections for Server Push / ActiveX components). Try to set the HTML synchronization mode. Check the *Page Synchronization for xBrowser* topic for other troubleshooting hints.

Recorded an Incorrect Locator

The attributes for the element might change if the mouse hovers over the element. Silk4J tries to track this scenario, but it fails occasionally. Try to identify the affected attributes and configure Silk4J to ignore them.

Rectangles Around Elements in Windows Internet Explorer are Misplaced

- Make sure the zoom factor is set to 100%. Otherwise, the rectangles are not placed correctly.
- Ensure that there is no notification bar displayed above the browser window. Silk4J cannot handle notification bars.

Link.Select Does Not Set the Focus for a Newly Opened Window in Internet Explorer

This is a limitation that can be fixed by changing the Browser Configuration Settings. Set the option to always activate a newly opened window.

DomClick(x, y) Is Not Working Like Click(x, y)

If your application uses the `onclick` event and requires coordinates, the `DomClick` method does not work. Try to use `Click` instead.

FileInputField.DomClick() Will Not Open the Dialog

Try to use `Click` instead.

The Move Mouse Setting Is Turned On but All Moves Are Not Recorded. Why Not?

In order to not pollute the script with a lot of useless `MoveMouse` actions, Silk4J does the following:

- Only records a `MoveMouse` action if the mouse stands still for a specific time.

- Only records `MoveMouse` actions if it observes activity going on after an element was hovered over. In some situations, you might need to add some manual actions to your script.

I Need Some Functionality that Is Not Exposed by the xBrowser API. What Can I Do?

You can use `ExecuteJavaScript()` to execute JavaScript code directly in your Web application. This way you can build a workaround for nearly everything.

Why Are the Class and the Style Attributes Not Used in the Locator?

These attributes are on the ignore list because they might change frequently in AJAX applications and therefore result in unstable locators. However, in many situations these attributes can be used to identify objects, so it might make sense to use them in your application.

Dialog is Not Recognized During Replay

When recording a script, Silk4J recognizes some windows as `Dialog`. If you want to use such a script as a cross-browser script, you have to replace `Dialog` with `Window`, because some browsers do not recognize `Dialog`.

For example, the script might include the following line:

```
/BrowserApplication//Dialog//PushButton[@caption='OK']
```

Rewrite the line to enable cross-browser testing to:

```
/BrowserApplication//Window//PushButton[@caption='OK']
```

Why Do I Get an Invalidated-Handle Error When I Call the WaitForProperty Method?

This topic describes what you can do when you call the `WaitForProperty` method and Silk4J displays the following error message: `The handle for this object has been invalidated.`

This message indicates that something caused the object, on which you called `WaitForProperty`, to disappear. For example, if something causes the browser to navigate to a new page, during the `WaitForProperty` call in a Web application, all objects on the previous page are automatically invalidated.

Sometimes the reason for this problem is the built-in synchronization. For example, suppose that the application under test includes a shopping cart, and you have added an item to this shopping cart. You are waiting for the next page to be loaded and for the shopping cart to change its status to `contains items`. If the action, which adds the item, returns too soon, the shopping cart on the first page will be waiting for the status to change while the new page is loaded, causing the shopping cart of the first page to be invalidated. This behavior will result in an invalidated-handle error.

As a workaround, you should wait for an object that is only available on the second page before you verify the status of the shopping cart. As soon as the object is available, you can verify the status of the shopping cart, which is then correctly verified on the second page.

Why Are Clicks Recorded Differently in Windows Internet Explorer 10?

When you record a `Click` on a `DomElement` in Windows Internet Explorer 10 and the `DomElement` is dismissed after the `Click`, then the recording behavior might not be as expected. If another `DomElement` is located beneath the initial `DomElement`, Silk Test records a `Click`, a `MouseMove`, and a `ReleaseMouse`, instead of recording a single `Click`.

A possible workaround for this unexpected recording behavior depends on the application under test. Usually it is sufficient to delete the unnecessary `MouseMove` and `ReleaseMouse` events from the recorded script.

Active X/Visual Basic Applications

Silk4J provides support for testing ActiveX/Visual Basic applications.

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

Dynamically Invoking ActiveX/Visual Basic Methods

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

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

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

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

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

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

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

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

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

Returned Values

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

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

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

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*`.

Troubleshooting Performance Issues for XPath

When testing applications with a complex object structure, for example complex web applications, you may encounter performance issues, or issues related to the reliability of your scripts. This topic describes how you can improve the performance of your scripts by using different locators than the ones that Silk4J has automatically generated during recording.

Note: In general, we do not recommend using complex locators. Using complex locators might lead to a loss of reliability for your tests. Small changes in the structure of the tested application can break such a complex locator. Nevertheless, when the performance of your scripts is not satisfying, using more specific locators might result in tests with better performance.

The following is a sample element tree for the application MyApplication:

```
Root
  Node id=1
    Leaf id=2
    Leaf id=3
    Leaf id=4
    Leaf id=5
  Node id=6
    Node id=7
      Leaf id=8
      Leaf id=9
    Node id=9
      Leaf id=10
```

You can use one or more of the following optimizations to improve the performance of your scripts:

- If you want to locate an element in a complex object structure , search for the element in a specific part of the object structure, not in the entire object structure. For example, to find the element with the identifier 4 in the sample tree, if you have a query like `Root.Find("//Leaf[@id='4']")`, replace it with a query like `Root.Find("/Node[@id='1']/Leaf[@id='4']")`. The first query searches the entire element tree of the application for leafs with the identifier 4. The first leaf found is then returned. The second query searches only the first level nodes, which are the node with the identifier 1 and the node with the identifier 6, for the node with the identifier 1, and then searches in the subtree of the node with the identifier 1 for all leafs with the identifier 4.
- When you want to locate multiple items in the same hierarchy, first locate the hierarchy, and then locate the items in a loop. If you have a query like `Root.FindAll("/Node[@id='1']/Leaf")`, replace it with a loop like the following:

```
public void test() {
  TestObject node;
  int i;

  node = desktop.find("//Node[@id='1']");
```

```
  for (i=1; i<=4; i++)
  node.find("/Leaf[@id='"+i+"']");
}
```

Locator Spy

Use the **Locator Spy** to identify the caption or the XPath locator string for GUI objects. You can copy the relevant XPath locator strings and attributes into methods in your scripts. Additionally, you can manually edit the attributes of the XPath locator strings in your test scripts and validate the changes in the **Locator Spy**. Using the **Locator Spy** ensures that the XPath query string is valid.

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

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

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

Attributes for Web Applications

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

Supported attributes for Web applications include:

- caption (supports wildcards *?* and *)
- all DOM attributes (supports wildcards *?* and *)

Note: Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single

space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

```
<a>abc
abc</a>
```

Uses the following locator:

```
//A[@innerText='abc abc']
```

Attributes for Windows API-based Client/Server Applications

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

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

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

Note: Attribute names are case sensitive. 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 Silk4J recognizes.	`//WPFButton[@className='MyCustomButton']"`

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

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

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

Dynamic Locator Attributes

In a locator for identifying a control during replay you can use a pre-defined set of locator attributes, for example *caption* and *automationId*, which depend on the technology domain. But you can also use every property, including dynamic properties, of a control as locator attribute. A list of available properties for a certain control can be retrieved with the `GetPropertyList` method. All returned properties can be used for identifying a control with a locator.

Note: You can use the `GetProperty` method to retrieve the actual value for a certain property of interest. You can then use this value in your locator.

Example

If you want to identify 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 Silk Test 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`.

Enhancing Tests

This section describes how you can enhance a test.

Calling Windows DLLs

This section describes how you can call DLLs. You can call a DLL either within the process of the Open Agent or in the application under test (AUT). This allows the reuse of existing native DLLs in test scripts.

DLL calls in the Open Agent are typically used to call global functions that do not interact with UI controls in the AUT.

DLL calls in the AUT are typically used to call functions that interact with UI controls of the application. This allows Silk4J to automatically synchronize the DLL call during playback.

Note: In 32-bit applications, you can call 32-bit DLLs, while in 64-bit applications you can call 64-bit DLLs. The Open Agent can execute both 32-bit and 64-bit DLLs.

Note: You can only call DLLs with a C interface. Calling of .NET assemblies, which also have the file extension .dll, is not supported.

Calling a Windows DLL from Within a Script

All classes and annotations that are related to DLL calling are located in the package `com.borland.silktest.jtf.dll`.

A declaration for a DLL starts with an interface that has a Dll attribute. The syntax of the declaration is the following:

```
@Dll("dllname.dll")
public interface DllInterfaceName {
  FunctionDeclaration
  [FunctionDeclaration]…
}
```

dllname	The name of or the full path to the DLL file that contains the functions you want to call from your Java scripts. Environment variables in the DLL path are automatically resolved. You do not have to use double backslashes (\\) in the path, single backslashes (\) are sufficient.
DllInterfaceName	The identifier that is used to interact with the DLL in a script.
FunctionDeclaration	A function declaration of a DLL function you want to call.

DLL Function Declaration Syntax

A function declaration for a DLL typically has the following form:

```
return-type function-name( [arg-list] )
```

For functions that do not have a return value, the declaration has the following form:

```
void function-name( [arg-list] )
```

return-type	The data type of the return value.
function-name	The name of the function.

arg-list A list of the arguments that are passed to the function.

The list is specified as follows:

```
data-type identifier
```

data-type The data type of the argument.

- To specify arguments that can be modified by a function or passed out from a function, use the `InOutArgument` and the `OutArgument` class.
- If you want the DLL function to set the value of the argument, use the `OutArgument` class.
- If you want to pass a value into the function, and have the function change the value and pass the new value out, use the `InOutArgument` class.

identifier The name of the argument.

DLL Calling Example

This example writes the text *hello world!* into a field by calling the `SendMessage` DLL function from `user32.dll`.

DLL Declaration:

```
@Dll("user32.dll")
public interface IUserDll32Functions {
  int SendMessageW(TestObject obj, int message, int wParam, Object lParam);
}
```

The following code shows how to call the declared DLL function in the AUT:

```
IUserDll32Functions user32Function =
DllCall.createInProcessDllCall(IUserDll32Functions.class, desktop);
TextField textField = desktop.find("//TextField");
user32Function.SendMessageW(textField, WindowsMessages.WM_SETTEXT, 0, "my
text");
```

Note: You can only call DLL functions in the AUT if the first parameter of the DLL function has the C data type HWND.

The following code shows how to call the declared DLL functions in the process of the Open Agent:

```
IUserDll32Functions user32Function =
DllCall.createAgentDllCall(IUserDll32Functions.class, desktop);
TextField textField = desktop.find("//TextField");
user32Function.SendMessageW(textField, WindowsMessages.WM_SETTEXT, 0, "my
text");
```

Note: The example code uses the `WindowsMessages` class that contains useful constants for usage with DLL functions that relate to Windows messaging.

Passing Arguments to DLL Functions

DLL functions are written in C, so the arguments that you pass to these functions must have the appropriate C data types. The following data types are supported:

int Use this data type for arguments or return values with the following data types:

- int
- INT
- long
- LONG
- DWORD
- BOOL
- WPARAM
- HWND

The Java type int works for all DLL arguments that have a 4-byte value.

long	Use this data type for arguments or return values with the C data types long and int64. The Java type long works for all DLL arguments that have an 8-byte value.
short	Use this data type for arguments or return values with the C data types short and WORD. The Java type short works for all DLL arguments that have a 2-byte value.
boolean	Use this data type for arguments or return values with the C data type bool.
String	Use this for arguments or return values that are Strings in C.
double	Use this for arguments or return values with the C data type double.
com.borland.silktest.jtf.Rect	Use this for arguments with the C data type RECT. Rect cannot be used as a return value.
com.borland.silktest.jtf.Point	Use this for arguments with the C data type POINT. Point cannot be used as a return value.
com.borland.silktest.jtf.TestObject	Use this for arguments with the C data type HWND. TestObject cannot be used as a return value, however you can declare DLL functions that return a HWND with an Integer as the return type.

Note: The passed TestObject must implement the com.borland.silktest.jtf.INativeWindow interface so that Silk4J is able to determine the window handle for the TestObject that should be passed into the DLL function. Otherwise an exception is thrown when calling the DLL function.

List Use this for arrays for user defined C structs. Lists cannot be used as a return value.

Note: When you use a List as an in/out parameter, the list that is passed in must be large enough to hold the returned contents.

Note: A C struct can be represented by a List, where every list element corresponds to a struct member. The first struct member is represented by the first element in the list, the second struct members is represented by the second element in the list, and so on.

Note: Any argument that you pass to a DLL function must have one of the preceding Java data types.

Passing Arguments that Can Be Modified by the DLL Function

An argument whose value will be modified by a DLL function needs to be passed either by using an InOutArgument, if the value can be changed, or by using an OutArgument.

Example

This example uses the `GetCursorPos` function of the `user32.dll` in order to retrieve the current cursor position.

DLL declaration:

```
@Dll( "user32.dll" )
public interface IUserDll32Functions {
  int GetCursorPos( OutArgument<Point> point);
}
```

Usage:

```
IUserDll32Functions user32Function =
DllCall.createAgentDllCall(IUserDll32Functions.class, desktop);

OutArgument<Point> point = new OutArgument<Point>(Point.class);
user32Function.GetCursorPos(point);

System.out.println("cursor position = " + point.getValue());
```

Passing String Arguments to DLL Functions

Strings that are passing into a DLL function or that are returned by a DLL function are treated by default as Unicode Strings. If your DLL function requires ANSI String arguments, use the `CharacterSet` property of the `DllFunctionOptions` attribute.

Example

```
@Dll( "user32.dll" )
public interface IUserDll32Functions {
  @FunctionOptions(characterSet=DllCharacterSet.Ansi)
  int SendMessageA(TestObject obj, int message, int wParam,
Object lParam);
}
```

Passing a String back from a DLL call as an OutArgument works per default if the String's size does not exceed 256 characters length. If the String that should be passed back is longer than 256 characters, you need to pass an InOurArgument with a String in that is long enough to hold the resulting String.

Example

Use the following code to create a String with 1024 blank characters:

```
char[] charArray = new char[1024];
Arrays.fill(charArray,' ');
String longEmptyString = new String(charArray);
```

Pass this InOutArgument as an argument into a DLL function and the DLL function will pass back Strings of up to 1024 characters of length.

When passing a String back from a DLL call as a function return value, the DLL should implement a DLL function called `FreeDllMemory` that accepts the C String pointer returned by the DLL function and that frees the previously allocated memory. If no such function exists the memory will be leaked.

Aliasing a DLL Name

If a DLL function has the same name as a reserved word in Java, or the function does not have a name but an ordinal number, you need to rename the function within your declaration and use the alias statement to map the declared name to the actual name.

Example

For example, the `goto` statement is reserved by the Java compiler. Therefore, to call a function named `goto`, you need to declare it with another name, and add an alias statement, as shown here:

```
@Dll("mydll.dll")
public interface IMyDllFunctions {
  @FunctionOptions(alias="break")
  void MyBreak();
}
```

Conventions for Calling DLL Functions

The following calling conventions are supported when calling DLL functions:

- __stdcall
- __cdecl

The *__stdcall* calling convention is used by default when calling DLL functions. This calling convention is used by all Windows API DLL functions.

You can change the calling convention for a DLL function by using the `CallingConvention` property of the `DllFunctionOptions` annotation.

Example

The following code example declares a DLL function with the *__decl* calling convention:

```
@Dll("msvcrt.dll")
public interface IMsVisualCRuntime {
  @FunctionOptions(callingConvention=CallingConvention.Cdecl)
  double cos(double inputInRadians);
}
```

Custom Controls

You can create custom classes for custom controls for which Silk4J does not offer any dedicated support. Creating custom classes offers the following advantages:

- Better locators for scripts.
- An easy way to write reusable code for the interaction with the custom control.

Example

Suppose that a custom tab control is recognized by Silk4J as the generic class `Control`. Using the custom control support of Silk4J has the following advantages:

Better object recognition because the custom control class name can be used in a locator.

Many objects might be recognized as `Control`. The locator requires an index to identify the specific object. For example, the object might be identified by the locator `//Control[13]`. When you create a custom class for this control, for example the class `MyTabControl`, you can use the locator `//MyTabControl`. By creating the custom class, you do not require the high index, which would be a fragile object identifier if the application under test changed.

You can implement reusable playback actions for the control in scripts.

Without using the custom classes, when you want to select a tab in your custom tab controls, you can write code like the following:

```
tabControl.TextClick("<TabName>")
```

When you are using custom classes, you can encapsulate the behavior for selecting a tab into a method by adding the following code to your custom class, which is the class that gets generated when you specify the custom control in the user interface:

```
public void SelectTab(string tabText) {
  TextClick(tabText);
}
```

The custom class looks like the following:

```
package <package>;

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

public class MyTabControl {

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

  public void SelectTab(string tabText) {
    TextClick(tabText);
  }

}
```

You can now use the newly created method `SelectTab` in a script like the following:

```
tabControl.SelectTab("<TabName>")
```

Supporting a Custom Control

To create a custom class for a custom control for which Silk4J does not offer any dedicated support.

1. Click **Silk4J** > **Manage Custom Controls**. The **Manage Custom Controls** dialog box opens.

2. In the **Silk4J Custom Controls Output Package** field, type in a name or click **Browse** to select the package that will contain the custom control.
3. Click on the tab of the technology domain for which you want to create a new custom class.
4. Click **Add**.
5. Click one of the following:
   - Click **Identify new custom control** to directly select a custom control in your application with the **Identify Object** dialog box.
   - Click **Add new custom control** to manually add a custom control to the list.

   A new row is added to the list of custom controls.
6. In the **Silk Test base class** column, select an existing base class from which your class will derive.

   This class should be the closest match to your type of custom control.
7. In the **Silk Test class** column, enter the name to use to refer to the class.

   This is what will be seen in locators. For example: `//MyTabControl` instead of `//Control[13]`.

   Note: After you add a valid class, it will become available in the **Silk Test base class** list. You can then reuse it as a base class.
8. In the **Custom control class name** column, enter the fully qualified class name of the class that is being mapped.

   For example: `Infragistics.Win.UltraWinTabControl.UltraTabControl`.
9. *Only for Win32 applications:* In the **Class mapping** column, set the class mapping to true to map the name of a custom control class to the name of a standard Silk Test class.

   When you map the custom control class to the standard Silk Test class, you can use the functionality supported for the standard Silk Test class in your test.
10. Click **OK**.
11. *Only for scripts:*
    a) Add custom methods and properties to your class for the custom control.
    b) Use the custom methods and properties of your new class in your script.

       Note: The custom methods and properties are not recorded.

Note: Do not rename the custom class or the base class in the script file. Changing the generated classes in the script might result in unexpected behavior. Use the script only to add properties and methods to your custom classes. Use the **Manage Custom Controls** dialog box to make any other changes to the custom classes.

Custom Controls Options

Silk4J > **Manage Custom Controls**.

In the **Silk4J Custom Controls Output Package**, define the package into which the new custom classes should be generated.

The following **Custom Controls** options are available:

Option	Description
Silk Test base class	Select an existing base class to use that your class will derive from. This class should be the closest match to your type of custom control.
Silk Test class	Enter the name to use to refer to the class. This is what will be seen in locators. For example: `//MyTabControl` instead of `//Control[13]`.

Option	Description
Custom control class name	Enter the fully qualified class name of the class that is being mapped. For example: `Infragistics.Win.UltraWinTabControl.UltraTabControl`.
Class mapping	This option is available only for Win32 applications. Set the class mapping to true to map the name of a custom control class to the name of a standard Silk Test class. When you map the custom control class to the standard Silk Test class, you can use the functionality supported for the standard Silk Test class in your test.

Note: After you add a valid class, it will become available in the **Silk Test base class** list. You can then reuse it as a base class.

Windows Accessibility

This section describes how you can use Windows Accessibility (Accessibility) to ease the recognition of objects at the class level.

Using Accessibility

Win32 uses the Accessibility support for controls that are recognized as generic controls. When Win32 locates a control, it tries to get the accessible object along with all accessible children of the control.

Objects returned by Accessibility are either of the class `AccessibleControl`, `Button` or `CheckBox`. `Button` and `Checkbox` are treated specifically because they support the normal set of methods and properties defined for those classes. For all generic objects returned by Accessibility the class is `AccessibleControl`.

Example

If an application has the following control hierarchy before Accessibility is enabled:

- Control
  - Control
- Button

When Accessibility is enabled, the hierarchy changes to the following:

- Control
  - Control
    - Accessible Control
    - Accessible Control
      - Button
- Button

Enabling Accessibility

If you are testing a Win32 application and Silk4J cannot recognize objects, you should first enable Accessibility. Accessibility is designed to enhance object recognition at the class level.

To enable Accessibility:

1. Click **Edit Options**. The **Script Options** dialog box opens.

2. Click **Advanced**.
3. Select the **Use Microsoft Accessibility** option. Accessibility is turned on.

Dynamic Invoke

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

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

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

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

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

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

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

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

Text Recognition Support

Text recognition methods enable you to conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *text clicks* instead of coordinate-based clicks to click on a specified text string within a control.

For example, you can simulate selecting the first cell in the second row of the following table:

CustomerName	FirstOrder	ID	IsActive	CreditCard
Bob Villa	01.01.2008	0	☑	MasterCard
Brian Miller	02.01.2008	1	☐	Visa
Caral Rudd	03.01.2008	2	☑	American Ex...
Dan Rundgren	04.01.2008	3	☐	MasterCard
Devie Yingstein	05.01.2008	4	☑	Visa

Specifying the text of the cell results in the following code line:

```
table.textClick("Brian Miller");
```

Text recognition methods are supported for the following technology domains:

- Win32.
- WPF.
- Windows Forms.
- Java SWT and Eclipse.
- Java AWT/Swing.

Note: For Java Applets, and for Swing applications with Java versions prior to version 1.6.10, text recognition is supported out-of-the-box. For Swing applications with Java version 1.6.10 or later, you have to add the following command-line element when starting the application:

```
-Dsun.java2d.d3d=false
```

For example:

```
javaw.exe -Dsun.java2d.d3d=false -jar mySwingApplication.jar
```

- xBrowser.

Text recognition methods

The following methods enable you to interact with the text of a control:

`TextCapture`	Returns the text that is within a control. Also returns text from child controls.
`TextClick`	Clicks on a specified text within a control.
`TextRectangle`	Returns the rectangle of a certain text within a control or a region of a control.
`TextExists`	Determines whether a given text exists within a control or a region of a control.

Text click recording

Text click recording is enabled by default. To disable text click recording, click **Silk4J** > **Edit Options** > **Recording** and uncheck the **OPT_RECORD_TEXT_CLICK** check box.

When text click recording is enabled, Silk4J records `TextClick` methods instead of clicks with relative coordinates. Use this approach for controls where `TextClick` recording produces better results than normal coordinate-based clicks. You can insert text clicks in your script for any control, even if the text clicks are not recorded.

If you do not whish to record a `TextClick` action, you can turn off text click recording and record normal clicks.

The text recognition methods prefer whole word matches over partially matched words. Silk4J recognizes occurrences of whole words previously than partially matched words, even if the partially matched words are displayed before the whole word matches on the screen. If there is no whole word found, the partly matched words will be used in the order in which they are displayed on the screen.

Example

The user interface displays the text *the hostname is the name of the host*. The following code clicks on *host* instead of *hostname*, although *hostname* is displayed before *host* on the screen:

```
control.textClick("host");
```

The following code clicks on the substring *host* in the word *hostname* by specifying the second occurrence:

```
control.textClick("host", 2);
```

Grouping Silk4J Tests

You can use the `SilkTestCategories` class to run Silk4J tests, write TrueLogs, and filter or group tests with annotations. Define categories of test classes to group the Silk4J tests into these categories, and to run only the tests that are included in a specified category or a subtype of that category.

To include a Silk4J test in a category, use the `@IncludeCategory` annotation.

Using the category `SilkTestCategories` class enables you to write TrueLogs for the Silk4J tests included in the category. You can also use the `SilkTestSuite` class to write TrueLogs. For additional information, see *Replaying a Test Method from the Command Line*.

Example

The following code sample shows how you can execute the Silk4J tests that are included in a category:

```
public interface FastTests {
}

public interface SlowTests {
}

public static class A {
  @Test
  public void a() {
    fail();
  }

  @Category(SlowTests.class)
  @Test
  public void b() {
  }
}

  @Category( { SlowTests.class, FastTests.class })
  public static class B {
    @Test
    public void c() {
  }
}

@RunWith(SilkTestCategories.class)
@IncludeCategory(SlowTests.class)
@SuiteClasses( { A.class, B.class })
// Note: SilkTestCategories is a kind of Suite
public static class SlowTestSuite {
}
```

Inserting a Result Comment in a Script

You can add result comments to a test script to provide supplemental information about the test. During the execution of the test, the result comments are added to the TrueLog file of the test.

You can add different comment types for information, warnings, and errors. The following code sample shows an example for each comment type:

```
desktop.logInfo("This is a comment!");
desktop.logWarning("This is a warning!");
desktop.logError("This is an error!");
```

Concepts

This section includes the conceptual overview topics for Silk4J.

Silk Test Open Agent

The Silk Test Open Agent is the software process that translates the commands in your scripts into GUI-specific commands. In other words, the Open Agent drives and monitors the application that you are testing.

One Agent can run locally on the host machine. In a networked environment, any number of Agents can replay tests on remote machines. However, you can record only on a local machine.

Starting the Silk Test Open Agent

Before you can create a test or run a sample script, the Silk Test Open Agent must be running. Typically, the Agent starts when you launch the product. If you must manually start the Open Agent, perform this step.

Click **Start** > **Programs** > **Silk** > **Silk Test** > **Tools** > **Silk Test Open Agent** . The Silk Test Open Agent icon displays in the system tray.

Open Agent Port Numbers

When the Open Agent starts, a random, available port is assigned to Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4J, Silk4NET, and the application that you are testing. The port numbers are registered on the information service. Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J contact the information service to determine the port to use to connect to the Open Agent. The information service communicates the appropriate port, and Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J connects to that port. Communication runs directly between Silk Test Workbench, Silk Test Classic, Silk Test Recorder, Silk4NET, or Silk4J and the Agent.

By default, the Open Agent communicates with the information service on port 22901. You can configure additional ports for the information service as alternate ports that work when the default port is not available. By default, the information service uses ports 2966, 11998, and 11999 as alternate ports.

Typically, you do not have to configure port numbers manually. However, if you have a port number conflict or an issue with a firewall, you must configure the port number for that machine or for the information service. You can use a different port number for a single machine or you can use the same available port number for all your machines.

Configuring the Port that Clients Use to Connect to the Information Service

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

The default port of the information service is 22901. When you can use the default port, you can type `hostname` without the port number for ease of use. If you do specify a port number, ensure that it matches the value for the default port of the information service or one of the additional information service ports. Otherwise, communication will fail.

If necessary, you can change the port number that all clients use to connect to the information service.

1. Navigate to the `infoservice.properties.sample` file and open it.

   This file is located in `C:\Documents and Settings\All Users\Application Data\Silk\Silk Test\conf`, where "`C:\Documents and Settings\All Users`" is equivalent to the content of the ALLUSERSPROFILE environment variable, which is set by default on Windows systems.

   This file contains commented text and sample alternate port settings.
2. Change the value for the appropriate port.

   Typically, you configure the information service port settings to avoid problems with a firewall by forcing communication on a specific port.

   Port numbers can be any number from 1 to 65535.

   - `infoservice.default.port` – The default port where the information service runs. By default, this port is set to 22901.
   - `infoservice.additional.ports` – A comma separated list of ports on which the information service runs if the default port is not available. By default, ports 2966, 11998, and 11999 are set as alternate ports.
3. Save the file as `infoservice.properties`.
4. If you are using Silk Test Recorder and you changed the information service port, specify the new port number in the Silk Test Recorder **Global Preferences** dialog box.
5. Restart the Silk Test Open Agent, Silk Test Classic, Silk Test Workbench, Silk Test Recorder, Silk4J, Silk4NET, and the application that you want to test.

Configuring the Port that Silk Test Workbench, Silk Test Classic, Silk4J, Silk4NET, or the Test Application Uses to Connect to the Open Agent

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

If necessary, change the port number that Silk Test Workbench, Silk Test Classic, Silk4J, Silk4NET, or the application that you want to test uses to connect to the Open Agent.

1. Navigate to the `agent.properties.sample` file and open it.

   By default, this file is located at: `%APPDATA%\Silk\Silk Test\conf`, which is typically `C:\Documents and Settings\<user name>\Application Data\Silk\Silk Test\conf` where <*user name*> equals the current user name.
2. Change the value for the appropriate port.

   Typically, you configure port settings to resolve a port conflict.

   Note: Each port number must be unique. Ensure that the port numbers for the Agent differ from the information service port settings.

   Port numbers can be any number from 1 to 65535.

   Port settings include:

   - `agent.vtadapter.port` – Controls communication between Silk Test Workbench and the Open Agent when running tests.
   - `agent.xpmodule.port` – Controls communication between Silk Test Classic and the Agent when running tests.
   - `agent.autcommunication.port` – Controls communication between the Open Agent and the application that you are testing.

- `agent.rmi.port` – Controls communication with the Open Agent and Silk4J.
- `agent.ntfadapter.port` – Controls communication with the Open Agent and Silk4NET.

Note: The ports for Adobe Flex testing are not controlled by this configuration file. The assigned port for Flex application testing is 6000 and increases by 1 for each Flex application that is tested. You cannot configure the starting port for Flex testing.

3. Save the file as `agent.properties`.
4. Restart the Silk Test Open Agent, Silk Test Classic, Silk Test Workbench, Silk Test Recorder, Silk4J, Silk4NET, and the application that you want to test.

Configuring the Port that Silk Test Classic, Silk4J, or Silk4NET Use to Connect to Silk Test Recorder

Before you begin this task, stop the Silk Test Open Agent.

Typically, you do not have to configure port numbers manually. The information service handles port configuration automatically. Use the default port of the information service to connect to the Agent. Then, the information service forwards communication to the port that the Agent uses.

If necessary, change the port number that Silk Test Classic, Silk4J, or Silk4NET uses to connect to Silk Test Recorder.

1. Navigate to the `recorder.properties.sample` file and open it.

   By default, this file is located at: `%APPDATA%\Silk\Silk Test\conf`, which is typically `C:\Documents and Settings\<user name>\Application Data\Silk\Silk Test\conf` where *<user name>* equals the current user name.
2. Change the `recorder.api.rmi.port` to the port that you want to use.

   Port numbers can be any number from 1 to 65535.

   Note: Each port number must be unique. Ensure that the port numbers for the Agent settings differ from the recorder and the information service port settings.
3. Save the file as `recorder.properties`.
4. Restart the Silk Test Open Agent, Silk Test Classic, Silk Test Workbench, Silk Test Recorder, Silk4J, Silk4NET, and the application that you want to test.

Base State

An application's base state is the known, stable state that you expect the application to be in before each test case begins execution, and the state the application can be returned to after each test case has ended execution. This state may be the state of an application when it is first started.

When you create a class for a Web application or standard configuration, Silk4J automatically creates a base state.

Base states are important because they ensure the integrity of your tests. By guaranteeing that each test case can start from a stable base state, you can be assured that an error in one test case does not cause subsequent test cases to fail.

Silk4J automatically ensures that your application is at its base state during the following stages:

- Before a test runs
- During the execution of a test
- After a test completes successfully

Modifying the Base State

You can change the executable location, working directory, locator, or URL of the base state if necessary. For example, if you want to launch tests from a production Web site that were previously tested on a testing Web site, change the base state URL and the tests will run in the new environment.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.
2. Click **Edit Base State**. The **Edit Base State** dialog box opens.
3. In the **Executable** text box, type the executable name and file path of the application that you want to test.
   For example, you might type `C:\Program Files\Internet Explorer\IEXPLORE.EXE` to specify Internet Explorer.
4. To use a command line pattern in combination with the executable file, in the **Command Line Arguments** text box, type the command line pattern.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, `*\javaw.exe` is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you want is enabled for testing. For example, if the command line of the application ends with **com.example.MyMainClass** you might want to use ***com.example.MyMainClass** as the command line pattern.
5. If the application that you want to test depends on a supplemental directory, specify a directory in the **Working Directory** text box.

   For example, if you use a batch file to start a Java application, the batch file may reference a JAR file that relies on a relative path. In this case, specify a working directory to reconcile the relative path.
6. In the **Locator** text box, type the XPath locator string that identifies the main window of the application.
7. If you are testing a Web site, in the **Url** text box, type the Web address for the Web page to launch when a test begins.
8. Click **OK**.

Application Configuration

An application configuration defines how Silk4J connects to the application that you want to test. Silk4J automatically creates an application configuration when you create the base state. However, at times, you might need to modify, remove, or add an additional application configuration. For example, if you are testing an application that modifies a database and you use a database viewer tool to verify the database contents, you must add an additional application configuration for the database viewer tool.

An application configuration includes the:

- Executable pattern

  All processes that match this pattern are enabled for testing. For example, the executable pattern for Notepad is `\*notepad.exe`. All processes whose executable is named `notepad.exe` and that are located in any arbitrary directory are enabled
- Command line pattern

  The command line pattern is an additional pattern that is used to constrain the process that is enabled for testing by matching parts of the command line arguments (the part after the executable name). An application configuration that contains a command line pattern enables only processes for testing that match both the executable pattern and the command line pattern. If no command-line pattern is defined, all processes with the specified executable pattern are enabled. Using the command line is especially

useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, `* \javaw.exe` is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you want is enabled for testing. For example, if the command line of the application ends with **com.example.MyMainClass** you might want to use ***com.example.MyMainClass** as the command line pattern.

Modifying an Application Configuration

An application configuration defines how Silk4J connects to the application that you want to test. Silk4J automatically creates an application configuration when you create the base state. However, at times, you might need to modify, remove, or add an additional application configuration. For example, if you are testing an application that modifies a database and you use a database viewer tool to verify the database contents, you must add an additional application configuration for the database viewer tool.

1. Click the drop-down arrow next to the Silk Test toolbar icon and choose **Edit Application Configurations**. The **Edit Application Configurations** dialog box opens and lists the existing application configurations.
2. To add an additional application configuration, click **Add**.

   The **New Application Configuration** wizard opens.

   a) Select the type of the application that you want to test and click **Next**.
   b) Click the configuration that corresponds with the application that you want to test.

      If the application that you want to test does not appear in the list, uncheck the **Hide processes without caption** check box. This option, checked by default, is used to filter only those applications that have captions.
   c) If you have selected to test a Web application, select whether you want to use an existing browser instance or start a new one.
   d) Click **Finish**. The executable pattern of the application is added to the list of application configurations in the **Edit Application Configurations** dialog box.
3. To remove an application configuration, click **Remove** next to the appropriate application configuration.
4. To edit an application configuration, click **Edit Base State**. The **Edit Base State** dialog box opens.
5. Click **OK**.

Application Configuration Errors

When the program cannot attach to an application, the following error message opens:
Failed to attach to application <Application Name>. For additional information, refer to the Help.

In this case, one or more of the issues listed in the following table may have caused the failure:

Issue	Reason	Solution
Time out	• The system is too slow. • The size of the memory of the system is too small.	Use a faster system or try to reduce the memory usage on your current system.
User Account Control (UAC) fails	You have no administrator rights on the system.	Log in with a user account that has administrator rights.
Command-line pattern	The command-line pattern is too specific. This issue occurs especially for Java. The replay may not work as intended.	Remove ambiguous commands from the pattern.

Desktop Class

The Desktop class is the entry point for accessing the application that you are testing and the Open Agent.

The Desktop class represents a desktop of a specific machine. Therefore, the desktop is associated with one agent, which runs on that machine.

Contacting Micro Focus

Micro Focus is committed to providing world-class technical support and consulting services. Micro Focus provides worldwide support, delivering timely, reliable service to ensure every customer's business success.

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

Visit *http://supportline.microfocus.com/assistedservices.asp* to communicate directly with Micro Focus SupportLine to resolve your issues or email supportline@microfocus.com.

Visit Micro Focus SupportLine at *http://supportline.microfocus.com* for up-to-date support news and access to other support information. First time users may be required to register to the site.

Information Needed by Micro Focus SupportLine

When contacting Micro Focus SupportLine, please include the following information if possible. The more information you can give, the better Micro Focus SupportLine can help you.

- The name and version number of all products that you think might be causing an issue.
- Your computer make and model.
- System information such as operating system name and version, processors, and memory details.
- Any detailed description of the issue, including steps to reproduce the issue.
- Exact wording of any error messages involved.
- Your serial number.

To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Index

.NET support
    Silverlight 43
64-bit applications
    support 62

A

Accessibility
    enabling 82
    overview 82
    using 82
ActiveX
    invoking methods 61
    overview 61
Adobe Flex
    attributes 25, 68
    class definition file 34
    custom controls 26, 30, 34
    defining custom controls 26
    implementing custom controls 31, 34
    invoking methods 35
    invoking methods for custom controls 29
    overview 24
    security settings 25
    styles 24
agents
    configuring ports 86, 88
    overview 86
    port numbers 86
    starting 86
AJAX applications
    script hangs 59
Ant
    replaying test methods 15
API playback
    compared to native playback 54
application configurations
    adding 90
    definition 89
    errors 90
    modifying 90
    removing 90
    troubleshooting 90
attribute types
    Adobe Flex 25, 68
    Java AWT 36, 68
    Java Swing 36, 68
    Java SWT 39, 69
    overview 68
    SAP 42, 70
    Silverlight 44, 70
    Web applications 55, 71
    Windows 49, 72
    Windows Forms 47, 72
    xBrowser 55, 71

B

base state
    definition 88
    modifying 89
browser
    overview 53
browser type
    GetProperty 58
browsers
    setting preferences 18
browsertype
    using 58

C

calling dlls
    example 76
    Java 75
    scripts 75
Chrome
    cross-browser scripts 58
    prerequisites 56
classes
    exposing 20
    ignoring 20
client/server applications
    Windows API-based 40
command line
    running test methods 14
contact information 92
custom attributes
    setting 19
custom controls
    automation support 30
    creating custom classes 80
    defining 34
    dialog box 81
    dynamically invoking Adobe Flex 29
    overview 79
    supporting 80
    testing 30
    testing Flex 26
Customer Care 92

D

Desktop class 91
Dialog
    not recognized 60
dlls
    aliasing names 79
    calling conventions 79
    calling from Java 75
    calling from within a script 75
    example call 76
    function declaration syntax 75
    passing arguments that can be modified to functions 78
    passing arguments to functions 76
    passing string arguments to functions 78
DOM functions 54
downloads 92

dynamic invoke
overview 83
dynamic locator attributes
about 74
dynamic object recognition
overview 64
sample queries 66
dynamically invoking methods
ActiveX 61
Adobe Flex 35
Adobe Flex custom controls 29
Java AWT 36, 39
Java Swing 36, 39
Java SWT 36, 39
SAP 42
Silverlight 45
Visual Basic 61
Windows Forms 47
Windows Presentation Foundation (WPF) 51
dynamicInvoke
ActiveX 61
Adobe Flex 35
Java AWT 36, 39
Java Swing 36, 39
SAP 42
Silverlight 45
Visual Basic 61
Windows Forms 47
Windows Presentation Foundation (WPF) 51

E

embedded browser
testing 53
enabling TrueLog
TrueLog Explorer 15
exposing WPF classes 20

F

FAQs
xBrowser 57
Firefox
cross-browser scripts 58
locators 59
testing 53
firewalls
port numbers 86
resolving conflicts 86
Flash player
security settings 25
Flex
attributes 25, 68
class definition file 34
custom controls
defining 26, 34
implementing 31
defining custom controls 26
implementing custom controls 31, 34
invoking methods 35
invoking methods for custom controls 29
overview 24
security settings 25
styles 24

G

Google Chrome
limitations 56
prerequisites 56

I

identifying controls
dynamic locator attributes 74
Locator Spy 68
ignoring classes 20
information service
communication with Open Agent 86
innerHTML 57
innerText 57, 58
Internet Explorer
cross-browser scripts 58
link.select focus issue 59
locators 59
testing 53
Internet Explorer 10
unexpected Click behavior 60
invalidated-handle error
troubleshooting 60
invoke
ActiveX 61
Java AWT 36, 39
Java SWT 36, 39
SAP 42
Swing 36, 39
Visual Basic 61
Windows Forms 47
Windows Presentation Foundation (WPF) 51
InvokeMethods
ActiveX 61
Adobe Flex 35
Java AWT 36, 39
Java Swing 36, 39
SAP 42
Silverlight 45
Visual Basic 61
Windows Forms 47
Windows Presentation Foundation (WPF) 51

J

Java AWT
attribute types 36, 68
attributes 36, 68
invoking methods 36, 39
overview 35
Java AWT/Swing
priorLabel 38
Java Network Launching Protocol (JNLP)
configuring applications 37
Java Swing
attributes 36, 68

invoking methods 36, 39
overview 35
Java SWT
attribute types 39, 69
custom attributes 19
invoking methods 36, 39
overview 38
JNLP
configuring applications 37

L

locator attributes
dynamic 74
Rumba controls 41, 70
Silverlight controls 44, 70
WPF controls 49, 72
Locator Spy
overview 68
locators
attributes 18
incorrect in xBrowser 59
xBrowser 59

M

Microsoft UI Automation (MSUIA)
overview 48
using deprecated MSUIA values 52
mouse move actions 17
MSUIA
overview 48
using deprecated MSUIA values 52

N

native playback
compared to API playback 54
native user input
overview 54

O

Open Agent
configuring ports 86, 88
location 86
overview 86
port numbers 86
starting 86
OPT_ALTERNATE_RECORD_BREAK
options 17
OPT_ENSURE_ACTIVE_OBJDEF 21
OPT_RECORD_MOUSEMOVE_DELAY
options 17
OPT_RECORD_MOUSEMOVES
options 17
OPT_RECORD_SCROLLBAR_ABSOLUT
options 17
OPT_REPLAY_MODE 21
OPT_WAIT_RESOLVE_OBJDEF 20
OPT_WAIT_RESOLVE_OBJDEF_RETRY 20
OPT_XBROWSER_RECORD_LOWLEVEL 18
OPT_XBROWSER_SYNC_EXCLUDE_URLS 20
OPT_XBROWSER_SYNC_MODE 20
OPT_XBROWSER_SYNC_TIMEOUT 20

P

page synchronization for xBrowser 53
port conflicts
resolving 88
ports
Open Agent 86, 88
preferences
turning off error messages 23
prerequisites
Google Chrome 56
priorLabel
Java AWT/Swing technology domain 38
Win32 technology domain 41
Product Support 92
product updates 9

Q

Quick Start tutorial
introduction 10
replaying test 13
test class 11
test method 12

R

Record Break keys 17
recording
preferences 17
replay
Dialog not recognized 60
options 21
result comments
adding to scripts 85
Rumba
locator attributes 41, 70
Rumba locator attributes
identifying controls 41, 70

S

SAP
attribute types 42, 70
custom attributes 19
invoking methods 42
overview 42
scripts
adding result comments 85
specifying options 17
scroll events 17
serial number 92
SetText 18
shortcut key combination 17
Silk4J
best practices 63
creating project 10

quick start tutorial 10
Silk4J tests
grouping 84
Silverlight
attribute types 44, 70
invoking methods 45
locator attributes 44, 70
overview 43
support 43
troubleshooting 46
specifying options
scripts 17
styles
in Flex applications 24
SupportLine 92
Swing
attributes 36, 68
configuring JNLP applications 37
invoking methods 36, 39
overview 35
SWT
overview 38
synchronization options 20

T

test cases
sample queries 66
test class
creating 11
test method
adding 12
replaying 13
running 14
test methods
running 15
running from command line 14
running from Eclipse 14
tests
enhancing 75
text click recording
overview 83
text recognition
overview 83
textContents 57
timestamps 59
troubleshooting
Silverlight 46
troubleshooting XPath 67
TrueLog
configuring 17
enabling 15, 17
replacement characters for non-ASCII 16
SilkTestCategories class 84
wrong non-ASCII characters 16
TrueLog Explorer
configuring 17
enabling 17
enabling TrueLog 15
tutorial
quick start 10
TypeKeys 18

U

unexpected Click behavior
Windows Internet Explorer 60
updates 9

V

Visual Basic
invoking methods 61
overview 61

W

WaitForProperty method
invalidated-handle error 60
Web applications
custom attributes 19
overview 53
supported attributes 55, 71
WebSync 92
welcome 6
Win32
priorLabel 41
Windows
64-bit application support 62
attribute types 49, 72
Windows Accessibility
overview 82
Windows API-based
64-bit application support 62
overview 40
Windows applications
custom attributes 19
Windows Forms
64-bit application support 62
attribute types 47, 72
custom attributes 19
invoking methods 47
overview 47
Windows Internet Explorer
misplaced rectangles 59
Windows Internet Explorer 10
unexpected Click behavior 60
Windows Presentation Foundation (WPF)
64-bit application support 62
custom controls 50
invoking methods 51
locator attributes 49, 72
overview 48
using deprecated MSUIA values 52
WPFItemsControl class 50
works order number 92
WPF
64-bit application support 62
custom controls 50
exposing classes 20
invoking methods 51
locator attributes 49, 72
overview 48
using deprecated MSUIA values 52
WPFItemsControl class 50

WPF applications
custom attributes 19
WPF locator attributes
identifying controls 49, 72
writing TrueLogs
SilkTestCategories class 84

X

xBrowser
API and native playback 54
attribute types 55, 71
browser type distinctions 58
class and style not in locators 60
cross-browser scripts 58
custom attributes 19
Dialog not recognized 60
DomClick not working like Click 59
exposing functionality 60
FAQs 57
FieldInputField.DomClick not opening dialog 59
font type verification 57
innerText not being used in locators 58
link.select focus issue 59
mouse move recording 59
navigating to new pages 59
overview 53
page synchronization 53
playback options 54
recording an incorrect locator 59
recording locators 59
textContents, innerText, innerHTML 57
timestamps 59
Windows Internet Explorer misplaces rectangles 59
XPath
basic concepts 64
creating query strings 68
overview 64
samples 66
supported subset 65
troubleshooting 67